Performance Management in Clustered Edge Architectures Using Particle Swarm Optimization

Shelernaz Azimi\textsuperscript{1(✉)}, Claus Pahl\textsuperscript{1}, and Mirsaeid Hosseini Shirvani\textsuperscript{2(✉)}

\textsuperscript{1} Free University of Bozen-Bolzano, Bolzano, Italy
{seyedehshelernaz.azimi,claus.pahl}@unibz.it
\textsuperscript{2} Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

Abstract. Recently distributed computing capacities are brought to the edge of the Internet, permitting Internet-of-Things applications to process calculation all the more locally and subsequently more productively and this has brought a totally different scope of apparatuses and usefulness. This instrument can The most significant characterizing highlights of edge processing are low latency, location awareness, wide geographic distribution, versatility, support for countless nodes, etc. We want to likely limit the latency and delay in edge-based structures. We center around a progressed compositional setting that considers communication and processing delays and the management effort notwithstanding a real request execution time in an operational efficiency situation. Our design is based on multi-cluster edge layer with nearby autonomous edge node clusters. We will contend that particle swarm optimization as a bio-motivated optimization approach is a perfect candidate for distributed IoT load handling in self-managed edge clusters. By designing a controller and utilizing a particle swarm optimization algorithm, we show that delay and end-to-end latency can be reduced.

Keywords: Internet of Things · Edge computing · Cloud computing · Edge cluster · Particle swarm optimization · Latency · Delay · Performance

1 Introduction

With the growing internet, we are seeing also devices in our vicinity in many everyday activities. So far, the fundamental use of the internet has been the utilization by human individuals for a specific function. We are now seeing another type of internet known as the Internet of Things. The Internet of Things gives the capacity to associate items over internet-based connectivity.

The IoT is the foundation to convey energizing new services. A framework is needed without which there would be no service. We need to compare the Internet of Things and the traditional internet for a better comprehension of the problem. Has the internet alone presented to us a service? To tell the truth, the internet has given a stage to offer appealing types of services, for example, the web, email and online services by interfacing PCs, cell phones and other networkable equipment. So is the Internet of Things. By connecting items to the network, it gives a scope of appealing and helpful services,
for example, remote control, reporting and alerting, accident prevention, cost reduction and smart automation noted. IoT alludes to a situation wherein everything, regardless of whether human, creature, or lifeless, has a unique identifier or Internet protocol fit for recognizing, controlling, transmitting, and transmitting information to each other and to the pertinent database. Information gathered from items will be visible through different apparatuses, for example, cell phones, kinds of PCs and tablets. When IoT is implemented, data can be moved between various entities. IoT is the consequence of the assembly and advancement of the three components of the Internet, wireless technology and microelectronic frameworks. The fundamental result of IoT is the interconnection of devices [17].

Edge computing gives a transitional layer to computation and storage at the ’edge’ of the system, regularly between Internet-of-Things devices and centralized data center clouds [18,28]. Edge computing guarantees better performance through lower latency since calculation is drawn closer to where the application sits. Lessening the transfer of information by keeping away the exchange of huge volumes of information to distant clouds has likewise the side effect of reducing security risks. Localization here is the key standard. Edge computing is suitable for applications that require high response speed, low latency, and real time. All these studies show that resource sharing provides low latency, better scalability, distributed processing, better security, crash tolerance and privacy to provide better haze infrastructure.

Propagation delay refers to the amount of time it takes for the first bit to travel over a link between sender and receiver, whereas latency refers to the total amount of time it takes to send an entire message.

Execution and load management in edge architectures has been tended to in the past [5,22], yet frequently the structures alluded to do not mirror the regularly geographically distributed nature of edge computing. We extend here on works like [9,34] that have considered single autonomous clusters as it were. We propose here an answer for a multi-cluster arrangement, where each cluster works semi-independently, just being composed by an orchestrator that oversees load distribution. Another direction that we include is a sensible impression of performance concerns. In our performance model we consider delays cause by correspondence and queueing just as processing times of controllers and edge execution nodes into a comprehensive end-to-end latency idea that understands the response time from the requestor’s point of view.

Accordingly, our methodology expands the state-of-the-art by joining an end-to-end latency optimization framework with a multi-cluster edge architecture. We propose Particle Swarm Optimization (PSO) for the optimization here which extends our previous work in [3]. PSO is a bio-motivated evolutionary optimization technique [30] reasonable to organize between autonomous elements, for example, edge clusters in our case. PSO recognizes individual (here local clusters) best fitness and global (here cross-cluster) best fitness in the distribution of load to clusters and their nodes - which we use to optimize latency. Our orchestrator takes local cluster computation, yet additionally, whenever required, incorporated cloud processing as choices on board. We extend earlier work in [3] by providing more technical implementation detail, the comparison with related work and broaden the evaluation.
We show the viability of our performance optimization by contrasting it and other normal load distribution systems and settings.

The paper is organized as follows. In Sect. 2, particle swarm optimization will be introduced. In Sect. 3, our method for multi-cluster performance optimization of load distribution will be presented. In Sect. 4, the implementation will be evaluated. In the following section, the related work will be discussed. Finally, conclusions and suggestions for future work will be presented.

2 Principles of Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a bio-inspired concept that is at the core answer for our performance optimization technique. We introduce central PSO ideas as well as explicit tools and methods that we consolidated for our context.

2.1 Particle Swarm Optimization Basics

Particle swarm optimization is an optimization strategy that can manage issues whose solution is a point or surface in a n-dimensional space. In such a space, a basic speed is assigned to particles in the swarm, just as the channels of communication between particles. Firstly, particles in our research are edge nodes that provide computing resources. Secondly, velocity is linked to processing load and performance. These nodes then move through the response space and the results are calculated on the basis of a merit criterion after each time interval. Over time, nodes accelerate toward nodes of higher competence that are in the same communication group. Although each method works well in a range of problems, PSO has shown great success in solving continuous optimization problems. This algorithm is one of the optimization algorithms based on the random generation of the initial population. In this algorithm, it is constructed by modeling and simulating the group flight behavior of birds or group movement of fish. Each member in this group is defined by the velocity vector and the position vector in the search space. At each time iteration, the new position of the node is defined by the velocity vector and the position vector in the search space. At each iteration, the new node position is updated according to the current velocity vector, the best position found by that node, and the best position found by the best node in the group. This algorithm was originally defined for continuous parameters, but since in some applications we deal with discrete parameters, it is also extended to discrete parameters. The node swarm optimization algorithm is introduced with BPSO. In this algorithm, the position of each node is defined by a value of 1. In this algorithm, the position of each node is represented by a binary value of zero or one. In BPSO the value of each node can be changed from zero to one or from one to zero. The velocity of each node is also defined as the probability of each node changing to one value [33].

2.2 Basic Algorithm Definitions

We assume here a d-dimensional search space. The first node in this d-dimensional space for the position vector \( X_i \) is described in Eq. (1):
\[ X_i = (x_{i1}, x_{i2}, x_{i3}, \ldots, x_{id}) \]  

(1)

The velocity vector \( i \) of the first node is also defined in Eq. (2) with the vector \( V_i \):

\[ V_i = (v_{i1}, v_{i2}, v_{i3}, \ldots, v_{id}) \]  

(2)

In Eq. (3), we define the best position that the \( i \)-node has with \( P_{i, best} \):

\[ P_{i, best} = (p_{i1}, p_{i2}, p_{i3}, \ldots, p_{id}) \]  

(3)

And we define the best position of the best node among all the nodes with \( P_{g, best} \) as Eq. (4):

\[ P_{g, best} = (p_{g1}, p_{g2}, p_{g3}, \ldots, p_{gd}) \]  

(4)

In order to update the location of each node when moving through the response space, we define the following equations:

\[
V_i(t) = w \times v_i(t-1) + c_1 \times rand_1 \times (P_{i, best} - X_i(t-1)) \\
+ c_2 \times rand_2 \times (P_{g, best} - X_i(t-1)) 
\]

(5)

and

\[ X_i = x_i(t-1) + V_i(t) \]  

(6)

where \( w \) is the inertial weight coefficient (moving in its own direction) indicating the effect of the previous iteration velocity vector (\( V_i(t) \)) on the velocity vector in the current iteration (\( V_i(T+1) \)). \( c_1 \) is the constant training coefficient (moving along the path of the best value of the node examined). \( c_2 \) is the constant training coefficient (moving along the path of the best node found among the whole population). \( rand_1 \), \( rand_2 \) are random numbers with uniform distribution in the range of 1 to 2. \( V_i(t-1) \) is the velocity vector in iteration (\( t-1 \)). \( X_i(t-1) \) is the position vector in iteration (\( t-1 \)). To limit the velocity of a node moving from one location to another or diverting the velocity vector, we limit the velocity variations to the range \( V_{min} \) to \( V_{max} \), that is. The upper and lower speed limits are determined by the type of problem.

Some domain issues have specific definitions for their parameters and only have a limited, logical, and defined value in this domain. In other words, if there are any constraints or constraints in the problem under consideration, they must be accounted for by a mechanism to prevent nodes from entering the unauthorized space. This mechanism is called space constraint. If these mechanisms are not used, the response found by the algorithm is incorrect or unreliable. For example, Eq. (7) for negative x values in most programming languages is an error.

\[ f_x = \sum_{d=1}^{d} \sqrt{x} \]  

(7)

The mechanism used to account for this constraint is as follows:

\[ X = max(0, x) \]  

(8)

Sometimes, there are differences in how the algorithm is executed, which means that the steps are referred to in more separate ways, sometimes combining two or more steps.
into one step. But this does not make any sense in the programming that is done because what is important is to execute the program steps in the following order, and how these steps can be separated. In some references, for example, they combine steps four and five, meaning the step of updating the node velocity and moving the nodes to new locations as one step. This change will not cause any problems in the implementation of the algorithm.

### 2.3 Population Generation

The starting point is the generation of an initial population. The arbitrary generation of the underlying population is just the random determination of the underlying location of the nodes by a uniform dispersion in the solution space (search space). The arbitrary population generation phase of the underlying population exists in practically all probabilistic optimization algorithms. In any case, in this algorithm, notwithstanding the underlying random location the nodes, a specific measure of initial node velocity is additionally allocated. The underlying proposed range for node velocity results from Eq. (9).

\[
\frac{X_{\text{min}} - X_{\text{max}}}{2} \leq V \leq \frac{X_{\text{max}} - X_{\text{min}}}{2}
\]  

### 2.4 Selection of Primary Nodes

Expanding the quantity of essential nodes reduces the quantity of iterations required for the algorithm to converge. In any case, this reduction in the quantity of iterations doesn’t mean reducing the runtime of the program to accomplish convergence. An increase in the number of primary nodes does result in a decrease in the number of repeats. The increase in the number of nodes causes the algorithm to spend more time in the node evaluation phase, which increases the time it takes to run the algorithm until it achieves convergence, despite decreasing the number of iterations. So, increasing the number of nodes cannot be used to reduce the execution time of the algorithm. Another misconception is that the number of nodes can be reduced to optimize the execution time of the algorithm. It should be noted that decreasing the number of nodes may cause local minima to fall and the algorithm fails to reach the original minimum. If we consider the convergence condition as the number of iterations, although decreasing the number of initial nodes decreases the execution time of the algorithm, the solution obtained would not be the optimal solution to the problem. Thus, the initial population size is determined by the problem. In general, the number of primary nodes is a compromise between the parameters involved in the problem. Experimentally selecting the initial population of nodes of 2 to 5 nodes is a good choice for almost all test problems.

### 2.5 Evaluation of the Objective Function

The objective functions is also called the cost or fitness calculation of nodes. We have to assess every node which represents an answer for the issue under investigation. Contingent upon this, the assessment strategy differs. For instance, in the event that it is
conceivable to define a mathematical function for the purpose, essentially by setting the input parameters (extracted from the node position vector) into this mathematical function, it is anything but difficult to calculate the cost of the node. Note that every node contains total data about the input parameters of the issue that this data is extracted and targeted to.

Sometimes it is not realistic to expect to characterize a scientific function for node evaluation. This happens when we have linked the algorithm to another software or used the algorithm for experimental data. In such cases, information about software input or test parameters should be extracted from the node position vector and placed in the software associated with the algorithm or applied to the relevant test. Running software or performing tests and observing or measuring the results determines the cost of each node at the end.

2.6 Recording the Best Position

The best position needs to be determined for each node (Pi.best) and the best position among all nodes (Pg.best). There are two cases to consider: On the off chance that we are in the first iteration (t = 1), we think about the current position of every node as the best location for that node - see Eqs. (10) and (11).

\[ P_{i, \text{best}} = X_i(t), \ i = 1, 2, 4, ..., d \] (10)

and

\[ \text{cost}(P_{i, \text{best}}) = \text{cost}(X_j(t)) \] (11)

Furthermore, in the other iterations, we compare the amount of cost for the nodes in Step 2 with the value of the best cost for each node. If this cost is less than the best recorded cost for this node, then the location and cost of this node replaces the previous one. Otherwise there is no change in the location and cost recorded for this node:

\[
\begin{cases} 
\text{if} \ \text{cost}(X_i(t)) < \text{cost}(P_{i, \text{best}}) \\
\text{else} \ \text{no change}
\end{cases} \Rightarrow \\
\begin{cases} 
\text{cost}(P_{i, \text{best}}) = \text{cost}(X_j(t)) \\
P_{i, \text{best}} = x_i(t)
\end{cases}
\] (12)

3 Performance Optimization for Clustered Architectures

The PSO technique can now be applied as to optimize processing times in our multi-cluster edge scenario. We present another approach to limit total delay and latency in edge-based clusters.

Our optimization method for clustered edge architectures has the following four main steps:

1. the edge cluster architecture is defined.
In the following, each of these steps will be explained in detail.

3.1 Edge Request Management

Three layers can be identified in our architecture: the things layer, where the objects and end users are found, the edge layer, where the edge nodes for processing are found, lastly the cloud layer, where the cloud servers are located [11]. We expect the edge nodes to be clustered. That is, there are various clusters of edge nodes, where each has a local coordinator.

A cloud server can consist of a number of processing units, such as a rack of physical servers or a server with multiple cores. In each layer, the nodes are divided into domains where the IoT-edge-cloud application is implemented. A range of IoT nodes can be objects from a smart home, a temperature sensor in a factory, or a soil moisture sensor in a farm, all objects around which they are considered to be part of a domain.

Both edge and cloud computing provide users with storage, applications and data. But edge is closer to the end user and has a wider geographical distribution. The edge networking consists of a data page and a control page. For example, on a data page, cloud computing allows services to be at the edge of the network rather than in the data center [18]. Devices in the edge are known as nodes. Any device with network, computing, and storage connections can be up to one node. The reason is simple: when data collection is close, data analysis is also less delayed [24].

**Node-Based Request Processing.** The correspondence between IoT, edge and cloud nodes occurs as follows. IoT nodes can process demands locally or send them to the controller for processing in edge or cloud. Edge nodes can process approaching demands or, if inaccessible, pass them to another edge node or the controller. Cloud nodes process the dispensed demands and return the outcome to the IoT nodes.

The purpose for this investigation is to minimize total execution time for IoT nodes in a proposed framework based on edge computing. The edge layer is located between the things layer and the cloud so that it can cover the majority of requests from the IoT nodes to minimize service delays.

We first introduce latency minimization rules and then formulate the IoT delay so we can analyze the rules analytically [36].

**Request Offloading in the Edge Layer.** The concept and rules of computational discharge in moving vehicles have been studied in different researches and on different criteria such as energy consumption, response time, availability, and so on. In this proposed method, the load discharge decision is a request based on the response time of the nodes of the edge, which depends on different factors. These include the amount of processing required to execute a request, the status of the node queue, and the computational power of a node. A controller decides to which edge (or cloud) node to allocate a
request. The transfer time and the waiting (e.g., queuing time) at the controller typically causes a delay in processing.

**Definition 1.** A *delay* is the time spent by transferring a request to the next node and waiting there to be processed. Thus, we typically have controller delays $D_C$, edge node delays $D_E$ and IoT node delays $D_I$. A **processing time** is the time for execution at a node, i.e., either a controller processing time $P_C$ or an edge node processing time $P_E$.

**Definition 2.** The **response time** $R$ for an IoT node is the time it takes to get a request processed, i.e., the time between sending a request for processing until receiving the result.

$$RT = D_C + P_C + D_E + P_E + D_I$$

This is also known as **end-to-end latency** in networked environments.

The requests are produced by the IoT nodes with a specific deadline for handling and they are sent to the controller for allocation of processing nodes. The architectural framework of the edge-based system used in this study is shown in Fig. 1. The requests transferred from different IoT nodes get into a queue until they finally get to the controller. The controller will consider the total waiting time of all edge nodes from their availability tables with consideration of the request deadline and will then allocate the request to the best edge node with the lowest total waiting time.

![Diagram of IoT-edge-cloud architecture](image)

**Fig. 1.** The IoT-edge-cloud architecture [3].
3.2 **Orchestration Controller Design**

As a central component, the controller is inserted as an orchestrator between the things layer and the edge. All requests from the things layer will initially be moved to the controller and afterward sent to either the best edge node or directly to the cloud. As stated, the controller plays out the decision procedure based on the total waiting time of the entire request in various edge nodes. Upon receiving the new request, the controller determines the best node and allocates the request to that node according to the deadline of the request and the lowest total waiting time of all the edge nodes using the particle optimization algorithm PSO. The status of the selected node’s queue and its execution status are updated in the availability table. If no appropriate node is found in the edge layer for a received request, the controller sends the request directly to the cloud for backup processing.

**Types of Cluster Interactions.** We can identify two types of interaction for edge nodes that can be implemented [32]:

- coordinated, in which some dedicated nodes control the interactions of their surrounding nodes,
- distributed, in which each edge node interacts with the other node.

In the coordinated strategy, the edge layer is divided into smaller clusters, with a central coordinating node in every one of these clusters, which is directly cooperating with the controller and which controls other nodes in its cluster. This coordinator is aware of the queue status of those nodes and stores all the information in the availability table. The central coordinating nodes are also processing nodes which aside from their processing responsibility, can manage the other nodes in their clusters too. These central coordinators have 3 different connections, 2 direct connections and 1 public connection. The central coordinators are directly connected to their cluster’s nodes and the controller and they communicate with other central coordinators with public announces. When a request is sent to the controller, the cluster coordinators announce their best node in their area (personal best) as a publicly and support the lead controller in determining the best node in the layer (global best). This step was formalised in Eq. (6) presented earlier on.

3.3 **Performance Optimization**

PSO-based edge performance optimization is the concern of the third step. In our technique, the optimization issue has one main objective and one sub-objective so that the fulfillment of the sub-objective will lead to the satisfying of the main objective. In the following, each one of these objectives will be characterized.

- Primary objective: The main purpose is to minimize total response time $R$. In the proposed method, two elements, controller and the particle optimization algorithm have been used to accomplish this goal.
- Secondary objective: The secondary objective is to reduce delay $D$. Delay in each layer must be considered separately to calculate the total delay.
For our delay calculation, we adopt the solution presented in [37] in our cluster management setting.

**Calculating Delay in the Things Layer.** Note that thing nodes can both process requests themselves or send them to the edge or cloud for processing. On the off chance that an IoT node chooses to send their request to the edge or cloud for process, the request will be sent to the controller first. Considering the number of the IoT nodes and the number of the requests, the sent request will get into a queue before reaching the controller and after reaching the controller, the request should wait until the controller finds the best edge node for allocation. In other words, This is the delay before the allocation takes place.

**Definition 3.** The delay in the IoT node $i$ is represented by $D_i$ and is calculated as follows:

$$
D_i := P_I^i \times (A_i) + P_F^i \times (X_{ij}^{IF} + Y_{ij}^{IF} + L_{ij}) + P_C^i \\
\times (X_{ik}^{IC} + Y_{ik}^{IC} + \bar{H}_k + X_{ki}^{CI} + Y_{ki}^{CI})
$$

(13)

where

- $P_I^i$ is the probability that the things node will process the request itself in the things layer, $P_F^i$ is the probability of sending the request to the edge layer, and $P_C^i$ is the probability of sending the request directly to the cloud; with $P_I^i + P_F^i + P_C^i = 1$.

- $A_i$ is the average processing delay of node $i$ when processing its request. $X_{ij}^{IF}$ is the propagation delay from object node $i$ to node $j$. $Y_{ij}^{IF}$ is the sum of all delays in linking from object node $i$ to node $j$. Likewise, $X_{ik}^{IC}$ delays propagation from object node $i$ to cloud $k$ server and $X_{ki}^{CI}$ is the sum of all delays in sending a link from object node $i$ to cloud server $k$. $X_{ik}^{C1}$ and $Y_{ki}^{C1}$ are broadcast and send delays from the $k$ server to the node $i$. Delayed transmission from the cloud layer to the object layer will be considered in $L_{ij}$, as the request edge later be unloaded to another node in the edge layer. $L_{ij}$ is the processing delay of the node $i$ request in the edge layer or even the cloud layer, if it is unloaded from the edge node to the cloud server, so that the node $j$ edge be the first node in the edge layer to which the node request of object $i$ is sent. Note that the edge node $j$ edge load the request for any reason to its best neighbor node or discharge cloud, and all similar delays occurring in $L_{ij}$ are considered. $\bar{H}$ is the average delay for processing a request on the cloud server $k$, which includes the queue waiting time on the cloud server $k$ plus the request processing time on the cloud server $k$.

There is no specific distribution for $P_I^i$, $P_F^i$, and $P_C^i$, because their values will be defined by separate applications based on service quality requirements and rules. In other words, their values will be given as input to this framework.

**Calculating Delay in Edge Layer.** We now define a recursive function for the calculation of $L_{ij}$. 
**Definition 4.** $L_{ij}$ is a delay for processing IoT node $i$’s requests in the edge layer. After allocation, the request will get into the chosen edge node’s queue. This is the delay after allocation. Thus, $L_{ij}$ is calculated from the equation below:

$$L_{ij} = P_j(W_j + X_{ji}^F + Y_{ji}^F) + (1 + p_j \cdot (1 - \phi(x)) [X_{jj'}^F + Y_{jj'}^F + L_{ij'}(x + 1)]$$

$$+ \phi(x) [X_{jk}^C + Y_{jk}^C + (\overline{T_k} + X_{ki}^C + Y_{ki}^C)]$$

$$j' = \text{best}(j), k = h(j)$$

(14)

$W_j$ refers here to the mean waiting time at node $j$ and $\phi(x)$ is also a discharge function.

### 3.4 PSO-Based Performance Optimization

In the fourth and final step, the actual performance optimization is carried out. The PSO algorithm is utilized to take care of the optimization issue. This algorithms is comprised of a few stages, which will be discussed now.

**Establish an Initial Population and Evaluate It.** The particle swarm optimization algorithm starts with an initial random population matrix like many evolutionary algorithms, such as genetic algorithms. This algorithm, unlike genetic algorithms however, has no evolutionary operator such as a mutant. Each element of the population is called a node. In fact, the particle swarm optimization algorithm consists of a finite number of nodes that randomly take the initial value.

Here, the edge layer is divided into different clusters and in each cluster consists of a central coordinator node and its dependent nodes. For each node, two states of location and velocity are defined, which are modeled with a location vector and a velocity vector, respectively.

**The Fitness Function.** The fitness function is used for evaluating the initial population. Since the problem is a two-objective optimization, both goals must be considered in the fitness function.

- The first objective is to minimize the total response time in the edge-based architecture indicated by $RT$.
  To achieve the first goal, we define a metric called $T_E = P_E + D_E$ that represents the total execution time of the request at the edge node, which is the sum of the processing time $P_E$ of the request and the waiting time $D_E$ of the request in the edge node’s queue. $TimeFinal (TF)$ is the maximum time $T_E$ that is allowed for the execution at the edge node in other to meet the required deadline $DL$ with

$$TF = DL - (D_C + P_C + D_I)$$

(15)

i.e., $Max(T_E) = TF$ or $T_E \in [0 \ldots TF]$. 

The second objective is to reduce the delay of the edge-based architecture $D$.

The second goal relates to the sum of the delays in the IoT layer and the delay in the edge layer:

$$D = D_C + D_E + D_I$$

(16)

As is clear, both goals are defined as minimization.

Ultimately, fitness is calculated as follows:

$$Fitness = TF + D$$

(17)

**Determine the Best Personal Experience and the Best Collective Experience.** The nodes move in the solution space at a dynamic rate based on the experience of the node itself and the experience of the neighbors. Unlike other evolutionary algorithms, particle swarm optimization does not use smoothing operators such as intersections in the frog algorithm. Thus, the answers remain in the search space to share their information and guide the search to the best position in the search space. So, here the coordinating nodes search for the best experience within their own and their neighbor’s domain. To update the node’s velocity and position, first the best position of each node and then the best position among all nodes in each step must be updated.

**Location and Velocity Updates.** The dimension of the problem space is equal to the number of parameters in the function to optimize. A memory is allocated to store the best position of each node in the past, and a memory is allocated to store the best position of all nodes. With the experience of these memories, the nodes decide how to move next. At each iteration, all the nodes move in the next n-dimensional space of the problem to find the general optimum point. The nodes update their velocities and their position according to the best absolute and local solutions. Here, the coordinating nodes read the availability table of their cluster nodes and publish their best nodes to the controller. In this way, they move towards the best node.

**Check the Stop Condition.** Finally, the stop condition is checked. If this condition is not met, we return to the stage of determining the best personal experience and the best collective experience.

There are several types of stopping conditions:

- Achieve an acceptable level of response,
- Reach a specified number of repetitions/time,
- Reach a certain number of iterations or time specified without seeing an improvement,
- Check a certain number of responses.

Here, the stop condition is to achieve an acceptable level of response.

**3.5 Algorithm Definition (Pseudocode)**

The proposed PSO Performance Optimization algorithm is presented in Algorithm 1.
4 Framework Implementation and Evaluation

The concept of IoT is to connect different devices through the Internet. With the help of IoT, different applications and devices can interact and talk to each other, even humans, via the Internet. For example, smart refrigerators that connect to the Internet tell you the inventory and expiry date of foods in the refrigerator. In fact, IoT enables you to remotely manage and control your used objects with the help of Internet infrastructure. IoT provides opportunities to integrate directly into the physical world and computer-based systems, such as smart cars, smart glaciers and smart homes, which are referred to these days in various discussions and conventions. And it is necessary to know that all of these devices fall under the category of Internet of Things. Edge is another layer of distributed environmental networks and is closely linked to cloud computing and IoT. Edge computing is the idea of a distributed network that connects the two environments. In the previous section, using a layer of edge, a method was proposed to reduce the amount of execution time and delay in executing IoT requests. In this section we will present the results of implementation. The main proposed method is to use the node swarm optimization algorithm in the controller but also to compare the bat algorithm in the controller.

4.1 Test Environment and Objectives

The overall objective here is to reduce the total response time, i.e., the end-to-end latency. We chose a comparative experimental approach to evaluate our framework.

The PSO particle swarm optimization algorithm and another bio-inspired, so-called BAT algorithm are utilized to build up the controller and compare. The BAT algorithm was picked because of its principle similarity to the particle swarm optimization.
Thus, it allows meaningful performance comparisons. Furthermore, its wide-spread use in different optimization situations make it a suitable benchmark. The BAT algorithm is an algorithm inspired by the collective behavior of bats in the natural environment proposed in [35]. This algorithm is based on the use of bats’ reflection properties.

We used MATLAB software to evaluate the solution. The concepts presented earlier are fully coded and implemented in this software.

### 4.2 Definition of PSO and BAT Parameters

For better understanding, this algorithm is actualized with the double target evaluation function as indicated by Eq. (11) with 200 iterations. The underlying parameters indicated in this implementation are shown in Table 1.

| Parameters            | Amounts |
|-----------------------|---------|
| Population number     | 50      |
| Number of repeats     | 200     |
| Value w               | 1       |
| Decrease coefficient w| 0.99    |
| c1, c2, c3            | 2       |

### 4.3 Comparison of Fitness Values

Using the values in Table 1 as initialization, the particle swarm optimization algorithm is formed and the graph in Fig. 2 shows the result of the implementation of this algorithm.

As can be seen in Fig. 2, with the expansion of iterations, the results of the fitness evaluation function made for the two targets (runtime and delay) is decreased. Our PSO performance optimization algorithm is a two-target algorithm that decreases execution time and delay in execution of requests. The objective function value in the implementation of the particle swarm optimization algorithm is approximately 233. It should be noted that due to the random structure of the evolutionary algorithms, the results per run may be different from the previous ones.

So as to compare the proposed strategy, an evolutionary BAT algorithm has been actualized so as to contrast accurately and similar conditions. Thus, the BAT algorithm has been implemented with the same two-objective evaluation function and with 200 iterations as the particle optimization algorithm. The initial parameters specified in this implementation are shown in Table 2.

| Parameters            | Amounts |
|-----------------------|---------|
| Population number     | 50      |
| Number of repeats     | 200     |
| Value w               | 1       |
| Decrease coefficient w| 0.99    |
| c1, c2, c3            | 2       |

Utilizing the above qualities as the initialization, the BAT algorithm is formed and the diagram in Fig. 3 shows the aftereffect of the implementation of this algorithm.

In Fig. 3, the motion diagram of the BAT algorithm is shown. The conditions are the same for both particle swarm and bat optimization algorithms and the objective function in both algorithms has been implemented and evaluated with respect to both
runtime and delay reduction. As can be seen, the BAT algorithm has reached the target function of 237.5, while in the particle swarm optimization algorithm this value is 233. These results indicate that the proposed algorithm is better than the evolutionary BAT algorithm.

### 4.4 Scenario-Based Comparison: Overview

In order to deepen the analysis, the proposed solution was tested across 3 different scenarios:

- once with different number of requests,
- once with different number of edge layer nodes,
- once with identical parameters, but in different iterations,

These will now be discussed in sequence.

---

**Fig. 2.** Fitness results for the PSO algorithm [3].

**Table 2.** The initial parameters in bat [3].

| Parameters               | Amounts |
|--------------------------|---------|
| Population number        | 50      |
| Number of repeats        | 200     |
| Minimum frequency        | 0       |
| Maximum frequency        | 100     |
| Sound intensity coefficient | 0.1    |
| Pulse rate coefficient   | 0.1     |
4.5 Scenario 1: Request Variation

In the first scenario, various requests and nodes in the edge layer are utilized to compare the outcomes. In this scenario, we fixed the quantity of edge layers nodes and assumed variable and incremental user requests. Table 3 shows the details of this scenario configuration.

Figure 4 shows the results of several different user requests with 20 nodes in the edge layer. In this scenario, the number of edge layer nodes is 20 and the number of user requests are 30, 50, 100, 150, 200 and 250. Equation (11) is used to calculate the fitness function.

As can be found in Fig. 4, our PSO-based optimization algorithm produces much better outcomes than the BAT algorithm. As the quantity of requests builds, the estimation of the target function (execution time + delay time) increments. Expanding the quantity of requests will build the execution time, just as increment the execution time,

---

**Fig. 3.** Fitness results for the BAT algorithm [3].

**Table 3.** The initial parameters in first scenario [3].

| Parameters                          | Amounts                               |
|-------------------------------------|---------------------------------------|
| Number of user requests             | 30/60/100/150/200/250                 |
| Number of edge layer nodes          | 20                                    |
| Processing power of each edge node  | 4 G Ram/8 Mips Processor              |
| Amount of time each user requests   | Randomized in range [1, 20]           |
| Amount of CPU per user request      | Random in the interval [2, 8]         |
as more nodes will be included. This expands the estimation of the target function, which is the sum of the execution time and delay. In all cases, our particle swarm optimization algorithm shows better results. Despite the increase in the objective function value in both algorithms, the growth rate of the objective function value in the particle swarm optimization algorithm is lower than the BAT algorithm, which means that this algorithm outperforms the BAT algorithm.

In order to make comparisons under different conditions, the next step is to increase the number of layer nodes in order to observe the effect of this increase in a graph.

### 4.6 Scenario 2: Edge Node Variation

In the second scenario, in contrast to the previous scenario, now the quantity of requests is fixed, yet the quantity of edge layer nodes is thought to be variable. Expanding the quantity of nodes has been done as an experimentation and no exceptional algorithm is utilized. Full details of the second scenario are given in Table 4.

**Table 4.** The initial parameters in the second scenario [3].

| Parameters                        | Amounts                        |
|----------------------------------|-------------------------------|
| Number of user requests          | 100                           |
| Number of edge layer nodes       | 5/10/15/20/30/50              |
| Processing power of each edge node | 4 G Ram/8 Mips Processor      |
| Amount of time each user requests | Randomized in range [1, 20]   |
| Amount of CPU per user request   | Random in the interval [2, 8] |
For this experiment, 100 input requests are considered. In this scenario, the number of user requests is considered to be fixed, but the number of edge layer nodes is considered to be 5, 10, 15, 20, 30 and 50. Equation (11) is used to calculate the fitness function.

![Fig. 5. Result of second scenario: fitness over number of requests [3].](image)

By expanding the quantity of layer nodes in the edge, the fitness function diminishes because of the chance of executing requests on more nodes. The higher the quantity of edge layer nodes, the more probable it is that requests will be processed utilizing nodes whose latency is lower. In other words, with the increase in the number of edge layer nodes the controller’s options for allocating more requests are increased and thus the chances of finding a suitable node with low latency increases. As the conditions change, the way in which requests are executed is also varied, which reduces execution time. For the particle optimization algorithm, the greater the number of edge layer nodes, the lower the objective function. Furthermore for PSO algorithm, the reduction of the target function is much faster than the BAT algorithm, which is evident in Fig. 5. Each algorithm was run multiple times to better compare the algorithms.

4.7 Scenario 3: Iteration Variation

In the third scenario, a number of iterations assume both the used requests and the quantity of nodes in the edge layer fixed. In this scenario, every algorithm was run multiple times with similar inputs and the outcomes were acquired. It ought to be noticed that in this investigation, the quantity of requests is 100 and the quantity of edge layer nodes is 50, which were steady at all 5 times. Equation (11) is used to calculate the fitness function. Table 5 shows the full configuration details of the third scenario.
### Table 5. The initial parameters of the third scenario [3].

| Parameters                               | Amounts                              |
|------------------------------------------|--------------------------------------|
| Number of user requests                  | 100                                  |
| Number of edge layer nodes               | 50                                   |
| Processing power of each edge node       | 4 G Ram/8 Mips Processor             |
| Amount of time each user requests        | Randomized in range [1, 20]          |
| Amount of CPU per user request           | Random in the interval [2, 8]        |

According to Fig. 6, in different iterations we can see different results despite not changing the input values at each iteration. There are no identifiable rules or explanations detectable by analysing the outputs for the two different algorithms. For example, the value of the fitness function in the first step with one iteration is less than that of the BAT algorithm in the particle swarm optimization algorithm, and the value of this function in the second step with two iterations in both algorithms decreased while in the next step with three iterations, the value of this function is increased in both algorithms.

As a rule, it tends to be reasoned that the particle swarm optimization algorithm in executing requests utilizing the double fitness function (execution time + execution delay) yields essentially better outcomes over the evolutionary BAT algorithm. This is on the grounds that in many iterations, the target function value in the particle swarm optimization algorithm was lower than the BAT algorithm.

A feature of the particle swarm optimization algorithm is faster convergence. In addition, the particle swarm optimization algorithm yields overall good performance results that reduce orchestration and response time.

![Fig. 6. Result of second scenario: fitness over number of requests [3].](image)
5 Related Work

In this section, we review approaches to cloud and edge cluster management in general, but also look at how bio-inspired approaches such as PSO have been used. Regarding the first concern, [12] have studied the connection for the circulation of work and virtual machine task in cyber-physical frameworks based on edge computing standards. They looked at minimizing the final cost and satisfying service quality requirements. The quality needs of services provided by cyber-physical systems are challenged by unstable and delayed communication between data sources and devices. Processing resources at the edge of the network is introduced as a solution. The paper did not use realistic IoT and cloud scenarios. In [21], energy-delay computing is tended to in a task allocation setting. Given the significance of cost and energy in delay-delicate interactions for mentioning and giving processing resources, an ideal strategy for delay-sensitive associations was introduced. The plan looks to accomplish a energy-delay bargain at the edge of the cloud. The authors formalized the task allocation issue in a cloud-edge setting, and yet utilized basic models to plan energy loss and delay as the key indicators.

Another performance-centric view is [31], which also centers around demonstrating delay, energy loss and cost. In this examination, comparably to the past articles, delay and energy loss were detailed with fundamental models, and no laws were acquainted to minimize delay. Our aim is to use an evolutionary algorithm for edge orchestration and to obtain the optimal response times for the problem.

In a different direction, [4] proposes a human services framework. A server converts the perceptions and estimations to a far off server for future recovery of clinical experts over the Internet. In the proposed strategy, data and requests are sent over the Internet from the earliest starting point, in this way evading any potential delays because of the physical issues of the devices without adequate performance optimization. In [29], the solution is based on a wireless sensor network. The purpose of the proposed method is ultimately to identify the delays-sensitive requests and take action directly when the problem occurs.

We can also find PSO-specific solutions relevant to our context. In [10], a framework of the particle swarm optimization algorithm is proposed. Based on the proposed framework, a multi-objective discrete particle swarm optimization algorithm is proposed to solve the network clustering problem. The decomposition mechanism was adopted. A problem-specific population initialization method based on label propagation and a turbulence operator was introduced. In the proposed method, two evaluation objectives termed as kernel k-means and ratio cut were to be minimized. The clustering performances of the proposed algorithm have been validated on signed networks and unsigned networks.

In [1], a greedy discrete particle swarm optimization framework for large-scale social network clustering was suggested. To determine the performance of the algorithm, experiments on both synthetic and real-world social networks were carried out. The authors also compared the proposed algorithm with several state-of-the-art clustering methods for network settings. In [8], the authors developed an automated orchestration technique for clustered cloud architectures. An Autonomous Particle Swarm Optimization, called the A-PSO algorithm, was implemented that enabled an edge node, such as a remote storage, to work as part of a decentralized, self-adaptive intelligent
task scheduling and load balancing agent between resources in distributed edge settings as one concrete example.

Often multi-objective optimisation is a concern. In [2], the authors model the task of complex network clustering as a multi-objective optimization problem and solve the problem with the quantum mechanism based particle swarm optimization algorithm, which is a parallel algorithm. Consequently, a quantum-behaved discrete multi-objective particle swarm optimization algorithm is proposed for complex network clustering. This algorithm has the ability to determine the number of clusters automatically, rather than setting the number of clusters in advance, which is very important for large scale network analysis. Here, since the network clustering is a discrete problem, the discrete PSO algorithm is adopted instead of the continuous one.

6 Conclusions and Future Work

In order to make the Internet-of-Things work, edge computing guarantees low latency because of more local processing. In any case, a more intensive look uncovers distributed and independently managed cluster of processing edge nodes that need be considered in a performance-oriented load allocation procedure. Moreover, delays do happen as the aftereffect of transmission processing and waiting times of requests at nodes that perform orchestration and processing tasks in the edge clusters.

We presented here a performance optimization framework, for which we utilized an orchestration algorithm based on particle swarm optimization, adapted to the multi-cluster requirements and concentrating on delay and end-to-end latency reduction, We compared our solution with a evolutionary BAT algorithm, another strategy to optimize and diminish the mean target function (delay and execution latency) of processing demands. Developmental algorithms are among the best optimization algorithms, and the particle swarm optimization algorithm we embraced here is less mind boggling than some other evolutionary algorithms. These advantages made our PSO-based technique an ideal orchestration strategy aiming to reduce execution time and delay as the key performance criteria.

Despite its general suitability here, particle swarm optimization algorithm has several characteristics and possible limitations that shall briefly be discussed here. In this algorithm, it is possible to place the node in local optimality. While the particle swarm optimization algorithm is faster than other evolutionary algorithms, it usually cannot offset the quality of the solution by increasing iterations. One of the reasons is that in this algorithm the nodes converge to a particular point, which is between the best general position and the best personal position. Due to this disadvantage, many changes have been made to the particle swarm optimization algorithm. Another disadvantage of this approach is its dependence on the problem. This dependence is usually the result of changes in the parameters of the algorithm. In general, one parameter cannot be used for all problems. The particle swarm optimization algorithm has several advantages over standard optimization methods:

- Particle swarm optimization algorithm is a population-based algorithm. This property makes it less likely to get caught in the local minimum.
This algorithm operates on contingency rules, not definitive rules. Therefore, the node swarm optimization algorithm is a stochastic optimization algorithm that can search for uncertain and complex areas. This property makes the particle swarm optimization algorithm more flexible and robust than conventional methods.

Particle swarm optimization algorithm deals with non-differential objective functions because the particle swarm optimization algorithm uses information output (efficiency index or objective function) to guide the search in the problem space.

The quality of the proposed route response does not depend on the initial population. Starting at any point in the search space, the algorithm converges to the optimal solution.

The particle swarm algorithm has a great deal of flexibility to control the balance between local and general search space. This unique property of the particle swarm optimization algorithm overcomes the problem of convergence in time and increases the search capacity that all of these properties make the particle swarm optimization algorithm different from Genetic Algorithm (GA) and other heuristic algorithms.

The results of this method show better suitability over some of the search optimization methods.

We have show a core solution that can be expanded in different ways. As a part of our future work, we intend to consider other algorithm bases, for example, the firefly algorithm rather than the particle swarm optimization algorithm for edge performance optimization. The firefly algorithm is a helpful and extremely common algorithm in optimization issues, yet it doesn’t have the impediments of the genetic algorithm in how to choose the necessary parameters, which is the best decision for these two tasks. We could likewise consider the ant colony algorithm rather than PSO. This algorithm is extremely effective and is exceptionally utilized in routing issues. Aside from the algorithmic side, we additionally plan to refine the model by more exactly isolating starting points of delay in communication and buffering times. Likewise different coordination standards from completely centralized to peer-to-peer management can be thought of. Also, as a final plan, we mean to combine this with an auto-scaling controller [9], which we implemented so far just for a single cluster condition.

We provided a generic framework here. However, we aim to explore this in more concrete application settings. We consider here traffic management and coordinated cars as a concrete IoT and edge setting, where traffic and car movement is captured and processed, maybe combined by infotainment information with image and video data [6,16,20,25]. This mobile setting would need to be supported by local clusters that act autonomously, but require some higher-level coordination [7,14,27].

Another application domain is mobile learning. This also relies on the extensive use of multimedia content being delivered to mobile learners, similar to the infotainment case above, and their devices [13,23,26]. These types of applications also need interaction with application-specific processing of interactions in order to support the required learning processes [15,19]. In widely delivered course, particularly to mobile learners, these would need be provided at the edge to ensure sufficient platform performance for adequate user experience.
References

1. Cai, Q., Gong, M., Ma, L., Ruan, S., Yuan, F., Jiao, L.: Greedy discrete particle swarm optimization for large-scale social network clustering. Inf. Sci. 316, 503–516 (2015). https://doi.org/10.1016/j.ins.2014.09.041. Nature-Inspired Algorithms for Large Scale Global Optimization

2. Li, L., Jiao, L., Zhao, J., Shang, R., Gong, M.: Quantum-behaved discrete multi-objective particle swarm optimization for complex network clustering. Pattern Recogn. 63, 1–14 (2017). https://doi.org/10.1016/j.patcog.2016.09.013

3. Azimi, S., Pahl, C., Shirvani, M.H.: Particle swarm optimization for performance management in multi-cluster IoT edge architectures. In: International Conference on Cloud Computing and Services Science CLOSER. SciTePress (2020)

4. Babu, S., Chandini, M., Lavanya, P., Ganapathy, K., Vaidehi, V.: Cloud-enabled remote health monitoring system. In: 2013 International Conference on Recent Trends in Information Technology (ICRITT), pp. 702–707, July 2013. https://doi.org/10.1109/ICRITT.2013.6844286

5. Baktyan, A.A., Zahary, A.T.: A review on cloud and fog computing integration for IoT: platforms perspective. EAI Endorsed Trans. Internet Things 4(14) (2018). https://doi.org/10.4108/eai.20-12-2018.156084

6. Barzegar, H.R., Ioini, N.E., Le, V.T., Pahl, C.: Wireless network evolution towards service continuity in 5G enabled mobile edge computing. In: 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC), pp. 78–85 (2020)

7. El Ioini, N., Pahl, C., Helmer, S.: A decision framework for blockchain platforms for IoT and edge computing. SCITE Press (2018)

8. Faheem, H., Pahl, C.: Enhanced particle swarm optimisation and multi objective optimization for the orchestration of edge cloud clusters, September 2019

9. Gand, F., Fronza, I., Ioini, N.E., Barzegar, H.R., Azimi, S., Pahl, C.: A fuzzy controller for self-adaptive lightweight container orchestration. In: International Conference on Cloud Computing and Services Science CLOSER. SCITE Press (2020)

10. Gong, M., Cai, Q., Chen, X., Ma, L.: Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition. IEEE Trans. Evol. Comput. 18(1), 82–97 (2014)

11. González, L.M.V., Rodero-Merino, L.: Finding your way in the fog: towards a comprehensive definition of fog computing. Comput. Commun. Rev. 44(5), 27–32 (2014)

12. Gu, L., Zeng, D., Guo, S., Barnawi, A., Xiang, Y.: Cost efficient resource management in fog computing supported medical cyber-physical system. IEEE Trans. Emerg. Top. Comput. 5(1), 108–119 (2017). https://doi.org/10.1109/TETC.2015.2508382

13. Kenny, C., Pahl, C.: Automated Tutoring for a Database Skills Training Environment. Association for Computing Machinery, New York (2005)

14. Le, V., Pahl, C., El Ioini, N.: Blockchain based service continuity in mobile edge computing. In: 6th International Conference on Internet of Things: Systems, Management and Security (2019)

15. Lei, X., Pahl, C., Donnellan, D.: An evaluation technique for content interaction in web-based teaching and learning environments. In: Proceedings 3rd IEEE International Conference on Advanced Technologies, pp. 294–295 (2003)

16. von Leon, D., Miori, L., Sanin, J., Ioini, N.E., Helmer, S., Pahl, C.: A lightweight container middleware for edge cloud architectures, pp. 145–170 (2019)
17. Mahmoud, M.M.E., Rodrigues, J.J.P.C., Saleem, K., Al-Muhtadi, J., Kumar, N., Korotaev, V.: Towards energy-aware fog-enabled cloud of things for healthcare. Comput. Electr. Eng. 67, 58–69 (2018)
18. Mahmud, R., Srirama, S.N., Ramamohanarao, K., Buyya, R.: Quality of experience (QoE)-aware placement of applications in fog computing environments. J. Parallel Distrib. Comput. 132, 190–203 (2019)
19. Melia, M., Pahl, C.: Constraint-based validation of adaptive e-learning courseware. IEEE Trans. Learn. Technol. 2(1), 37–49 (2009)
20. Mendonça, N.C., Jamshidi, P., Garlan, D., Pahl, C.: Developing self-adaptive microservice systems: challenges and directions. IEEE Softw. 38, 70–79 (2019)
21. Meng, H., Zhu, Y., Deng, R.: Optimal computing resource management based on utility maximization in mobile crowdsourcing. Wirel. Commun. Mob. Comput. 2017, 1–13 (2017)
22. Minh, Q.T., Nguyen, D.T., Le, V.A., Nguyen, D.H., Pham, T.V.: Task placement on fog computing made efficient for IoT application provision. Wirel. Commun. Mob. Comput. 2019, 6215454:1–6215454:17 (2019)
23. Murray, S., Ryan, J., Pahl, C.: A tool-mediated cognitive apprenticeship approach for a computer engineering course. In: Proceedings 3rd IEEE International Conference on Advanced Technologies, pp. 2–6 (2003)
24. Mutlag, A.A., Ghani, M.K., Arunkumar, N.A., Mohammed, M.A., Mohd, O.: Enabling technologies for fog computing in healthcare IoT systems. Future Gener. Comput. Syst. 90, 62–78 (2019)
25. Pahl, C., Ioini, N.E., Helmer, S., Lee, B.: An architecture pattern for trusted orchestration in IoT edge clouds. In: 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC), pp. 63–70 (2018)
26. Pahl, C., Barrett, R., Kenny, C.: Supporting active database learning and training through interactive multimedia. ACM SIGCSE Bull. 36(3), 27–31 (2004)
27. Pahl, C., Fronza, I., El Ioini, N., Barzegar, H.R.: A review of architectural principles and patterns for distributed mobile information systems. In: WEBIST, pp. 9–20 (2019)
28. Pahl, C., Jamshidi, P., Zimmermann, O.: Architectural principles for cloud software. ACM Trans. Internet Technol. (TOIT) 18(2), 17 (2018)
29. Rolim, C.O., Koch, F.L., Westphall, C.B., Werner, J., Fracalossi, A., Salvador, G.S.: A cloud computing solution for patient’s data collection in health care institutions. In: Finkelstein, J., Ossebaard, H.C., van Gemert-Pijnen, L. (eds.) eTELEMED, pp. 95–99. IEEE Computer Society
30. Saboori, A., Jiang, G., Chen, H.: Autotuning configurations in distributed systems for performance improvements using evolutionary strategies. In: 2008 The 28th International Conference on Distributed Computing Systems, pp. 769–776, June 2008. https://doi.org/10.1109/ICDCS.2008.11
31. Sarkar, S., Chatterjee, S., Misra, S.: Assessment of the suitability of fog computing in the context of internet of things. IEEE Trans. Cloud Comput. 6(1), 46–59 (2018)
32. Shin, K.G., Chang, Y.: Load sharing in distributed real-time systems with state-change broadcasts. IEEE Trans. Comput. 38(8), 1124–1142 (1989)
33. Sung, W., Chiang, Y.: Improved particle swarm optimization algorithm for android medical care IOT using modified parameters. J. Med. Syst. 36(6), 3755–3763 (2012). https://doi.org/10.1007/s10916-012-9848-9
34. Tata, S., Jain, R., Ludwig, H., Gopisetty, S.: Living in the cloud or on the edge: opportunities and challenges of IoT application architecture. In: 2017 IEEE International Conference on Services Computing (SCC), pp. 220–224, June 2017. https://doi.org/10.1109/SCC.2017.35
35. Yang, X.S.: Bat algorithm for multi-objective optimisation. arXiv preprint arXiv:1203.6571 (2012)

36. Yi, S., Li, C., Li, Q.: a survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data, Mobidata@MobiHoc 2015, Hangzhou, China, 21 June 2015, pp. 37–42 (2015)

37. Yousefpour, A., Ishigaki, G., Jue, J.P.: Fog computing: towards minimizing delay in the internet of things. In: IEEE International Conference on Edge Computing, EDGE 2017, Honolulu, HI, USA, 25–30 June 2017, pp. 17–24 (2017)