On the Fog-Cloud Cooperation: How Fog Computing can address latency concerns of IoT applications

Amir Karamoozian
Network Research Lab, DIRO
University of Montreal
Montreal, Canada
amir.karamoozian@umontreal.ca

Abdelhakim Hafid
Network Research Lab, DIRO
University of Montreal
Montreal, Canada
ahafid@iro.umontreal.ca

El Mostapha Aboulhamid
DIRO
University of Montreal
Montreal, Canada
em.aboulhamid@umontreal.ca

***Abstract*—Fog computing emerged as a new computing paradigm which moves the computing power to the proximity of users, from core to the edge of the network. It is known as the extension of Cloud computing and it offers inordinate opportunities for real-time and latency-sensitive IoT applications. An IoT application consists of a set of dependent Processing Elements (PEs) defined as operations performed on data streams and can be modeled as a Directed Acyclic Graph (DAG). Each PE performs a variety of low-level computation on the incoming data such as aggregation or filtering. A key challenge is to decide how to distribute such PEs over the resources, in order to minimize the overall response time of the entire PE graph. This problem is known as distributed PE scheduling and placement problem. In this work, we try to address the question of how fog computing paradigm can help reducing the IoT application response time by efficiently distributing PE graphs over the Fog-Cloud continuum. We mathematically formulate the fundamental characteristics of IoT application and Fog infrastructure, then model the system as an optimization problem using Gravitational Search Algorithm (GSA) meta-heuristic technique. Our proposed GSA model is evaluated by comparing it with a well-known evolutionary algorithm in the literature via simulation. Also, a comparative analysis with the legacy cloud infrastructure is done in order to show the significant impact of fog presence on the performance of PE processing. Evaluation of our model demonstrates the efficiency of our approach comparing to the current literature.**

***Index Terms*—Internet of Things, IoT stream processing, Fog/Cloud Computing, Scheduling and Placement optimization.**

I. Introduction

Internet of Things (IoT) is an emerging paradigm which is primarily founded as the network of intelligent and self configuring nodes (*things*) interconnected in a dynamic and global network infrastructure [1]. The rise of these connected smart devices provide new opportunities to be practically leveraged in a broad spectrum of applications in different areas, such as smart cities, smart homes or smart healthcare.

It is estimated that by 2020, the number of connected devices will reach 24 billion [2]. From one side, all these IoT devices are constantly generating massive amount of data and, from the other side, most often they need to execute rapid analysis on the data and provide instant results. Generally, the current state of existing cloud infrastructure is limited to only two unsatisfactory actions: either (1) offload and process all input IoT data in the cloud which cause additional communication costs and latency, or (2) process all data locally which is not possible specially in case of applications with high-computational overhead. Due to these decifiencies that Cloud computing incurs (additional latency for computations and a lot of bandwidth consumption), it is not capable of supporting millions of things spread over wide geo-distributed areas. Hence, a new computing paradigm introduced by Cisco [3] known as Fog Computing (a.k.a. edge computing). It is indeed an extension of the cloud computing paradigm which moves the computing resources to the proximity of users from the core to the edge of the network (Fig. 1). Indeed, it is a virtualized platform which sits between traditional cloud servers and IoT devices, providing storage, computation and networking services.

An IoT application consists of a set of dependent Processing Elements (PEs) defined in terms of operations and can be modeled as a Directed Acyclic Graph (DAG). Each PE (packaged as a lightweight virtual machine, a.k.a. container [4]) performs a variety of low-level computation tasks according to specific streaming applications and graph edges represent a stream flowing downstream from the producing vertex to the consuming vertex. Besides describing the high-level abstractions of processing elements, DAG models greatly facilitate parallel computations of PEs, which are adopted by many high-performance distributed stream processing engines. For instance, Apache Storm[1] uses a DAG network topology consisting of *spouts* and *bolts*, where spouts produce new streams, and bolt consumes injected streams as input and produces streams as output. A key challenge is to decide how

[1]Apache Storm project: https://storm.apache.org

978-1-7281-1796-6/19/$31.00 ©2019 IEEE

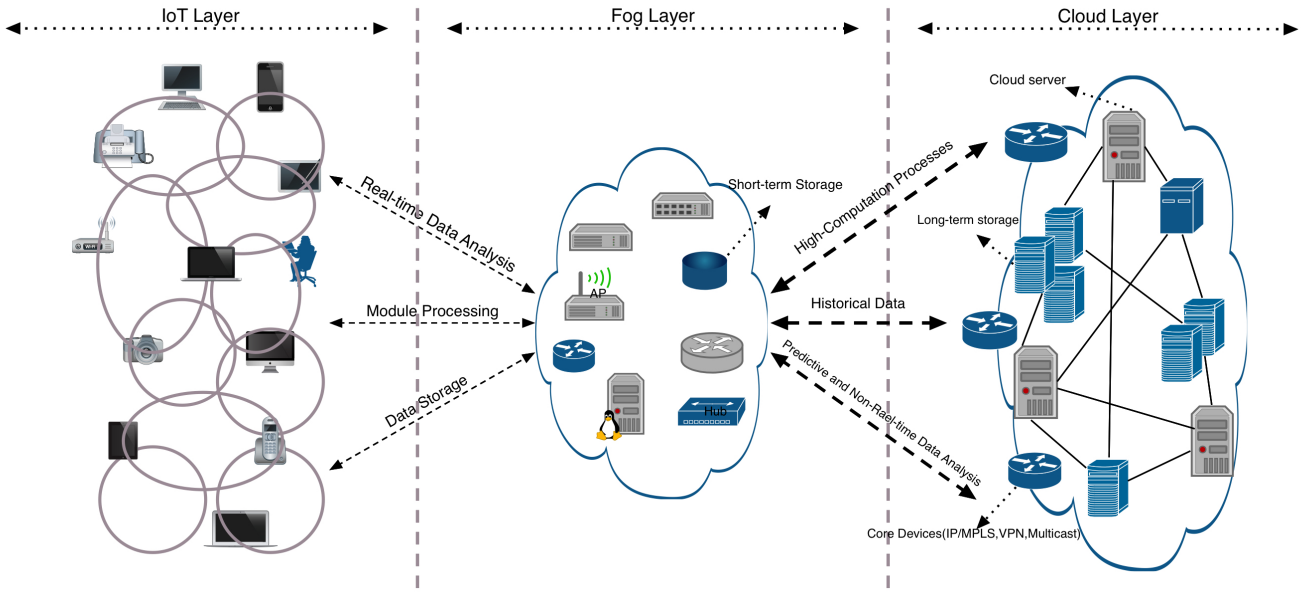

Figure 1: Fog Computing layered architecture

to distribute such PEs over the Fog-Cloud continuum, in order to minimize the overall response time of the entire graph of PEs. This problem is referred to as distributed PE scheduling and placement problem.

There are several challenges in distributed PE scheduling and placement problem because (i) infrastructure resources are heterogeneous and geo-distributed, and resources closer to the data source are more constrained while their power naturally increase as we travel along the hierarchy. Such resource asymmetry requires careful modeling of the trade-off between computation and communication latency; (ii) processing elements are interdependent and the scheduling and placement algorithm has to optimize the entire PE processing without violating the dependency constraints.

This work delivers an approach to address the question of how fog computing paradigm can help reducing the IoT applications response time by efficiently distributing PE graphs over the Fog-Cloud continuum. One may say that by placing the PEs as close as possible to the edge devices we can reduce transmission costs. However, fog nodes close to the edge are likely to have limited processing and storage capabilities. So they may not be capable of handling sophisticated PEs. Thus, an efficient scheduling and placement algorithm with higher performance is required in order to minimize the latency/response time [2] of IoT applications taking into account the overall computation and communication cost constraints.

In this paper, we use Gravitational Search Algorithm (GSA) [5] to address the impact of the fog computing paradigm on the distributed PE scheduling and placement problem and, hence, our contributions can be summarized as follows:

- We propose a model to optimize the response time of PE processing across Fog-Cloud continuum. Our model considers the imposed computation and communication cost limitations on Fog resources.
- We encode and formulate the PE scheduling and placement problem with a meta-heuristic optimization algorithm named GSA.
- We evaluate the efficiency of our proposed approach by comparing it with Particle Swarm Optimization (PSO) [6] which is one of the well-known evolutionary algorithms in the literature.

The rest of the paper is organized as follows. In Section II, we discuss related work in PE scheduling and placement problem. In Section III, we present the Fog computing system model and formally define the PE scheduling and placement problem. Section IV gives a background on GSA algorithm and encode it to our problem of interest. Section V evaluates the simulation results, and Finally, Section VI concludes the paper.

II. Related work

The advantages of fog computing over traditional cloud computing are increasingly attracting the attention of researchers and organizations for its capabilities and benefits [7]. Despite significant advantages of fog computing, the researchers have mostly focused on its preliminary notions and design principles [3], [8].

Furthermore, there are several researches on IoT application deployment on Fog-Cloud resources [9]–[14]. In [9], the authors suggested an approach for application placement

[2]Throughout the paper, the terms 'latency' and 'response time' are sometimes used interchangeably.

problem in Fog-enabled infrastructure for IoT. They proposed an optimization model to jointly study application placement and data routing. Their approach can also be interpreted as finding the minimum congestion ratio of the links such that all IoT demands are fully satisfied. However, (1) they simply assume an IoT application only as one independent microservice (processing element) while complex applications are made up of multiple dependent microservices; (2) they modeled the infrastructure only as a set of fog-enabled nodes located at the edge, not joined with the nodes at the cloud tier. Similarly, in [10] and its extension [11], a simulation-based study presented in which the authors modeled a conceptual framework for Fog resource provisioning with the aim of maximizing the number of assignments of available Fog resources to tasks at the edge; however, they only assume the requests only as independent single tasks. In [12], authors proposed a VM allocation solution in distributed cloud architecture which takes into account time-sensitive/real-time processing; however, (1) their approach only considers resources at the Cloud, not the ones located at the Fog levels (no multi-tier solution involved); (2) no dependency between VMs is involved (Each application is considered to be packed and implemented in one single VM).

Hence, our proposed approach differs from the aforementioned approaches in a sense that (i) we model an IoT application as a DAG graph consisting of dependent processing elements to represent heterogeneous complex IoT applications; and (ii) it takes advantage of high performance and more efficient population-based meta-heuristic algorithm, GSA, which outperforms other well-known evolutionary algorithms in the literature.

III. System Model and Problem Formulation

In this section, we, first, provide basic definitions of all the key concepts related to the optimal PE scheduling and placement of IoT applications. Then, we formulate the optimization problem and its constraints.

IoT Application Model: Assume that each IoT application consists of n PEs. Each PE is modeled as a triplet $PE_i = (\omega_i, d^i_{in}, d^i_{out})$, where ω_i represents workload of PE i, d^i_{in} and d^i_{out} are respectively input and output stream data size of PE i, $1 \leq i \leq n$. Thus, the total workload of all PEs is $\sum_{i=1}^{n} \omega_i$.

The PE processing graph of an IoT application is modeled as $AG = (V, E)$ where $V = \{PE_1, PE_2, PE_3, ..., PE_n\}$ is the finite set of PEs to be executed either on Fog or Cloud and E is the set of edges indicating the amount of stream data flowing between two vertices. Two nodes PE_u and PE_v are called adjacent if there exists an edge $e_{u,v}$ of the form (PE_u, PE_v) which represents stream data dependency between the PEs and is associated with a non-negative weight. Since the graph is directed, for each edge pair (PE_u, PE_v), PE_u is called the parent node of PE_v and PE_v is called the child node of PE_u. Furthermore, each processing graph is associated with a deadline γ_w which represents the latency sensitivity of the graph such that its entire processing time should not exceed that deadline.

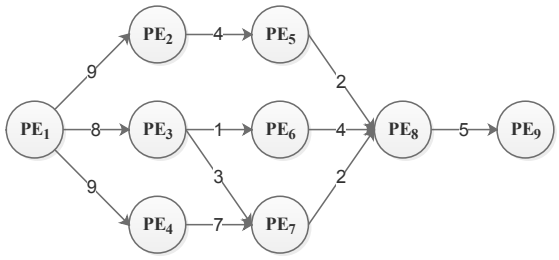

Figure 2: A sample PE graph in which each vertex represents a Processing Element and each edge represents downstream cost from the producing vertex to the consuming vertex

Resource Model: Now let $IG = (N, L)$ be an infrastructure graph which denotes the topology of a cloud-fog infrastructure, where the set of vertices $N = \{n_1, n_2, ..., n_m\}$ identifies the set of nodes, each representing a Fog or a Cloud resource and the link $l_{ij} \in L$ denotes a link between resource n_i and n_j. Each node $n_i, 1 \leq i \leq m$, has its resource capacity $R_{i,k}$ of resource type k, where $i \in N$ and $k \in K$[3]. K refers to resource types and here is defined as node's CPU processing capability Cap_{n_i}. Each physical link (i, i') has its bandwidth capacity of B. If we denote N_{cld} as a set of cloud nodes and N_{fog} as a set of fog nodes, we will have $N = N_{cld} \cup N_{fog}$.

Given that, we can now calculate latency and cost parameters. Latency consists of processing latency and transmission latency. Processing latency of PE_u on a given resource n_i is denoted by $\ell^{n_i}_{ex}(PE_u)$ and calculated as:

$$\ell^{n_i}_{ex}(PE_u) = \frac{\omega_u}{Cap_{n_i}} \qquad \forall u, i : 1 \leq u \leq n, 1 \leq i \leq m \quad (1)$$

where ω_u denotes the workload that PE_u imposes on resource n_i. Transmission latency refers to the time it takes to transmit d units of data produced by PE_u downstreamed towards PE_v and is calculated as:

$$\ell_{tr}(PE_u, PE_v) = \frac{d_{(u,v)}}{B} \qquad \forall u, v : 1 \leq u, v \leq n \quad (2)$$

where $d_{(u,v)}$ is the output data d^u_{out} and B is the amount of available bandwidth between resources.

Given processing and transmission latencies, total latency is additive over the PEs and edges of the PE graph. This means that total latency of an individual PE recursively depends on the latency of its preceding PEs. Consider $L(PE_u)$ as the total accumulated latency of PE_u. It can be recursively calculated as the processing latency of PE_u itself, plus the maximum latency incurred by its preceding PEs and edges:

$$L(PE_u) = \ell^{n_i}_{ex}(PE_u) + \max_{p \in P(u)} \left\{ L(PE_p) + \ell_{tr}(PE_p, PE_u) \right\} \quad (3)$$

[3]In general, K may contain any resource type e.g., CPU, RAM, storage and network.

where $P(u)$ refers to the preceding PEs of node PE_u. We use *max* function to determine the maximum latency (slowest link) up to node PE_u. For better understanding, let us consider the processing graph/workflow shown in Fig. 2. The preceding PEs of PE_8 are PE_5, PE_6 and PE_7. For each preceding PE_p, total accumulated latency adds latency up to node PE_p with the transmission latency from PE_p to PE_u. Therefore, total accumulated latency takes into account the maximum delay value it can take to process a PE graph/workflow.

Similar to latency, cost function also includes execution cost and transmission cost. Execution cost $C_{ex}^{n_i}(PE_u)$ is defined as the cost of deploying PE_u on resource n_i. Transmission cost $C_{tr}^{n_i,n_j}(PE_u, PE_v)$ represents the cost of transferring data on a particular physical link between n_i and n_j if accommodating PE_u and PE_v. Thus, the total cost C_{tot} can be defined as follows:

$$C_{tot} = \sum_{i=1}^{m}\sum_{u=1}^{n} c_i^p . C_{ex}^{n_i}(PE_u) + \sum_{(i,j)\in L}\sum_{(u,v)\in E} c_{ij}^{tr} . C_{tr}^{n_i,n_j}(PE_u, PE_v) \tag{4}$$

where c_i^p represents the unit cost of processing in MIPS and c_{ij}^{tr} shows the unit cost of data transmission per unit BW. This cost may for instance represent the pricing policy imposed by a service provider.

In order to find an efficient scheduling and placement policy to distribute a PE processing graph along the Fog-Cloud resources, we model a schedule tuple $S = (N, PE, L, C)$ in terms of the set of Fog-Cloud nodes, PE mapping, total latency and total cost. Based on this model, the problem can be formally defined as follows: *Given a PE processing graph to execute, find an efficient scheduling and placement strategy S that minimizes the latency of the whole PE workflow, while not exceeding the cost constraint*. This definition results in the following optimization problem:

$$\begin{aligned} &\textbf{Minimize} \quad L(PE_n) \\ &\textbf{s.t.} \quad C_{tot} \leq \psi \end{aligned} \tag{5}$$

where ψ represents the cost policy threshold defined by the service provider.

IV. Proposed GSA modeling

To solve this problem, GSA algorithm is proposed as a well-suited approach due to its robustness and effectiveness [5], [15]. It was originally derived from Newton's theory of gravity and mass interactions to solve specially non-linear optimization problems. GSA is an optimization algorithm that is made up of particles interacting with each other based on the Law of Gravity. Each particle has a mass which indicates the particle's performance metric. In particular, GSA follows the Newton's theory concept which states that 'every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them' [16]. Particles tend to move towards the heavier particles under the gravity force law. GSA can also be viewed as a collection of agents each representing a solution to the problem. At each epoch, agents with heavier masses attract agents with lighter masses and, at the end, the heaviest mass in the search space demonstrates an optimal solution to the problem. Mathematically speaking, let's assume a system with n masses in which the position of ith mass with respect to other masses is:

$$X_i = (x_i^1, ..., x_i^d, ..., x_i^n)$$

where x_i^d indicates the position of ith mass in the dth dimension. The gravitational force of each particular mass j influencing another mass i at time t can be calculated as:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)) \tag{6}$$

where $M_{pi}(t)$ and $M_{aj}(t)$ identify respectively passive and active gravitational masses i and j. $R_{ij}(t)$ is defined as the Euclidean distance between masses i and j, and $G(t)$ is the gravitational constant at time t.

The total force that acts on mass i in dimension d at time t can be formulated as:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{n} rand(j) \times F_{ij}^d(t) \tag{7}$$

Note that in order to apply stochastic characteristic to GSA, the sum of all forces exerted from other masses are multiplied by a random number $rand(j)$ in the range of [0,1]. Now, given the total force, we can easily obtain the acceleration of each mass i at time t by the law of motion. So, we have

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \tag{8}$$

where $M_i(t)$ shows the inertia mass of i.

Also, based on the law of motion, the velocity v and mass position x of mass i in its dth dimension at time $t+1$ can be respectively formulated as:

$$\begin{aligned} v_i^d(t+1) &= rand(i) \times v_i^d(t) + a_i^d(t) \\ x_i^d(t+1) &= x_i^d(t) + v_i^d(t+1) \end{aligned} \tag{9}$$

As stated before, each mass value is proportional to the mass attractiveness. This means that heavier masses are more likely to attract others and move at slower paces comparing to the lighter ones at each epoch. Now let $fit_i(t)$ denote the fitness value of mass i at time t. In order to update the gravitational and inertial masses, we have

$$\begin{aligned} m_i(t) &= \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \\ M_i(t) &= \frac{m_i(t)}{\sum_{j=1}^{n} m_j(t)} \end{aligned} \tag{10}$$

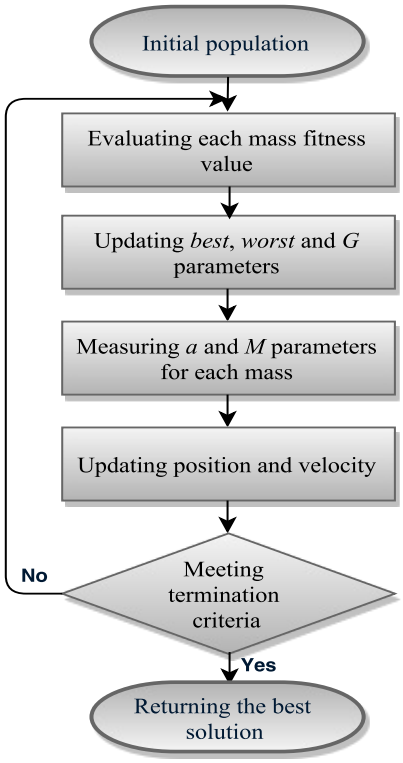

Figure 3: How GSA Algorithm works

Furthermore, since we are modeling a minimization problem, $best(t)$ and $worst(t)$ will be formulated as:

$$\begin{aligned} best(t) &= \min_{j\epsilon\{1,\ldots,N\}} fit_j(t) \\ worst(t) &= \max_{j\epsilon\{1,\ldots,N\}} fit_j(t) \end{aligned} \tag{11}$$

For the sake of simplicity, we illustrate GSA procedures in Fig. 3. However readers are encouraged to check [5] for further details.

A. GSA versus PSO

To evaluate our approach, we compare our solution with one of the well-known meta-heuristic algorithms in optimization problems, PSO. We will show that our proposed methodology with GSA outperforms PSO. First, we need to differentiate the main differences between GSA and PSO as follows:

- Both GSA and PSO algorithms perform optimization following the same concept of mass movement in the search space; however, their movement strategies are different.
- Mass direction in PSO is obtained using only two position values, i.e., $pbest$ and $gbest$; however, mass direction in GSA is defined by the total force from all other masses.
- During the update procedure, PSO does not consider the quality of solutions and the fitness values, while in GSA, the force is proportional to the fitness value.
- Update procedure in PSO does not take the distance between solutions into account, while in GSA, the force is inversely proportional to the distance between solutions.
- In velocity update procedure in PSO, $pbest$ and $gbest$ are required to be stored in memory, while in GSA, the algorithm is memory-less and only considers the current position of the masses.

B. Encoding GSA to our model

To model our problem with GSA algorithm, we need to address the following questions: (i) how to encode the problem, or, in other words, how to interpret the solution? (ii) how to define the fitness function and calculate the problem efficiency?

Regarding the first question, in order to address the encoding problem, we first need to clearly define the notions of mass and dimensions. In our case, a mass is defined as a set of PE requests. Therefore, the dimension of a mass shows the number of PE requests at a time. The dimension of each mass also represents the coordinate system, and the coordinate system identifies the position of the mass. For example, if the mass has five dimensions, its position will be identified by five coordinates; or if it has six dimensions, its position will be identified by six coordinates and so on.

To fully understand the mass definition and its dimensions, an example is provided in table I in which a mass with nine dimensions is depicted (i.e., nine PE requests), and their positions are determined by nine coordinates (C1 through C9). The value of each specific coordinate can take a number between 0 and the number of available resources (3 in this example). Each coordinate consists of a real number $\mathbb{R}$ where its decimal part corresponds to the assigned fog/cloud resource index. For instance, coordinate 2, indicated as C2 in table I, refers to a PE with a value of 3.2. The decimal part of this value indicates the resource number (i.e., 3) that particular PE should be mapped/assigned to. The same rule applies to other coordinates.

Table I: Mass Positions

C1	C2	C3	C4	C5	C6	C7	C8	C9
2.3	3.2	1.1	2.1	1.3	3.1	2.1	1.3	1.0

With this mechanism, we can identify where to reside each particular PE along Fog/Cloud resource continuum. Table II demonstrates this PE to Fog/Cloud resource mapping based on the values provided for our example in table I.

Regarding the second question in GSA modeling, fitness function should be defined precisely, because it plays a key role in the result evaluation process. Indeed, fitness function evaluates the efficiency of solution candidates according to the objective of the problem, which in our case, is the total response time L. So we aim at minimizing the total response time L according to the schedule S. To do this, assume all agents that represent all solution candidates are first randomly initialized. After initialization step, gravitational forces and accelerations among agents are obtained using (6), (7) and (8). At each epoch, with the achieved so-far best solution, the velocities and positions of all masses will be updated using (9)

Table II: PE to Fog/Cloud resource mapping

$PE_1 \mapsto n_2$	$PE_2 \mapsto n_3$	$PE_3 \mapsto n_1$	$PE_4 \mapsto n_2$	$PE_5 \mapsto n_1$	$PE_6 \mapsto n_3$	$PE_7 \mapsto n_2$	$PE_8 \mapsto n_1$	$PE_9 \mapsto n_1$

and (10). This updating process ends when termination criteria are met.

V. Experimental Results

In this section, the efficiency of the proposed approach is evaluated via simulation. In order to technically evaluate GSA, it is compared with PSO, a well-known meta-heuristic algorithm used in [6], [17], [18]. We will show that our proposed methodology with GSA outperforms the one with PSO.

Generic parameters of simulation are set based on [5] as follows: Maximum iteration is set to 1000. Initial value of gravitational constant G_0 is set to 100. In PSO, correction factors c_1 and c_2 are set to 2, and Inertia value (ω) is assigned with the value of 1. Since we are focusing on the effects of fog in the context of IoT applications, we generate masses with different sizes to represent heterogeneity requirement of complex IoT applications. Thus, different number of PE requests are considered: 50, 100, 150, 200, 250 and 300. For both algorithms, results are averaged over 50 runs and the average, best and worst solutions are provided.

Fig. 4 shows the average, best and worst response time under different number of Fog/Cloud resources varying from 10 to 90. As illustrated, response time decreases as the number of resources increase. This is because more fog nodes provide more computing resources at the edge, leading to have more chance of handling the requests at the edge, and thus results in a lower response time.

As stated, to validate and compare our approach, the average, best and worst solutions for response time are compared in the case of both algorithms, GSA and PSO. As demonstrated in Fig. 5, by increasing the number of PE requests, our proposed GSA method, comparing to PSO, finds a more efficient scheduling and placement strategy in terms of the total response time.

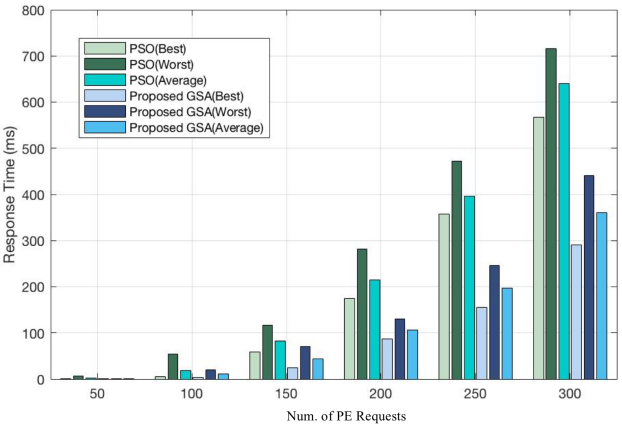

Figure 5: Response Time vs. Number of PE Requests

Furthermore, in order to show the significant non-negligible effect of having fog infrastructure, we run a comparative analysis on PE graph processing when Fog resources are leveraged versus a baseline scenario where no fog resource presents (i.e., the legacy cloud infrastructure). Investigations reveal that the average response time is reduced by over 20% in the presence of fog infrastructure. The achieved results with the proposed approach highlight the potential of using fog computing in decreasing the average response time for different number of PE requests, and thereby improving QoS.

VI. Conclusion

Fog computing is indeed a new computing paradigm which is known as the extension of Cloud computing to the edge and it offers inordinate opportunities for real-time and latency-

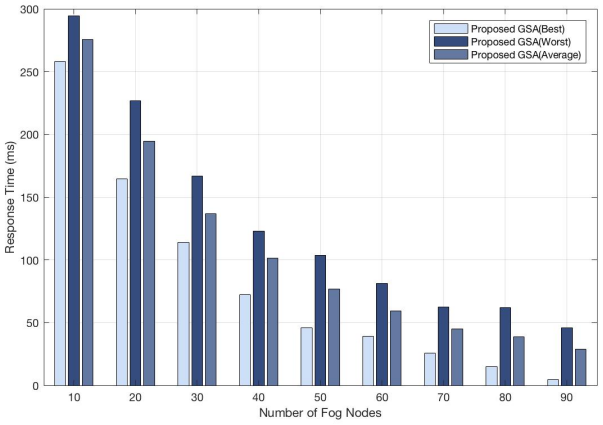

Figure 4: Response Time vs. Number of Fog Nodes

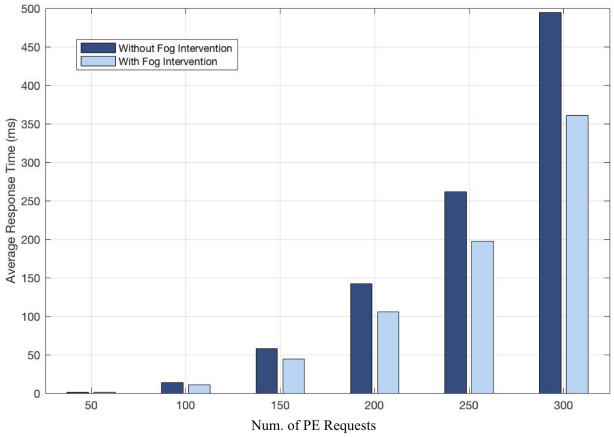

Figure 6: Avg. Response Time with/out fog intervention

sensitive IoT applications. In this paper, in summary, we investigated one of the key challenges in this field named distributed PE scheduling and placement problem which studies how to distribute PEs over the Fog-Cloud continuum in order to minimize the overall response time over the entire PE graph. We mathematically formulated the problem as an optimization problem using an algorithm called GSA. We encoded GSA to our model and defined the appropriate fitness function for our problem of interest. Finally, we evaluated our proposed solution by comparing it with one of the well-known approaches proposed in the literature, i.e., PSO. Simulation results demonstrated efficiency of our approach comparing to the current literature.

REFERENCES

[1] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: A survey. *Future Generation Computer Systems*, 56(Supplement C):684 – 700, 2016.

[2] M. Aazam and E. N. Huh. Dynamic resource provisioning through fog micro datacenter. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*, pages 105–110, March 2015.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[4] B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe. Evaluation of docker as edge computing platform. In *2015 IEEE Conference on Open Systems (ICOS)*, pages 130–135, Aug 2015.

[5] Esmat Rashedi, Hossein Nezamabadi-pour, and Saeid Saryazdi. Gsa: A gravitational search algorithm. *Information Sciences*, 179(13):2232 – 2248, 2009. Special Section on High Order Fuzzy Sets.

[6] Soma Prathibha, B Latha, and G Suamthi. Particle swarm optimization based workflow scheduling for medical applications in cloud. *Biomedical Research*, 2017.

[7] Yu Cao, Peng Hou, Donald Brown, Jie Wang, and Songqing Chen. Distributed analytics and edge intelligence: Pervasive health monitoring at the era of fog computing. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 43–48, New York, NY, USA, 2015. ACM.

[8] Dragi Kimovski, Humaira Ijaz, Nishant Saurabh, and Radu Prodan. Adaptive nature-inspired fog architecture. In *Fog and Edge Computing (ICFEC), 2018 IEEE 2nd International Conference on*, pages 1–8. IEEE, 2018.

[9] R. Yu, G. Xue, and X. Zhang. Application provisioning in fog computing-enabled internet-of-things: A network perspective. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 783–791, April 2018.

[10] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner. Resource provisioning for iot services in the fog. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 32–39, Nov 2016.

[11] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized iot service placement in the fog. *Service Oriented Computing and Applications*, 11(4):427–443, Dec 2017.

[12] Fang Hao, Murali Kodialam, T. V. Lakshman, and Sarit Mukherjee. Online allocation of virtual machines in a distributed cloud. *IEEE/ACM Trans. Netw.*, 25(1):238–249, February 2017.

[13] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang. Cost efficient resource management in fog computing supported medical cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 5(1):108–119, Jan 2017.

[14] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal*, 3(6):1171–1181, Dec 2016.

[15] A. Zarrabi, E. K. Karuppiah, Y. K. Kok, N. C. Hai, and S. See. Gravitational search algorithm using cuda. In *2014 15th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 193–198, Dec 2014.

[16] S. Mirjalili and S. Z. M. Hashim. A new hybrid psogsa algorithm for function optimization. In *2010 International Conference on Computer and Information Application*, pages 374–377, Dec 2010.

[17] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair. Efficient task scheduling multi-objective particle swarm optimization in cloud computing. In *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, pages 17–24, Nov 2016.

[18] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 400–407, April 2010.