Future Generation Computer Systems 131 (2022) 121–136

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

QoS-aware placement of microservices-based IoT applications in Fog computing environments

Samodha Pallewatta *, Vassilis Kostakos, Rajkumar Buyya

The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 21 September 2021
Received in revised form 5 December 2021
Accepted 19 January 2022
Available online 24 January 2022

Keywords:
Fog computing
Microservice applications
Internet of Things
Application placement
QoS-awareness

ABSTRACT

The Fog computing paradigm, offering cloud-like services at the edge of the network, has become a feasible model to support computing and storage capabilities required by latency-sensitive and bandwidth-hungry Internet of Things (IoT) applications. As fog devices are distributed, heterogeneous and resource-constrained, efficient application scheduling mechanisms are required to harvest the full potential of such computing environments. Due to the rapid evolution in IoT ecosystems and also to better suit fog environment characteristics, IoT application development has moved from the monolithic architecture towards the microservices architecture that enhances scalability, maintainability and extensibility of the applications. This architecture improves the granularity of service decomposition, thus providing scope for improvement in QoS-aware placement policies. Existing application placement policies lack proper utilisation of these features of microservices architecture, thus failing to produce efficient placements. In this paper, we harvest the characteristics of microservice architecture to propose a scalable QoS-aware application scheduling policy for batch placement of microservices-based IoT applications within fog environments. Our proposed policy, QoS-aware Multi-objective Set-based Particle Swarm Optimisation (QMPSO), aims at maximising the satisfaction of multiple QoS parameters (makespan, budget and throughput) while focusing on the utilisation of limited fog resources. Besides, QMPSO adapts and improves the Set-based Comprehensive Learning Particle Swarm Optimisation (S-CLPSO) algorithm to achieve better convergence in the fog application placement problem. We evaluate our policy in a simulated fog environment. The results show that compared to the state-of-the-art solutions, our placement algorithm significantly improves QoS in terms of makespan satisfaction (up to 35% improvement) and budget satisfaction (up to 70% improvement) and ensures optimum usage of computing and network resources, thus providing a robust approach for QoS-aware placement of microservices-based heterogeneous applications within fog environments.

1. Introduction

The growing popularity of the Internet of Things (IoT) paradigm has resulted in the rapid increase of IoT applications and the number of smart devices that are connected to such data analytic platforms. For the processing and storage of the data generated by the smart devices, first, Cloud computing was identified as a viable solution. But transmitting large amounts of data from IoT devices towards the cloud results in higher network congestion and higher latency due to cloud data centres being multiple hops away from the data sources. As a solution to these issues, the Fog computing paradigm emerged, where cloud-like services are provided at the edge of the network by using devices with computational, storage, and networking capabilities that reside within the path connecting IoT devices to the cloud data centres [1]. Fog computing resources are distributed, heterogeneous and resource-constrained compared to cloud resources, which has resulted in driving application development towards modular, decoupled architectures such as microservices architecture so that limited edge resources can be utilised for latency-critical and bandwidth-hungry application modules while placing the rest of the modules on the cloud resources. In literature, the Osmotic computing paradigm also follows this concept [2,3].

Microservices architecture enables the design and development of applications as a collection of small and modular components known as 'microservices' that communicate through lightweight protocols to perform end-user requested services [4]. In this paper, we use the term 'service' to refer to functionalities of the IoT application, accessed by end-users. In contrast to this, a 'microservice' represents a fine-grained application

* Corresponding author.
E-mail address: ppallewatta@student.unimelb.edu.au (S. Pallewatta).

https://doi.org/10.1016/j.future.2022.01.012
0167-739X/

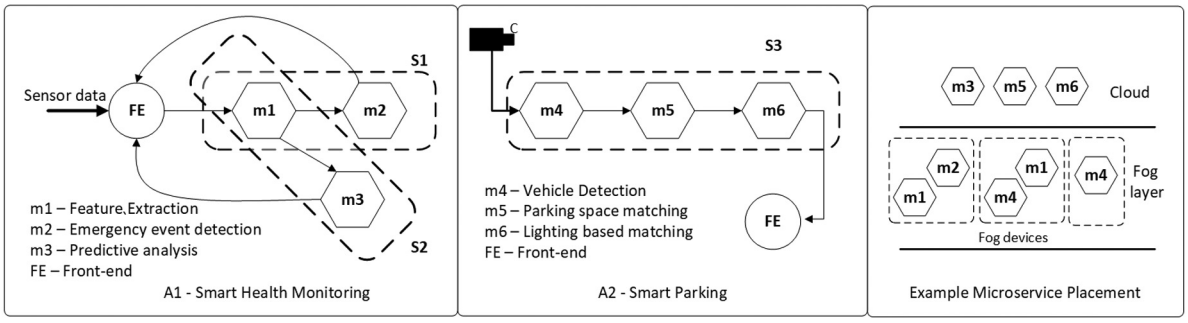

Fig. 1. Example scenarios for IoT application placement.

component. As a result, each service can be defined as a composition of one or more microservices. Being a fine-grained, scalable, and extensible design, microservices architecture not only aids the agile development of IoT applications but also helps in achieving elasticity and reliability with ease in the resource-constrained and heterogeneous fog computing environments [5,6]. A microservices-based IoT application would contain multiple services with heterogeneous quality requirements and characteristics (i.e., latency-critical, latency tolerant, high bandwidth, etc.). Moreover, due to the fine granularity of the architecture, a single end-user service can consist of multiple microservices with data dependencies among them and microservices can also be part of more than one service. Such application design enables the definition of per service quality requirements in terms of parameters like makespan, budget, and throughput. By properly utilising this information and handling the complexities introduced by the granularity of inter-connected microservices, application placement policies can harness both fog and cloud resources optimally to maximise Quality of Service (QoS) satisfaction.

Due to latency and bandwidth improvements at the edge of the network, resource providers can charge higher prices for fog resources compared to the cloud resources [7,8], which has led many existing placement policies to consider minimising total latency and cost to reach a trade-off between the two [9–11]. But due to limited resources, makespan and budget aware prioritising is crucial to distribute fog resources among competing applications or services. Per service makespan and budget expectations defined for microservice applications along with batch placement can enable this. Moreover, throughput expectations of the services can be used to reap the benefits of vertical and horizontal scalability of the microservices to make maximum use of heterogeneous fog resources to minimise the effect of resource limitations on application performance.

As a motivating scenario, Fig. 1 presents a smart health care application for patient monitoring ($A1$) [12] and a smart city application for parking occupancy detection ($A2$) [13]. $A1$ consists of three microservices that communicate together to provide two services to the user. m_1, m_2 form a latency-critical emergency alert service, whereas $m1$, $m3$ form a latency tolerant service that generates long term analysis reports for the user. $A2$ consists of three microservices forming a single service that detects parking spot occupancy in real-time. Due to the microservice-based decomposition of the applications, QoS requirements (i.e, makespan, budget, throughput etc.) can be defined separately for each service. Using this knowledge, placement decisions can be made to satisfy the QoS requirements of each service by utilising both fog and cloud resources (i.e, $m1$ and $m2$ are mapped to fog layer devices to satisfy stringent latency requirements; $m4$ is mapped to fog layer to reduce the amount of data sent towards the cloud; $m3$, $m5$ and $m6$ are mapped to the cloud to satisfy their high computation resource requirements). Furthermore, microservices placed in fog can be horizontally scaled to satisfy the throughput requirements of the services under resource limitations of fog devices (i.e, two instances of $m1$ placed on two separate fog devices when a single device does not have enough processing power to support the request rate). $A1$ and $A2$ represent two heterogeneous applications trying to utilise fog resources. In the example scenario, the service S_1 in $A1$ is more latency-critical compared to the service $S3$ in $A2$ and due to the resource-constrained and heterogeneous nature of the fog devices, batch placement of applications has the potential to prioritise $S1$ over $S3$ to ensure QoS satisfaction.

Although fog application placement has been studied extensively, microservices architecture provides a novel perspective, where per service quality requirements and independently scalable nature of the microservices can enable harnessing the power of both the fog devices and cloud data centres to improve application performance through batch placement. Research on this is still at its early stages and has much room for improvement. Therefore, in this work, we propose a QoS-aware placement algorithm that improves the total QoS satisfaction considering multiple QoS parameters (makespan, cost, and throughput) and at the same time, ensures optimum resource usage through collaboration among fog and cloud resources. **The key contributions** of our work are as follows:

1. We formulate the fog application placement problem as a Lexicographic Combinatorial Optimisation Problem considering QoS satisfaction (in terms of makespan, budget, and throughput) as the primary objective and optimum resource usage as the secondary objective.
2. We propose an IoT application batch placement technique based on Set-based Comprehensive Learning Particle Swarm Optimisation (S-CLPSO). To improve the convergence rate, we introduce a heuristic-driven swarm initialisation and fitness parameter normalisation method and further incorporate a priority-based particle construction technique to overcome premature convergence due to the resource constraints of the fog devices.
3. We implement our policy using iFogSim2 [14] simulated fog environment and compare it against existing scheduling approaches based on their resultant QoS satisfaction and balanced fog and cloud resource usage.

The rest of the paper is organised as follows. Section 2 highlights related research followed by Section 3, which presents system architecture. In Section 4, the fog application placement problem and our proposed solution is detailed. Section 5 presents performance evaluation and Section 6 concludes the paper and draws future work and research directions.

2. Related work

In this section, we summarise existing work on fog application placement, compare them based on their key features and

also provide a detailed background on Particle Swarm Optimisation (PSO) algorithm and its derivatives used in designing our placement policy.

2.1. Application placement in fog environments

Existing research propose numerous algorithms to schedule applications within fog environments. They mainly fall under two categories: application offloading and application service placement, where offloading deploys application modules from client devices to the fog to be used by each client separately while service placement refers to the deployment of application services in the fog so that multiple clients can use them [15]. Since our work focuses on the latter, in this section we summarise research related to the application service placement in fog.

Brogi et al. [16] present a placement policy to place multi-component applications within a fog environment when inter-component link bandwidth and latency requirements are defined. They propose a heuristic placement algorithm consisting of two steps, where it first searches for all eligible nodes to host each application component based on its software and hardware requirements and then employs a greedy backtracking algorithm to place each component considering inter-component latency and bandwidth requirements. Yousefpour et al. [10] implement a framework that supports dynamic deployment and release of IoT services. Their work presents two separate greedy algorithms for minimising delay violation and minimising total cost for IoT services with delay constraints. In their work, each service is an independent task with an expected deadline for its completion and applications are built as a collection of such independent services. Skarlat et al. [11] propose a deadline-aware policy using Integer Linear Programming (ILP) to place applications within micro data centres know as fog colonies. They model applications as a set of independent tasks and define a deadline for the entire application. Their placement policy prioritises applications based on the deadline and tries to maximise the placement of applications within the fog layer such that for each application, the total deployment and execution time of the tasks does not exceed the application deadline. Skarlat et al. [17] extend the work proposed in [11] and solve the proposed optimisation problem using GA. GA based approach is evaluated against a mathematical programming based optimisation method. Results indicate that the GA algorithm can reduce deployment delays. Xie et al. [9] present a workflow application scheduling algorithm based on Particle Swarm Optimisation aiming to minimise the weighted sum of total latency and cost. They propose a non-local convergent PSO algorithm introducing a non-linear inertia weight calculation method along with a directional search process.

Deng et al. [18] form the microservices-based application scheduling problem in fog to minimise the cost of application deployment adhering to resource constraints and expected response time of the mobile services. The placement problem is solved through ILP. Their placement algorithm handles only the placement of a single application each turn. Thus, prioritising applications based on their latency requirements is not captured in their work. Guerrero et al. [15] compare three evolutionary algorithms; Weighted Sum Genetic Algorithm (WSGA), non-dominated sorting genetic algorithm (NSGA-II) and multi-objective evolutionary algorithm based on decomposition (MOEA/D), for solving fog service placement focusing on optimising latency, service spread and use of resources.

Chen et al. [19] apply Set-based Comprehensive Learning Particle Swarm Optimisation (S-CLPSO) algorithm for workflow application scheduling in cloud environments to satisfy user-defined QoS constraints in terms of deadline, budget and reliability. The proposed algorithm allows the user to select one of the QoS parameters as the optimisation objective while keeping the other two as constraints. Meta-heuristic algorithm combined with multiple heuristics to speed up the search process provides better results in satisfying QoS requirements. Verma et al. [20] propose a non-dominated sorting based PSO for minimisation of execution time, cost and energy consumption for workflow scheduling in cloud environments.

Table 1 compares features of the related works with our work in terms of three main categories; application model, placement properties and algorithm type. Features analysed under the application model aim to capture the granularity of the components (i.e., microservices, modules) of the application and their data dependencies. Application composition analyses each application model based on the number of services and service composition based on the collaboration pattern of the components that work together to perform a service. In literature, the term "service" is used for modules, microservices, components, processes etc. In our work, we define service from a user perspective where it represents a business functionality accessed by the user. Due to the fine-grained design of microservices, a service can consist of multiple microservices communicating together to perform a service. As existing works do not capture these features properly, we model our application placement problem to represents the granularity of the microservice design.

Placement properties characterise the works based on their ability to perform batch placement, decision parameters, fog/cloud balanced resource utilisation and scalability of application components. Decision parameters are analysed under two categories: QoS-aware parameters that represent quality expectations of the services in terms of makespan, budget and throughput, QoS-unaware parameters which focus on total makespan and budget of the placement irrespective of their QoS expectations. Due to resource constraints within fog environments, fog application placement can benefit from batch placement, QoS-awareness and optimum use of fog resources, which allow prioritising of services with stringent QoS requirements to achieve a balance between fog and cloud resource usage. Moreover, developing applications as microservices enables each microservice to scale independently, so that each microservice can be vertically or horizontally scaled based on the resource availability of heterogeneous fog devices.

Considered related works use three main types of algorithms to solve the application placement problem; heuristic, meta-heuristic and mathematical programming. As heuristic and greedy algorithms are unable to handle multiple objectives, [10,16] propose multiple heuristic algorithms where each focuses on one of the decision parameters. Mathematical programming-based approaches can only obtain the optimum solution when search space is small and not suitable for batch placement problems. Thus, due to the ability to handle multiple objectives and also to reach near-optimum solutions faster, meta-heuristic algorithms have become a popular approach in solving multi-objective optimisation problems. Meta-heuristics such as Genetic Algorithm (GA), Ant Colony Optimisation (ACO), Particle Swarm Optimisation (PSO) are popularly used in solving scheduling problems in both single objective and multi-objective scenarios. However, one of the main challenges when adapting meta-heuristics to the fog application placement is handling fog resource constraints without trapping the algorithm to a local optimum. This issue is not properly addressed in existing works in both fog and cloud placements. Proper use of heuristics to populate the initial solution and efficiently normalise weighted parameters is another area with scope for improvement. So in our work, we propose a placement technique based on Set-based Comprehensive Learning Particle Swarm Optimisation (S-CLPSO) and improve it to reach a near-optimum solution for batch placement of microservices-based IoT applications.

Table 1
Comparison of existing application placement policies.

Work	Environment		Application model						Placement properties								Algorithm type
	Cloud	Fog/ Edge	μservice architecture	QoS granularity	Application composition				Batch placement	Decision parameters						Scalability	
										QoS-aware			QoS-unaware		Resource		
					Services per app	Service composition: Single	Chained	Aggregator		Makespan	Budget	Throughput	Total makespan	Total budget	usage		
[16]		✓		Per link	Multiple	✓	✓			✓		✓					Heuristic
[10]		✓		Per service	Multiple	✓				✓		✓		✓			Heuristic
[11]		✓		Per app	Single	✓	✓		✓	✓							ILP solver
[17]		✓		Per app	Single	✓	✓		✓	✓							Meta-heuristic
[9]		✓		Per app	Single	✓	✓	✓					✓	✓			Meta-heuristic
[18]		✓	✓	Per app	Single	✓	✓			✓		✓		✓		✓	ILP solver
[15]		✓	✓	Per app	Multiple	✓	✓						✓		✓	✓	Meta-heuristic
[19]	✓			Per app	Single	✓	✓	✓		✓	✓						Meta-heuristic
[20]	✓			Per app	Single	✓	✓	✓		✓	✓						Meta-heuristic
Our work		✓	✓	Per service	Multiple	✓	✓	✓	✓	✓	✓	✓			✓	✓	Meta-heuristic

2.2. Particle swarm optimisation

Particle swarm optimisation (PSO) is a population-based meta-heuristic algorithm [21], which was originally introduced for the optimisation of problems defined in continuous solution space. In PSO, a set of solutions identified as a swarm of particles moves within the solution space using not only its own experience but also the experiences of other particles. Each particle is characterised based on two factors; its position and velocity. In each iteration, particles update their velocity taking their own best position (*pbest*) and best position of the swarm (*gbest*) into consideration and modify their position to move towards a better solution within the solution space. PSO is simple in concept, computationally inexpensive and has a higher convergence rate due to social sharing of information among particles in the swarm and use of previous experiences of particles for the decision making process.

As the traditional PSO is not designed for solving discrete optimisation problems, multiple approaches have been introduced to adapt PSO for discrete cases, including Binary PSO (BPSO), Discrete PSO (DPSO) etc. Among these, Chen et al. [22] propose a Set-based PSO (S-PSO) that can be used for solving Combinatorial Optimisation Problems (COPs) in the discrete space and demonstrate that this approach can efficiently navigate within discrete solution space and successfully solve COPs.

S-PSO employs a set-based representation of particles where particle position is depicted as a crisp set, whereas the velocity is a set with possibilities. In [22], COP is formulated as "finding a set of candidate solutions X which is a subset of the universal set of elements E, such that X satisfies some pre-defined constraints Ω and optimises the objective function f".

For a universal set E divided into N dimensions, velocity and position updating functions for nth dimension of kth particle are defined as,

$$V_k^n = \omega V_k^n + c_1.r_1^n.(pbest_k^n - X_k^n) + c_2.r_2^n.(gbest_k^n - X_k^n) \quad (1)$$

$$X_k^n = X_k^n + V_k^n \quad (2)$$

where, ω is the inertia weight that controls the momentum of the particle, c_1 and c_2 are learning factors related to particle's own experience and swarm's experience respectively and r_1, r_2 are random values in the range [0,1]. Eq. (1) depicts the velocity updating rule proposed by the original PSO algorithm. This tends to get trapped in local optimum solutions, specially when applied for discrete optimisation problems. Thus, [23], shows that for their proposed S-PSO for discrete space, Comprehensive Learning Particle Swarm Optimisation(CLPSO) algorithm [24], which is a variant of PSO, gives better performance.

CLPSO uses the following equation for velocity updating:

$$V_k^n = \omega V_k^n + c.r_1^n.(pbest_{f_k(n)}^n - X_k^n) \quad (3)$$

where $f_k(n)$ depicts the particle whose *pbest* is used by kth particle for updating its nth dimension. In this approach, instead of using *gbest* of the swarm, *pbest* of any particle including its own can be used. CLPSO uses tournament selection to select $f_k(n)$ depending on a probability (P_c) known as learning probability or uses its own *pbest*. To ensure that particles do not move towards poor directions, these exemplars are updated after a certain number of iterations (refreshing gap m), if the particle fitness fails to improve.

For the calculation of the new velocity, the following set-based operations are defined,

- Coefficient $\times$ Set with possibilities
  For a coefficient $c \geqslant 0$ and a set with possibilities defined on universal set E, depicted as $V = \{e/p(e)|e \in E\}$, product of the two is a set of possibilities $cV = \{e/p'(e)|e \in E\}$ calculated as,

$$p'(e) = \begin{cases} 1 & \text{if c} \times \text{p(e)} > 1 \\ c \times p(e) & \text{otherwise} \end{cases} \quad (4)$$

- Crisp Set - Crisp Set
  For two crisp sets X_1 and X_2, minus operator between the two $(X_1 - X_2)$ is defined as the crisp set of elements that are available in X_1, but not in X_2.
- Coefficient $\times$ Crisp Set
  For a coefficient $c \geqslant 0$ and a crisp set $X \in E$, product of the two results in a set of possibilities, $cX = \{e/p'(e)|e \in E\}$ calculated as,

$$p'(e) = \begin{cases} 1 & \text{if e} \in \text{X and c} > 1 \\ c & \text{if e} \in \text{X and c} \leq 1 \\ 0 & \text{if e} \notin \text{X} \end{cases} \quad (5)$$

- Set with Possibilities + Set with Possibilities
  Plus operator between two sets with possibilities generates a set with possibilities containing larger possibility for each element.

Updated velocity is used to adjust the position. Since solutions in discrete space should meet a pre-defined set of constraints, feasible positions are obtained using two main strategies; step-by-step construction, build and repair [23]. ω is used to achieve exploitation and exploration to overcome local optimums and move towards the global optimum of the problem. As larger values of ω supports global search whereas local search is supported by small ω values, changing ω from larger values to smaller values

through iterations enables the algorithm to converge into the global optimum value. The most common method of achieving this is by linearly changing ω over the iterations. But, [9] presents a non-linear function for varying ω that results in improved convergence.

Set-based CLPSO (S-CLPSO) is a successful method for solving COPs in discrete space with a higher convergence rate and simple implementation. In our work, we base our placement policy on S-CLPSO, integrate a lexicographic fitness function and further improve its performance by integrating multiple heuristics and propose a prioritised particle construction method to handle fog resource constraints to mitigate the issue of converging to a local optimum solution.

3. System model and architecture

This section details the microservices-based application model, fog architecture along with the pricing model used in this work.

3.1. Application model

To support the rapid modifications and agile development of IoT applications, microservices architecture is used to design and develop these applications. As microservices are designed as independently deployable modules adhering to a single business capability, the number of components that builds a single application increases. Due to the higher level of granularity presented by microservices architecture, a single service can consist of multiple microservices that collaborate to complete end-user requests. Moreover, a single microservice can be used by multiple services as well. So, higher flexibility and agility can be achieved by defining QoS parameters at these composite service level, instead of at the microservice or application levels.

Fig. 2.a shows the general representation of a microservices-based IoT application. Application is depicted using a Directed Acyclic Graph (DAG) where vertices represent microservices and edges denote the data dependencies among microservices. The starting point of the arrow indicates the client microservice and the arrowhead indicates the microservice invoked by the client microservice. Each application consists of a front-end denoted as *Client Module* that is always placed within client devices such as mobile phones, tablets, laptops that connect directly with IoT devices. The rest of the application consists of microservices that are placed either on fog or cloud resources based on the placement policy. Each application, $a \in A$ can be depicted as a tuple containing a set of microservices, data flows among them and a set of services where microservices collaborate to perform a function useful to the end-user; $\langle M_a, df^a, S_a \rangle$. Each microservice is characterised by its resource requirements; $\langle \Phi_m, \Omega_m, \Gamma_m, r_m \rangle$ indicating CPU, RAM and storage requirement of microservice $m \in M_a$ to support the request rate of r_m. This resource definition is used as the basic deployment unit of the microservice container and it is scaled horizontally or vertically based on the expected rate of the requests received by the microservice.

Based on the collaboration patterns among microservices, we have identified 3 types of service representations; *Chained*, *Aggregator* and *Hybrid* representation (Fig. 2.b). In a chained pattern, data flow within the service can be represented as a single chain whereas in an aggregator pattern, multiple data paths are invoked and the aggregator microservice waits for the processed data from those paths and aggregates them to return a single response. Aggregator microservice can invoke chains of microservices as well, which results in a hybrid representation. Thus, the completion time of each service differs based on the collaboration pattern of the microservices in the service. Microservices use the Asynchronous Request–Reply pattern, so once a request is made client microservice proceed to process other incoming requests until the response arrives. Each service $s \in S_a$ is denoted by a tuple containing a set of microservices creating the service and all possible data paths of the service; $\langle M_a^s, P_a^s \rangle$. The number of data paths in each service depends on the collaboration pattern of the microservices in the service.

The QoS profile of each application consists of QoS parameters that are defined separately per each service within the application. Our work considers makespan: end-to-end completion time of the service, budget: the amount the user expects to pay for the service and throughput: supported request rate by the service, as QoS requirements.

3.2. Fog architecture

Fog computing environment is a multi hierarchical environment consisting of IoT/client devices, fog layer and cloud layer. The fog layer is an intermediate layer that resides between the IoT and cloud layer, thus providing computational, networking and storage capabilities closer to the edge of the network. Fig. 3 depicts the architecture followed in this work. The fog layer consists of clusters of fog nodes deployed by multiple service providers. IoT sensors and actuators that connect to client devices (i.e., mobile phone, tablets) access fog clusters through *Fog Gateway Devices* (i.e., wireless access points, base transmission systems) and further connection to the cloud is maintained through a *Fog Cloud Gateway* node. We refer to this node as the *Fog Orchestration Node (FON)*, as it is responsible for monitoring fog nodes in the cluster and scheduling applications within them.

The fog layer consists of heterogeneous devices in terms of resource availability and access technologies. Each fog device ($f \in F$) is characterised by its resources in terms of CPU (ϕ_f), RAM (ω_f) and storage (γ_f). Fog nodes within the same cluster communicate with each other using Local Area Network (LAN) links which have considerably high bandwidths when compared with the Wide Area Network (WAN) links that connect fog clusters to the cloud. Multiple IoT and client devices use Wireless Area Network (WLAN) to connect to *Fog Gateway Devices*.

3.3. Pricing model

Due to distributed, scalable and independently deployable nature of the microservices, container technology has become the best-suited method of packaging and deploying microservice applications. Cloud service providers have server-less compute engines to support easy container deployment, by relieving the users of the responsibility to provision and manage servers. Such server-less platforms provide flexible pricing where users pay only for the amount of resources used by the containers. AWS Fargate [25] and Azure Container Instances [26] determine the pricing based on requested vCPUs, memory and storage amount where all three can be configured independently. In our work, we use the on-demand pricing models used by the above server-less platforms where the price of each fog/cloud device is defined as the total price for vCPUs, memory and storage.

4. QoS-aware application placement

We formulate the microservices-based application placement in fog environments as a *"Lexicographic Multi-objective Combinatorial Optimisation Problem"*, which aims at minimising QoS violation of services and ensures optimum use of fog and cloud resources while adhering to the resource requirements of the microservices. The proposed policy explores batch placement of services and also incorporate independently scalable nature of the microservices to obtain a more efficient placement. Relevant notations used in system model and problem formulation are represented in Table 2.

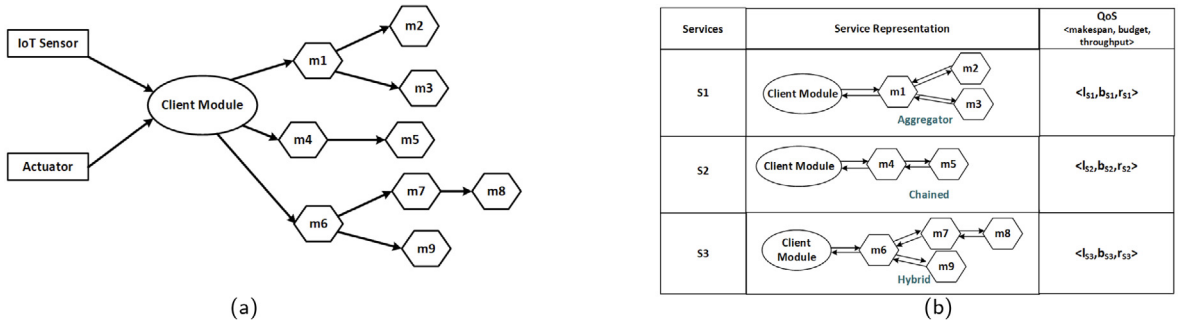

Fig. 2. Microservices-based IoT Application Architecture (a) DAG representation, (b) Service composition patterns.

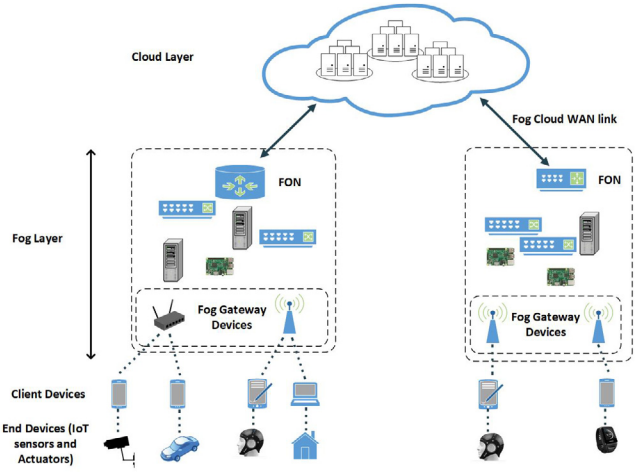

Fig. 3. An overview of the fog architecture.

Table 2
Notations.

Symbol	Definition
F	Set of all devices available in Fog layer.
C	Cloud.
E	Set of all client devices that connects to Fog Gateways.
D	All available devices. ($F \cup C \cup E$)
A	Set of all requested applications for placement.
M_a	Set of all microservices of application $a \in A$.
S_a	Set of all services defined for application $a \in A$.
M_a^s	Set of microservices of service $s \in S_a$.
P_a^s	Set of data paths in service $s \in S_a$.
df_p^s	Set of all data flows in path $p \in P_a^s$.
$\Delta_{mm'}$	Size of data transmitted from m to m'.
$i_{mm'}$	Number of instructions to process data sent from m to m'
$R_{mm'}$	Access rate among microservices m&m'.
d_{ij}^p	Network propagation delay among device $i, j \in D$
l_s	makespan requirement of service $s \in S_a$.
b_s	Budget requirement of service $s \in S_a$.
r_s	Throughput requirement of service $s \in S_a$.
ϕ_d	Processing capacity of device $d \in D$
ω_d	RAM of device $d \in D$.
γ_d	Storage capacity of device $d \in D$.
Φ_m	Processing capacity required by microservice $m \in a$.
Ω_m	RAM required by microservice $m \in a$.
Γ_m	Storage capacity required by microservice $m \in a$.
v_s^l	makespan violation of service $s \in S_a$.
v_s^b	Budget violation of service $s \in S_a$.
v^l	Total makespan violation.
v^b	Total budget violation.
$\eta(v^l)$	Normalised makespan.
$\eta(v^b)$	Normalised budget.
τ_{nw}	Total network usage due to placement.
τ_r	Total active devices due to placement.
Y	Set of devices $Y \subset D$, that are not eligible for placement of any microservice
$x_{m_i}^d \in \{0,1\}$	Equals to 1 if ith instance of microservice m is mapped to $d \in D$, 0 otherwise.
$act_f \in \{0,1\}$	Equals to 1 if at least one microservice is placed on $f \in F$, 0 otherwise.

4.1. Problem formulation

The fog application placement problem needs to focus on both application and fog resource heterogeneity. Thus, we formulate the placement problem to support the placement of a batch of applications (A) onto a set of devices (D) within the fog environment. As microservices are independently scalable, multiple instances of each microservice can be deployed. For each microservice m, number of instances is denoted by ins_m where m_i represents the ith instance. Resultant placement is expressed by $x_{m_i}^d$ where $x_{m_i}^d = 1$ depicts that the ith instance of microservice m is mapped to $d \in D$.

The main goal of the placement is to achieve maximum possible QoS satisfaction considering makespan, budget and throughput requirements. The throughput requirement of each service is satisfied by scaling the microservices. Throughput aware instance calculation for the microservices is discussed in detail in Section 4.1.1. Makespan and budget satisfaction are achieved by the first objective of the optimisation problem (Eq. (6)). Due to resource limits in the fog layer, it is not guaranteed that makespan and budget requirements of all services can be satisfied. Thus, we formulate Eq. (6) to minimise the weighted sum of normalised makespan violation ($\eta(v^l)$) and budget violation ($\eta(v^b)$) so that services with stringent QoS requirements are prioritised.

The purpose of the fog layer is twofold: to support latency-critical services and to reduce network usage by supporting bandwidth-hungry services. While the first objective handles latency-critical services, the second objective is introduced to move bandwidth-hungry microservices towards the edge of the network. Due to the resource-constrained nature of fog devices, it is important to strike a balance between fog and cloud resource usage to avoid over-use of limited fog resources. So we formulate the second objective (Eq. (7)) to achieve a trade-off between total fog resource usage (τ_r) and total network resource usage (τ_{nw}).

As QoS satisfaction is the primary objective of the placement, the second objective can be considered as a further improvement on the schedule proposed by the first objective. To ensure this, the placement problem is solved as a lexicographic optimisation where Eq. (6) is the primary objective and Eq. (7) is the secondary objective.

$$\text{minimise } [\omega_l\ \eta(v^l) + \omega_b\ \eta(v^b)] \tag{6}$$

$$\text{minimise } [\omega_{nw}\ \eta(\tau_{nw}) + \omega_r\ \eta(\tau_r)] \tag{7}$$

subject to,

$$\sum_{\forall d \in D} x_{m_i}^d = 1; \forall i \in [1, ins_m], \forall m \in M_a, \forall a \in A \tag{8}$$

$$\sum_{\forall d \in Y} \sum_{i=1}^{ins_m} x_{m_i}^d = 0; \forall m \in M_a, \forall a \in A \tag{9}$$

$$\sum_{\substack{\forall a \in A \\ \forall m \in M_a}} \sum_{i=1}^{ins_m} x_{m_i}^d \Phi_m \leq \phi_d; \forall d \in D \tag{10}$$

$$\sum_{\substack{\forall a \in A \\ \forall m_a \in M_a}} \sum_{i=1}^{ins_m} x_{m_i}^d \Omega_m \leq \omega_d; \forall d \in D \tag{11}$$

$$\sum_{\substack{\forall a \in A \\ \forall m \in M_a}} \sum_{i=1}^{ins_m} x_{m_i}^d \Gamma_m \leq \gamma_d; \forall d \in D \tag{12}$$

Both objectives are optimised under multiple constraints; placement constraints where each microservice instance is mapped to a single device (Eq. (8)) and each microservice is mapped only to eligible devices (Eq. (9)), resource constraints of all fog devices in terms of CPU, RAM and storage (Eqs. (10), (11), (12) respectively).

4.1.1. Throughput aware instance count calculation

In our proposed system model, resource requirements of each microservice are defined to support a certain request rate (r_m). We take this as the base instance and scale each microservice vertically or horizontally according to the expected service throughput (r_s per service s) in the application's QoS profile. Using the DAG representation of the microservice application, the expected request rate of each microservice (r'_m) can be calculated using the following equations:

$$r'_m = \sum_{\forall m' \in CM(m)} R_{m'm} \tag{13}$$

$$R_{m'm} = \begin{cases} r_s & m' \text{ is Client Module} \\ \alpha . r'_{m'} & \text{otherwise} \end{cases} \tag{14}$$

Function $CM(m)$ calculates all the client microservices of microservice m, that sends requests towards m. Thereby, Eq. (13) calculates access rate of the microservice m by taking summation of the request rates of all incoming edges of m. $\alpha \in [0, 1]$ represents the relationship between incoming and outgoing request rates of m'.

Accordingly, instance count for the microservice m is calculated as:

$$ins_m = \frac{r'_m}{r_m} \tag{15}$$

4.1.2. Primary objective - QoS violation

The first objective of the multi-objective fog placement is depicted in Eq. (6). Here the aim is to minimise the total violation of QoS in terms of makespan and budget requirements for a batch of services. Since QoS parameters are of different units, normalised values of each QoS parameter violation are used. The weighted sum of normalised sub-objectives forms the objective function. Weights are chosen to prioritise among the two parameters, maintaining $\omega_l + \omega_b = 1$.

1. Makespan violation -

Calculation of the makespan of a service depends on the data flow pattern of the service. For a service representing chained microservices, makespan can be calculated as the total processing time of each microservice and the data communication delay among microservices. But for aggregator and hybrid service patterns, aggregator microservice cannot complete the processing until results from all the data paths invoked by aggregator microservice are completed. Thus, the makespan of such services depends on the data path that takes the longest to complete.

Accordingly, makespan violation of service $s \in S_a$ where S_a indicates the set of services of application $a \in A$, can be calculated as,

$$v_s^l = max\{L(df_p^s); \forall p \in P_a^s\} - l_s \tag{16}$$

Eq. (16) calculates the difference between makespan defined in the QoS profile of the service (l_s) and makespan due to proposed placement. $L(df_p^{sa})$ is the function used to calculate the makespan of the datapath p of service s due to proposed placement. Data path with maximum makespan is equal to the makespan of the service irrespective of the data dependency pattern of the service.

This calculation considers both processing latency and network latency. Network latency includes transmission latency as well as propagation latency among different fog/ cloud nodes where microservices are placed. Since each microservice m has ins_m instances, we consider that for the dataflow among m and m', requests generated from ins_m are equally load balanced among ins'_m microservices. So, Eq. (17) aims to find the highest latency of the path considering all instances of a microservice.

$$L(df_p^s) = L_{nw}(df_p^s) + L_{proc}(df_p^s) \tag{17}$$

$$L_{nw}(df_p^s) = \sum_{\forall mm' \in df_p^s} max\left[\sum_{\forall dd' \in D} x_{m_i}^d x_{m'_j}^{d'} (d_{dd'}^p + L_{tr}); \forall i, j\right] \tag{18}$$

where $1 \leq i \leq ins_m$ and $1 \leq j \leq ins_{m'}$.

$$L_{tr}(d, d') = \Delta_{mm'}\left[\frac{\rho}{bw_{WLAN}} + \frac{\sigma}{bw_{LAN}} + \frac{\psi}{bw_{WAN}}\right] \tag{19}$$

ρ, σ and ψ contains binary values (0 or 1) depending on the d and d' device types.

$$\rho = \begin{cases} 1 & if\ (d \in E) \oplus (d' \in E) \\ 0 & otherwise \end{cases} \tag{20}$$

$$\sigma = \begin{cases} 1 & if\ (d \in F) \wedge (d' \in F) \wedge d \neq d' \\ 0 & otherwise \end{cases} \tag{21}$$

$$\psi = \begin{cases} 1 & if\ (d \in C) \oplus (d' \in C) \\ 0 & otherwise \end{cases} \tag{22}$$

$$L_{proc}(df_p^s) = \sum_{\forall (m,m') \in df_p^s} \frac{i_{mm'}}{\Phi_{m'}} \tag{23}$$

Total makespan violation of the placement is calculated taking the sum of violations as follows,

$$v^l = \sum_{\forall a \in A} \sum_{\forall s \in S_a} max\left[v_s^l, 0\right] \tag{24}$$

2. Budget violation -

Budget violation of service $s \in S_a$, where S_a indicates the set of services of application $a \in A$, can be calculated as,

$$v_s^b = \left(\sum_{\substack{\forall m \in M_a^s \\ \forall d \in D}} \sum_{i=0}^{ins_m} x_{m_i}^d C_m^d\right) - b_s \tag{25}$$

Cost of executing microservice m on device d, C_m^d is calculated based on the pricing model presented in Section 3.3. Total budget violation of the placement is calculated taking the sum of violations as follows,

$$v^b = \sum_{\forall a \in A} \sum_{\forall s \in S_a} max\left[v_s^b, 0\right] \tag{26}$$

4.1.3. Secondary objective - Resource utilisation

The second objective function (Eq. (7)) handles resource utilisation under two sub-objectives, computation resources and network resources. Similar to the primary objective, this also calculates the weighted sum of the two normalised sub-objectives.

1. Computation resource usage:

The fog layer consists of resource-constrained devices that are heterogeneous in their resource capacities. Thus within fog environments, it is important to place applications in such a way that limited computation power is utilised by using a minimum number of fog nodes so that only the services with stringent QoS requirements use limited fog resources. This provides fog service providers with the ability to host more applications within their fog infrastructure. Besides, this encourages a balance between horizontal and vertical scaling, thus reducing the carbon footprint as well.

$$\tau_r = \sum_{\forall f \in F} act_f \tag{27}$$

2. Network resource usage:

Fog resources can be used to reduce the amount of data sent towards cloud data centres by hosting bandwidth-hungry microservices. To this end, our placement policy introduces this sub-objective to reduce network usage, thereby increasing the placement of bandwidth-hungry microservices within the fog layer.

$$\tau^a_{nw} = \sum_{\substack{\forall mm' \in df^a \\ \forall dd' \in D}} \sum_{i=1}^{ins_m} \sum_{j=1}^{ins_{m'}} x^d_{m_i} x^{d'}_{m'_j} \frac{d^p_{dd'}\ \Delta_{mm'}\ R_{mm'}}{ins_m\ ins_{m'}} \tag{28}$$

$$\tau_{nw} = \sum_{\forall a \in A} \tau^a_{nw} \tag{29}$$

4.2. QoS-aware multi-objective S-CLPSO (QMPSO)

To solve the fog application placement problem, we propose a placement policy based on the S-CLPSO algorithm described in Section 2.2 and integrate multiple heuristics to improve the convergence rate by proposing novel approaches for multi-objective normalisation and particle construction. Algorithm 1 presents the overview of our proposed **QoS-aware Multi-Objective S-CLPSO (QMPSO)** placement policy.

Algorithm 1 QMPSO Algorithm

Input: Placement Requests and Meta-data
Output: Microservices to devices mapping

1: Calculate the number of instances per microservice
2: Initialise Min/Max sub-objective values using heuristics
3: Set iteration count $i \leftarrow 1$
4: Initialise population of N particles using **SWARM_INIT**
5: **while** $i \leq$ *Iterations* **do**
6: Calculate fitness values of all sub-objectives for each particle
7: Update Min/Max of sub-objectives
8: Obtain the normalised fitness values for each sub-objective
9: Calculate fitness values of the two main objectives for each particle using normalised values
10: Update *pBest* position of each particle
11: Update *gBest* position of the swarm
12: Select exemplar dimensions for each particle
13: Update velocity of each particle
14: Update position for each particle using **CPPC_VA**
15: Set $i \leftarrow i + 1$
return *gBest* of the swarm

QMPSO algorithm first derives the number of instances per each microservice (line 1) using Eq. (15), which calculates the instance count based on the throughput requirement of each service (r_s). Then the algorithm initialises the minimum and maximum possible values for each sub-objective of the multi-objective

Position Vector

Microservice	$m1_1$	$m1_2$	$m2_1$	$m2_2$	$m3_1$	$m4_1$
Device	d1	d4	d1	d3	d2	d4

Velocity Matrix

	$m1_1$	$m1_2$	$m2_1$	$m2_2$	$m3_1$	$m4_1$
d1	0.5	0.57	0.8	0.12	0.42	0.63
d2	0.25	0.4	0.73	0.44	0.91	0.5
d3	0.32	0.1	0.43	0.46	0.77	0.04
d4	0.12	0.74	0.68	0.33	0.8	0.65

Fig. 4. QMPSO particle representation.

fitness function formulated in Section 4.1 (line 2). QMPSO uses multiple heuristics to obtain estimates for these values. In our multi-objective optimisation problem, these values are used to calculate the normalised sub-objectives. Then an initial population is created using both heuristic-based and random placements (line 4), which is explained in detail in the Algorithm 2. After creating the initial population, the algorithm calculates fitness values for each sub-objective for all particles (line 6) and accordingly update the minimum and maximum values of each sub-objective (line 7). This enables the normalisation calculations to become more accurate as the swarm progresses through solution space in each iteration. Afterwards, normalised fitness values are calculated for each particle (line 8). Based on the values obtained for the two main objectives (line 9), the personal best (*pBest*) of each particle and the global best (*gBest*) of the swarm are updated using lexicographic comparison (lines 10–11). Then, the velocity matrix is updated using exemplar dimensions according to the S-CLPSO algorithm (lines 12–13). Finally, the new position of each particle is updated using the velocity matrix of the particle (line 14). Each created particle should adhere to the resource constraints of the fog devices. To satisfy this constraint, QMPSO proposes a novel particle construction process as **Constraint-aware Prioritised Particle Construction (CPPC)**. This contains two algorithms; **CPPC_INIT** for random construction of particles during swarm initialisation and **CPPC_VA** for velocity aware construction of particles after updating the velocity of the particle during each iteration. After executing these steps for a pre-defined number of iterations, the algorithm returns the global best position of the swarm as the placement mapping.

Integral steps of the QMPSO algorithm, which includes problem mapping, initial swarm, fitness calculation and particle position update are described in detail in the following sub-sections.

4.2.1. Mapping microservice application placement problem to S-CLPSO

As per the S-CLPSO, each particle representing a possible solution to the problem is characterised by a position vector and velocity matrix (Fig. 4). For the considered microservice placement problem, the position vector is a crisp set that maps each microservice instance to a device. Number of microservice instances are calculated at the start of the Algorithm 1 according to Eq. (15). The dimension of the position vector is equal to the total number of microservice instances that are to be placed. Velocity matrix is a set of possibilities that contains the possibility of each microservice instance being placed on each device. All position and velocity related basic calculations follow the concepts introduced in Section 2.2.

4.2.2. Initial swarm

Initial position vectors of the particles are generated using two methods: heuristic-based particle generation and random particle construction (Algorithm 2). We propose a novel makespan and

Algorithm 2 SWARM_INIT

Input: *numParticles*
Output: *Swarm*
1: *Swarm* ← {}
2: *pv* ← create position vector from **OHPP** placement
3: *vm* ← initialise velocity matrix
4: *particle* ← *createParticle*(*pv*, *vm*)
5: *Swarm.add*(*particle*)
6: *particleCount* ← 1
7: **while** *particleCount* < *numParticles* **do**
8: *pv* ← random construct using **CPPC_INIT**
9: *vm* ← initialise velocity matrix
10: *particle* ← *createParticle*(*pv*, *vm*)
11: *Swarm.add*(*particle*)
12: *particleCount* ← *particleCount* + 1
return *Swarm*

budget aware heuristic named, **Osmotic Heuristic Placement Policy (OHPP)** to seed the initial population (lines 2–4). The rest of the particles are generated using the random particle construction path of the CPPC process (lines 7–12).

OHPP: For the swarm to reach global optimum position faster, we introduce **Osmotic Heuristic Placement Policy (OHPP)**. This policy follows the concept of Osmotic computing and tries to move latency-sensitive services to the fog layer (Algorithm 3). The algorithm starts by placing all microservices on the cloud and afterwards calculates the latency violation of each service using Eq. (16) (line 1–2). Makespan violated services are sorted from minimum to maximum budget requirement (line 3–4) and matched on to fog devices sorted from minimum to maximum pricing (line 5–22). Other than providing a feasible placement, OHPP is also used to prioritise microservices for placement within the fog layer (lines 23–24). OHPP algorithm derives all microservices of the *FogServices* and outputs them as prioritised microservices to be placed on fog (*ToFogM*), while the rest of the microservices are added to a separate list (*ToCloudM*). This prioritisation plays an important role in the particle construction process of the CPPC (in both CPPC_INIT and CPPC_VA).

Algorithm 3 OHPP

Input: Placement Requests and Meta-data
Output: Microservices to devices mapping
1: Place all microservice instances in Cloud
2: Calculate deadline violation using Eq. (16)
3: *FogServices* ← get all deadline violated services
4: *sorted* ← sort *FogServices* from min to max budget requirement
5: *FogDevices* ←sort from min to max pricing
6: **for** each service *s* in *sorted* **do**
7: M_{inst} ← topologically sorted μservice instances of *s* from meta-data
8: **for** each *m* in M_{inst} **do**
9: **if** *m.predecessor* in *Cloud* **then**
10: *d* ← *Cloud*
11: **else**
12: *d* ← *FogDevices.first*
13: **while** *m* is not *placed* **do**
14: **if** *d.availResources* ≥ *m.resources* **then**
15: place *m* in *d*
16: update *availResources* of *d*
17: **else**
18: **if** *d* = *FogDevices.last* **then**
19: *d* ← *Cloud*
20: **else**
21: *d* ← *FogDevices.next*
22: Place the rest on Cloud
23: *ToFogM* ← microservices of *FogServices*
24: *ToCloudM* ← microservices not included in *FogServices*
25: **return** microservice placement, *ToFogM*, *ToCloudM*

CPPC_INIT: For the creation of the rest of the particles, the random construction path of the CPPC Algorithm is used (Algorithm 4). CPPA_INIT uses microservice prioritisation for the construction of placements under resource constraints. The algorithm prioritises latency-critical microservices in *ToFogM* for the placement within resource-constrained fog devices (lines 2–9). Afterwards, the algorithm proceeds with mapping *ToCloudM* microservices (line 10–16).

The above methods together populate the initial swarm with diverse and feasible solutions, thus improving the ability of the swarm to reach its global optimum solution within less amount of time. For the initialisation of the velocity matrix, a value in the range [0,1] is assigned for the mapped device of each microservice instance and the rest of the devices are assigned 0 for the said microservice.

Algorithm 4 CPPC_INIT Algorithm

Input: *D* devices, *ToFogM*, *ToCloudM*
Output: *PositionVector*
1: *PositionVector* ← {};
2: *devices* ← *D.getFogDevices*();
3: *devices.add*(*D.getCloudDevices*());
4: **for** each microservice *m* in *ToFogM* **do**
5: **for** each device *d* in *devices* **do**
6: **if** *d.availResources* ≥ *m.resources* **then**
7: *PositionVector.add*(*m*, *d*)
8: update *availResources* of *d*
9: break;
10: *devices.shuffle*();
11: **for** each microservice *m* in *ToCloudM* **do**
12: **for** each device *d* in *devices* **do**
13: **if** *d.availResources* ≥ *m.resources* **then**
14: *PositionVector.add*(*m*, *d*)
15: update *availResources* of *d*
16: break;
return *PositionVector*

4.2.3. Normalised fitness calculation

The placement problem is modelled with two main objectives where each is calculated as the weighted sum of its two sub-objectives (Section 4.1). As each sub-objective value has different units and has different ranges of values, a normalised weighted sum is required to reach a proper trade-off between the sub-objectives. The best approach for this would be to minimise and maximise each sub-objective separately to obtain the range of values for each. But due to the higher time consumption of this method, we propose a heuristic driven normalisation approach to initialise minimums and maximums for each sub-objective.

We use the following heuristics to initialise maximums and minimums for each sub-objective with close enough estimates:

Deadline-aware heuristic placement: Services are sorted from minimum to maximum makespan requirement and placed starting from fog layer and move to cloud-only if non of the fog devices have enough resources to host the microservice. This placement provides an estimate for minimum latency violation and minimum network usage.

Budget maximisation placement: To find an estimate for maximum cost violation, we propose a heuristic where devices are sorted from maximum to minimum unit price, services are sorted from minimum budget to maximum budget requirement and microservices from ordered services are matched with ordered devices.

Cloud only placement: All microservices are placed in the cloud providing an estimate for maximum latency violation, maximum network usage and minimum budget violation.

Moreover, for fog resource usage, the minimum is set to 0 and maximum is determined as *min(fog device count, microservice instance count)*.

During each iteration of the QMPSO algorithm, minimum and maximum values are updated based on the fitness values of

the particles in the swarm. Updated minimum and maximum values are used to calculate Min–Max Normalisation, which scales the value of each objective to the range 0–1. For an objective i (obj_i), with current value of x, normalised value using Min–Max Normalisation is calculated as follows:

$$\eta(x) = \frac{x - min(obj_i)}{max(obj_i) - min(obj_i)} \tag{30}$$

This approach enables the QMPSO algorithm to avoid premature convergence and reach a fair trade-off between the weighted sub-objectives.

4.2.4. Position update - Constraint-aware prioritised particle construction (CPPC)

Position updating undergoes three main steps; (1) selection of exemplar dimensions for the particle (2) update particle velocity (3) construct new particle position.

Selection of exemplar dimensions is the process of selecting which particle's *pBest* should be followed by each dimension for velocity updating as depicted in Eq. (3). Particle position update is carried out using the updated velocity matrix. Due to resource constraints of the fog resources, updated particle positions have to adhere to resource constraints, which is one of the main challenges of fog service placement, as mending particles based on resource constraints can considerably hinder the convergence to the global optimum position.

To overcome this issue, we propose CPPC_VA, which is the velocity aware path of our proposed CPPC process. It is a particle construction algorithm that checks resource constraints at the time of particle position update based on the velocity matrix. CPPC_VA uses two main features to improve convergence (Algorithm 5):

Use of prioritised microservices: Similar to CPPC_INIT, this algorithm also uses microservice prioritisation into two groups as *toFogM* and *toCloudM*. The idea behind this is to enable *toFogM* microservices a higher chance of being assigned to the devices indicated by its velocity matrix (lines 1–2).

Velocity aware device selection: For each microservice, the algorithm tries to determine the new fog device in a velocity conscious manner. First, all devices with higher or equal velocity values are identified (lines 5–7) and each selected device is considered for placement until a device with the required resource amount is met (lines 8–12). If all selected devices are infeasible for placement, microservice is added to a separate list (*notMapped*) for placement later (lines 13–14). After iterating through all microservices, each microservice in *notMapped* list are mapped to feasible devices randomly (lines 15–20). Here all devices are considered irrespective of the velocity value. This method provides a proper balance between exploitation and exploration, thus generating a diverse solution set and improving algorithm convergence.

Algorithm 5 CPPC_VA Algorithm

Input: D devices, particle P, *ToFogM*, *ToCloudM*
Output: updated particle P

1: $microservices \leftarrow ToFogM$
2: $microservices.add(ToCloudM)$
3: $notMapped \leftarrow \{\}$
4: **for** each microservice m in $microservices$ **do**
5: $currDevice \leftarrow P.positionVector.get(m)$;
6: $currVelocity \leftarrow P.velocityMatrix.get(currDevice)$
7: $D' \leftarrow$ get devices with velocities $\geq currVelocity$
8: **for** each device d in D' **do**
9: **if** $d.availResources \geq m.resources$ **then**
10: $P.PositionVector.add(m, d)$
11: update $availResources$ of d
12: break;
13: **if** m is not mapped to a device **then**
14: $notMapped.add(m)$
15: **for** each microservice m in $notMapped$ **do**
16: $D' \leftarrow$ get all possible devices from D
17: $r = random(1, D'.size)$
18: $d \leftarrow D'.get(r)$;
19: $P.PositionVector.add(m, d)$
20: update $Avail\_Resources$
return updated particle P

5. Performance evaluation

We evaluated the performance of our QMPSO algorithm through simulation of synthetic workloads of microservices-based IoT applications, that have heterogeneous QoS requirements in terms of makespan, budget and throughput. Evaluations are completed under two main categories:

QMPSO Performance Evaluation: Section 5.3.1 evaluates the performance of the QMPSO algorithm in terms of convergence improvement against two adaptations of the S-CLPSO algorithm for fog application placement problem.

1. **No-Heuristics:** Directly adapts the S-CLPSO algorithm to fog application placement problem without incorporating any heuristics.
2. **No-Prioritised-Construct:** Heuristics are used in this approach for swarm initialisation and fitness normalisation. But particle construction does not prioritise microservices during the construction process, but randomly select microservices.

QMPSO Placement Evaluation: Section 5.3.2 compares QMPSO with four other fog application placement approaches in terms of QoS satisfaction and optimum fog-cloud resource usage. These approaches are selected to cover different types of algorithms including optimisation-based, meta-heuristic and heuristic approaches to solve application placement problem within fog environments.

1. **Constraint Programming based Placement Algorithm (CPPA)** - Placement problem introduced in Section 4.1 is solved using a Constraint Programming solver.
2. **OHPP** - Algorithm which is used in generating our initial swarm of particles. We use this to demonstrate how the incorporation of improved S-CLPSO results in better placement decisions.
3. **FSPP** - Fog service placement approach proposed in [15], where service spread (scale microservices to evenly spread them within fog environment), latency (minimise communication delay among microservices) and resource usage (maximise fog device usage) are the focus of placement decision making.
4. **DNCPSO** - Algorithm proposed in [9] for workload scheduling in cloud–edge environments to minimise the total latency and cost of the placement.

Out of the existing works analysed in Section 2.1, FSPP and DNCPSO are the only works that can be adapted and applied to the batch placement of microservices-based applications addressed in our work. So, they are chosen for the performance comparison.

5.1. Implementation of the algorithms

For the performance evaluation, all placement algorithms were implemented using iFogSim2 [14] simulator, which is a toolkit for the simulation of fog computing environments. iFogSim2 extends iFogSim [27] simulator and provides support for simulation of microservice application placement through its advanced features such as service discovery and load balancing.

Table 3
Simulation parameters.

	Parameter	Value
Communication links (latency, bandwidth) [30–33]	LAN	0.5 ms, 1 Gbps
	WAN	30 ms, 100 Mbps
	WLAN	2 ms, 150 Mbps
Fog device resources [34,35]	CPU (MIPS)	1500–3000
	RAM (GB)	2–8
	Storage (GB)	32–256
Cost model parameters [25]	CPU (Cloud)	$0.040480 per 150 MIPS per hour
	RAM (Cloud)	$0.004445 per GB per hour
	Storage (Cloud)	$0.000111 per GB per hour
	Increase factor for fog	1.2–1.5
QoS parameters [36,37]	Makespan (l_s)	20–150 ms
	Budget (b_s)	$0.25–1.50 per hour
	Throughput (r_s)	200–800 requests/s

5.1.1. QMPSO implementation

To support the simulation of the proposed system architecture along with the QMPSO algorithm, we extended the iFogSim2 simulator by integrating features to support multiple service composition patterns and QoS profiles containing per service QoS definitions. Afterwards, the QMPSO algorithm was developed and simulated on top of that.

5.1.2. CPPA implementation

An optimised solution for the placement problem can be obtained by solving the problem modelled in Section 4.1 using a solver. In this work, we have used the Constraint Programming(CP) engine of IBM ILOG CPLEX 12.10.0 solver [28] for obtaining the optimum solution for fog application placement. iFogSim2 is used for the implementation of the placement policy by using Java API available in the solver.

CPPA approach uses lexicographic optimisation with objectives ordered as, minimising QoS violation as first objective (Eq. (6)) and resource utilisation as second objective (Eq. (7)). Normalised values of each objective ($\eta(v^l)$, $\eta(v^b)$, $\eta(\tau_{nw})$, $\eta(\tau_r)$) is calculated by taking "Nadir point" as minimum value and "Utopia point" as maximum for each objective [29] . These two points are calculated by optimising each objective separately. Afterwards, the placement problem is solved using multi-objective optimisation. To obtain the results within a reasonable time limit, the failure limit parameter is set to 10^7 failures during the search process.

5.1.3. DNCPSO and FSPP

DNCPSO and FSPP were implemented and simulated in iFogSim2 based on the algorithms described in [9,15] respectively. Necessary adaptations were made to the algorithm to adapt it to our proposed system model and IoT application batch placement scenario while maintaining core principles and fitness functions as proposed in the said works.

5.2. Experimental configurations

5.2.1. Simulation environment

To evaluate the performance of the algorithms, we created synthetic workloads based on the microservices-based application model proposed in Section 3.1. Each workload consists of multiple applications, including Smart health monitoring and Smart Parking application presented in the motivation scenario along with synthetic DAG-based applications created to represent all service composition patterns introduced in this work. Heterogeneity within the workloads is further ensured by modelling the diversity of microservices in terms of computation cost of the microservices (300–900 MIPS) and bandwidth usage among microservices (200–1500 bytes/packet). Moreover, when defining resource requirements of each microservice, the request rate supported by the basic deployment unit (r_m) is chosen between 100–200 requests/s. All the above parameter values are determined based on the IoT simulation benchmarks presented in previous simulation studies [14,38]. Diversity among services is maintained in terms of QoS by varying makespan, budget, and throughput requirements.

The fog environment is constructed according to the architecture proposed in Section 3.2. Table 3 lists the configurations used in constructing the simulated fog environment. Parameters of the fog network such as communication link latency and bandwidth represent novel network technologies like Wi-Fi 6 [30], 5G [31] for WLAN, and gigabit Ethernet [32] for LAN connections, acquired from edge network performance studies. Fog resources are modelled as a pool of heterogeneous devices with varying resource capacities similar to [34,35] which include heterogeneous fog devices such as Raspberrypi 4B, Jetson Nano, Dell PowerEdge, etc. Cost of execution of the microservices is modelled according to AWS Fargate pricing [25] defined for CPU, RAM, and storage separately. Due to service level improvements provided by the fog environment, fog resource prices are determined by multiplying on-demand prices of cloud resources by an increase factor between 1.2–1.5 according to [7], which models on-demand pricing within fog environments. vCPU to MIPS mapping for the simulation is obtained based on Microsoft Azure industrial benchmark where 150MIPS estimates to 1vCPU [39].

QoS parameters are varied to ensure makespan and budget limits of the services span from the edge of the network to the cloud. Makespan requirement is varied within 20–150 ms, following the IoT application latency requirements discussed in the previous studies [36,37]. The budget requirement is set based on the resource requirements of each microservice in the synthetic workload and cost parameters of the environment in such a way that the values span both cloud and fog deployment. Moreover, the budget parameter is adjusted so that latency-critical and bandwidth-hungry services have higher budget limits to enable their placement within the fog layer. For throughput requirement of services (r_s), we have considered a wide range of values (200–800 requests/s) compared to r_m of each microservice, to evaluate how the placement algorithm handle the scalability of microservices.

5.2.2. Algorithm parameter tuning

Table 4 lists parameters and their values for QMPSO, DNCPSO and FSPP algorithms. For QMPSO algorithm preliminary experiments were carried out to observe the fitness value achieved by the algorithm for different values of swarm size, iteration count and refreshing gap. Based on the observations, we set particle count to 50, iteration count to 300 and refreshing gap to 0. Further improvements to the fitness values can be obtained by

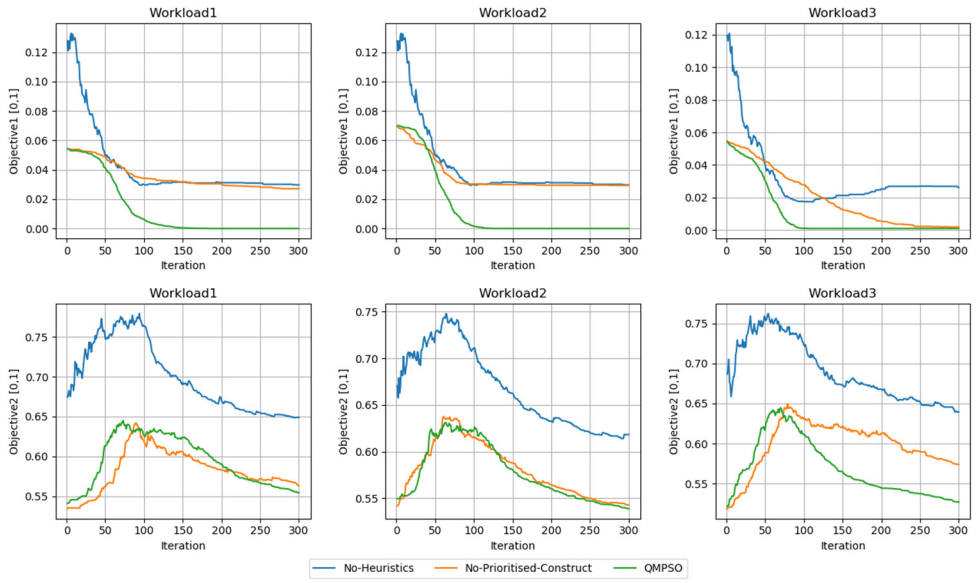

Fig. 5. Variation of fitness values for different adaptations of S-CLPSO.

Table 4
Parameters for placement algorithms.

Parameter	QMPSO	DNCPSO	FSPP
No. of particles in swarm	50	50	100
No. of iterations	300	300	400
Mutation rate	–	0.25	0.25
ω_{min} - ω_{max}	0.4–0.9	0.4–0.9	–
c_1, c_2	–	2	–
c	1.49445	–	–
m (refresh gap)	0	–	–
ω_l, ω_b, ω_{nw}, ω_r	0.5	–	–

increasing particle and iteration counts at the cost of increased algorithm execution time. Values for inertia weight ω and coefficient c are chosen based on previous studies on PSO algorithm [9, 24] conducted to determine the optimum values for these parameters. ω is changed from ω_{max} (0.9) to ω_{min} (0.4) over the iterations, according to the non-linear equation proposed in [9]. Coefficient c is set to 1.49445 as per [24]. For the performance evaluation, we consider all sub-objectives are equally important. Due to objective normalisation used in the QMPSO algorithm, this is achieved by setting all weights to 0.5.

For DNCPSO and FSPP algorithms, parameters are set according to [9,15] respectively.

5.3. Results and analysis

5.3.1. QMPSO performance evaluation

This section evaluates the performance of QMPSO by analysing how the values for the two main objectives gradually evolve with iterations. For the evaluation, three synthetic workloads are created according to the specifications detailed in Section 5.2.1, where Workload1, Workload2 and Workload3 consist of 5, 7 and 10, microservice-based applications respectively. For placement of the workloads, a fog environment with 17 fog devices is considered. For each workload, placement is generated using No-Heuristics, No-Prioritised-Construct and QMPSO algorithms and the fitness values for Objective1 (QoS violation) and Objective2 (Resource usage) are recorded over 300 iterations. Each algorithm is repeated 100 times and the average fitness values are obtained.

Fig. 5 depicts the variations of fitness values while Table 5 lists the average fitness value of each algorithm after 300 iterations. Results show that the QMPSO algorithm outperforms the other two approaches in reaching the global optimum solution within a lesser number of iterations. For both objectives, No-Heuristics demonstrates a higher fluctuation in fitness value during early iterations. In No-Prioritised-Construct and QMPSO algorithms, this behaviour is not present for Objective1 due to heuristics based minimum and maximum initialisation. No-Heuristics algorithm updates minimum and maximum values for each sub-objective only based on the particles available in the swarm. So it takes the algorithm a larger number of iterations to obtain accurate values, which results in the fluctuations. Besides, the use of OHPP in the initial swarm provides No-Prioritised-Construct and QMPSO with a better starting point. Moreover, both No-Heuristics and No-Prioritised-Construct tend to converge to local-optimum positions. QMPSO has overcome this with the proposed particle construction algorithm, CCPC. The use of prioritised microservices in CCPC ensures a proper balance between exploitation and exploration to make sure that the algorithm moves towards the global-optimum solution for Objective1. As solution space is a discrete space limited by resource constraints, there is a higher chance of algorithms converging to a local optimum solution. But prioritised particle construction in QMPSO helps the algorithm to traverse the discrete solution space successfully without getting stuck in local optimums.

As the placement problem is modelled as a lexicographic optimisation, fluctuations are expected to occur in the Objective2 value until Objective1 converges. This explains the increase in Objective2 during early iterations in all three approaches. In No-Heuristics, the increase and fluctuations in the value are considerably higher because it takes more time for this approach to obtain the accurate minimum and maximum values for the sub-objectives without the use of heuristics. Thus, faster convergence in Objective1 results in better results of Objective2 as well. This is evident in the behaviour of the QMPSO algorithm. Objective values denoted in Table 5 shows that QMPSO reaches lower objective values for both objectives. Besides, the standard error of the achieved values is also lower in QMPSO when compared with other approaches. This indicates that the performance of QMPSO is consistent over multiple runs. The above results demonstrate that the proposed QMPSO algorithm can reach better performance due to multiple features we have incorporated with the algorithm, including OHPP-based swarm initialisation (SWARM_INIT), heuristic-driven fitness value normalisation and prioritised particle construction (CCPC).

Table 5
Mean fitness values and standard error of the objectives for different adaptations of S-CLPSO to fog placement problem.

	QMPSO		No-Prioritised-Construct		No-Heuristics	
	Obj1	Obj2	Obj1	Obj2	Obj1	Obj2
Workload1	0	0.5544 ± 0.0025	0.0271 ± 0.0020	0.5633 ± 0.0068	0.0297 ± 0.0012	0.649 ± 0.0074
Workload2	0	0.5389 ± 0.0017	0.0293 ± 0.0007	0.5429 ± 0.0040	0.0262 ± 0.0013	0.6183 ± 0.0072
Workload3	0.0010 ± 6.145E−06	0.5271 ± 0.0018	0.0019 ± 0.0007	0.5738 ± 0.0049	0.005 ± 0.0015	0.6395 ± 0.0067

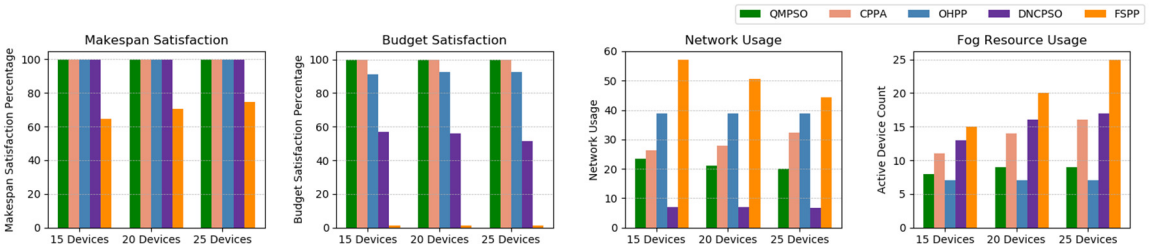

Fig. 6. Performance for different device counts.

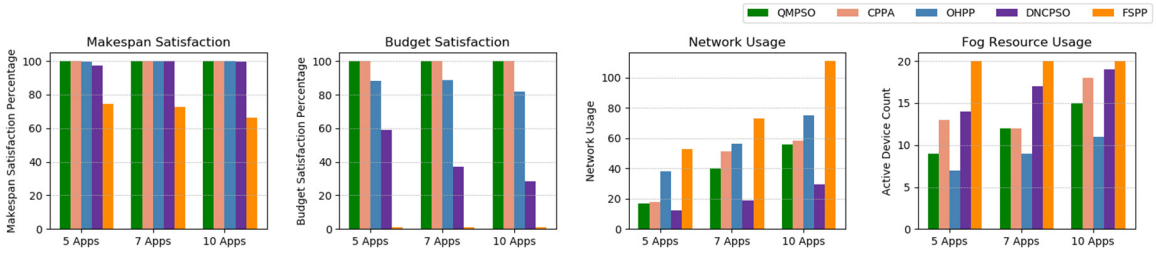

Fig. 7. Performance for different application/microservice counts.

5.3.2. QMPSO placement evaluation

In this section, we evaluate the placement generated by our algorithm using several performance metrics: makespan satisfaction percentage and budget satisfaction percentage are used to evaluate the QoS satisfaction of the placement, network usage and the total number of active fog devices to evaluate the fog resource usage.

Makespan Satisfaction: This metric is calculated as the number of service requests that meets makespan requirements of the said service, as a percentage of all the service requests received by the fog environment.

Budget Satisfaction: A metric reflecting budget satisfaction percentage of the fog environment. This metric is calculated as the difference between the cost violation after placement and the maximum possible cost violation of the environment for the requested placement, as a percentage of the maximum possible violation.

Network Usage: Indicates network occupancy as a measurement of *packet size (kilobytes) x link delay (ms)* within the duration of the simulation for all packets sent through the fog environment.

Active Fog Devices: Depicts the number of devices with at least one microservice deployed onto the device. Optimum usage of fog computing resources can be evaluated based on two main aspects; balanced use of fog and cloud where fog resources are used only for latency-critical and bandwidth-hungry services which mitigates overuse of limited fog resources, and the ability to avoid unnecessary dispersion of microservices within highly distributed fog environments. Active fog device count is a quantitative metric that can provide accurate insight on both of these aspects.

Solution Space Analysis: Experiments are conducted to observe the performance of each algorithm as the solution space grows. The size of the solution space depends on two parameters; the number of microservice instances to be placed and the number of devices considered for placement of the microservices. Fig. 6 depicts the performance for different device counts (15, 20 and 25 fog devices) keeping microservice count a constant (8 Applications, 30 microservices) whereas Fig. 7 is for the scenarios where the microservice count is changed keeping device count a constant (20 fog devices). Microservice count is increased by increasing the number of applications considered for placement (5, 7 and 10 applications). Moreover, it varies the degree of heterogeneity within the batch of services available for placement.

For QoS satisfaction, QMPSO and CPPA achieve the highest satisfaction percentage in both makespan and budget for all scenarios. But for network usage and fog resource usage which indicate the ability of the algorithm to obtain a proper balance between fog and cloud usage, QMPSO outperforms CPPA. As the solution space grows, network usage and active device count for CPPA placement increase. Due to the NP-complete nature of the fog application placement problem, CPPA is limited by a failure limit of 10^7 to obtain a solution within reasonable time limits. Thus, QMPSO with its meta-heuristic approach outperforms CPPA. Fig. 8 compares the execution time of QMPSO and CPPA algorithm as solution space grows. Both the execution time and increase in execution time with solution space growth is considerably higher in CPPA.

OHPP, which is our proposed heuristic for QMPSO initialisation can achieve high makespan satisfaction but lacks budget satisfaction. OHPP prioritises services based on stringent makespan

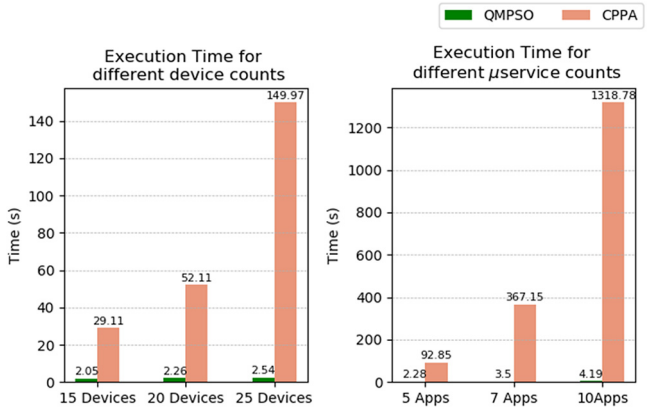

Fig. 8. Execution time of the QMPSO and CPPA algorithms.

and budget requirements but fails to handle the complexity introduced due to data dependencies among microservices. As a result, OHPP shows a decrease in budget satisfaction as the number of applications increases. Moreover, OHPP only focuses on moving latency-critical service to the fog and place the rest of the service in the cloud. As a result, bandwidth-hungry microservice of latency tolerant services are placed in the cloud, which under-utilises fog resources and increases network usage.

The fitness function of the DNCPSO algorithm is designed to minimise the weighted sum of total latency and cost of the placement. Moreover, DNCPSO aims to re-arrange particles to place all latency-critical microservices in the fog layer which results in high makespan satisfaction. However, the budget satisfaction of the algorithm drops significantly (up to 70%), as the fitness calculation does not contain budget awareness, but try to minimise the total cost. This approach lacks prioritisation of services with stringent budget requirements which moves more services to the fog to reduce total latency. This results in lower network usage, but over utilises fog resources and reduces budget satisfaction significantly.

Similarly, FSPP tries to minimise the latency of the services without taking their makespan requirements into consideration. As this approach does not prioritise latency-critical microservices, makespan satisfaction reduces up to 35% within a resource-constrained fog environment. As the number of devices increase, more latency-critical microservices are placed inside the fog layer, which results in a slight increase in makespan satisfaction. FSPP aims to increase the fog resource usage by placing replicas of the microservices without imposing a budget constraint, which results in closer to zero budget satisfaction. FSPP tries to maximise fog resource usage irrespective of the throughput requirements of the services. As a result, all fog devices are active in all placement scenarios. As FSPP scale microservices randomly in the fog layer, the number of microservices pushed to the cloud increases due to the resource constraints of fog devices. This results in a significant increase in network usage as well.

Scalability Analysis: Experiments are conducted to analyse the performance of the placement as throughput requirements of the services change. To this end, a workload of 5 applications is considered for two scenarios where throughput requirement in QoS profile of each application is doubled in scenario2 when compared with scenario1 (Fig. 9). For both scenarios, 25 fog devices are considered for placement.

As throughput requirement increases, microservices are horizontally scaled in QMPSO, CPPA and OHPP placement policies due to the throughput aware instance count calculation proposed in Section 4.1.1. This increases the number of microservice instances to be placed, thus expanding the solution space. As a result, the performance of the CCPA reduces with increased throughput (scenario2). In scenario1, CPPA is able to reach similar QoS satisfaction values as our QMPSO algorithm. But in scenario2, CPPA is unable to reach an optimum solution within the specified failure limit of the algorithm, which results in the reduction of makespan satisfaction. Although OHPP is able to achieve full makespan satisfaction, budget satisfaction drops significantly (up to 20% reduction) as throughput increases. Thus, as the number of microservice instances increase heuristic approach fails to provide satisfactory results. DNCPSO does not consider the scalability of microservices. So, as throughput increases, resource requirements of each microservice increase and resource-constrained fog devices are unable to handle them. As a result, DNCPSO moves these microservices towards the cloud, which results in the increase of latency violation and network usage. Without using horizontal scalability, DNCPSO is unable to fully utilise fog devices with limited resources. FSPP scales microservices to spread them evenly across the fog environment. So, FSPP does not demonstrate a significant difference in makespan satisfaction as sufficient microservice instances are available in both scenarios. But, FSPP randomly scales microservices without supporting throughput aware scalability which result in the overuse of fog resources.

Table 6
Complexity analysis.

	QMPSO	DNCPSO	FSPP
Initialisation	$\mathcal{O}(Slog(S) + \|D'\|log(\|D'\|) + S.M.I.\|D'\|)$	$\mathcal{O}(S.M.\|D'\|)$	$\mathcal{O}(S.M.\|D'\|)$
Evolution	$\mathcal{O}(S.M.I.\|D'\|)$	$\mathcal{O}(S.M.\|D'\|)$	$\mathcal{O}(S.M.\|D'\|)$

Compared to other approaches, QMPSO achieves improved performance in all considered metrics for both scenarios. Our placement vertically and horizontally scales microservices based on their throughput requirements, which results in proper utilisation of resource-constrained fog devices to maximise makespan and budget satisfaction. This also indicates the ability of the QMPSO algorithm to successfully navigate larger solution spaces, unlike CPPA and OHPP algorithms.

Based on the solution space analysis and scalability analysis, it is evident that QMPSO significantly improves QoS satisfaction along with resource utilisation. For the considered scenarios, QMPSO records up to 35% improvement in makespan satisfaction and up to 70% improvement in budget satisfaction. These results indicate the ability of the QMPSO algorithm to navigate large solution spaces successfully to reach optimum QoS satisfaction. Moreover, results depict that QoS-awareness in the fitness function of QMPSO enables it to successfully utilise both fog and cloud resources to handle heterogeneous QoS requirements. Thus, QMPSO provides a robust algorithm capable of harvesting fog and cloud resources to obtain an efficient placement schedule for heterogeneous microservice-based IoT applications.

5.3.3. Algorithm complexity analysis

We have introduced multiple approaches/algorithms to improve the performance of our QMPSO placement algorithm. In this section, we evaluate the time complexity introduced by these novel approaches and compare them with approaches used in DNCPSO and FSPP algorithms which use PSO and NSGA-II respectively. All three evolutionary algorithms have two main phases that affect the overall complexity of the algorithms and Table 6 presents their complexities. We consider the number of services for placement as S with each having a maximum of M microservices along with I instances per microservice for the placement within $|D'|$ devices where $D' = F \cup C$. The effect of population size

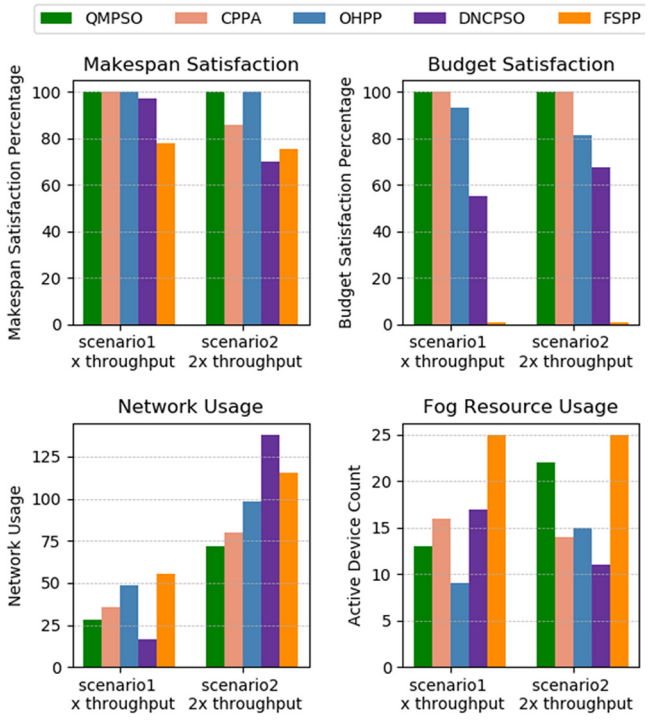

Fig. 9. Performance for different throughput requirements.

and iteration count is ignored as they are constants in all three algorithms.

Initialisation : For QMPSO, initialisation includes the creation of initial solution space (**SWARM_INIT**) and heuristic based initialisation of minimum and maximum values required for normalisation of the sub-objectives. SWARM_INIT consists of **OHPP** and **CPPC_INIT** algorithms. OHPP contains two main steps; sorting of services and devices which is completed with linearithmic time complexity of $\mathcal{O}(Slog(S))$ and $\mathcal{O}(|D'|log(|D'|))$ respectively, and mapping of each microservice instance of the service to a device ($\mathcal{O}(M.I.|D'|)$), which results in time complexity of $\mathcal{O}(S.M.I.|D'|)$ for all services. CPPC_INIT iterates through prioritised microservice instances to randomly find the eligible device. As prioritising is already completed by OHPP, the algorithm can be completed with worst case time complexity of $\mathcal{O}(S.M.I.|D'|)$. The total time complexity of SWARM_INIT is $\mathcal{O}(Slog(S) + |D'|log(|D'|) + S.M.I.|D'|)$. Heuristic approaches used for normalisation (Deadline-aware heuristic placement and Budget maximisation placement) follow the same placement approach as OHPP with different sorting orders for services and devices, thus resulting in polynomial time complexity of $\mathcal{O}(Slog(S)+ |D'|log(|D'|)+S.M.I.|D'|)$. Thus, for the Initialisation phase worst-case time complexity of the QMPSO resolves to $\mathcal{O}(Slog(S) + |D'|log(|D'|) + S.M.I.|D'|)$.

Both DNCPSO and FSPP, do not use heuristics when creating initial population nor use heuristic-based normalisation. Furthermore, these algorithms do not support throughput aware scalability of microservices. Thus, random initialisation of eligible solutions within resource-constrained devices results in time complexity of $\mathcal{O}(S.M.|D'|)$ for DNCPSO and FSPP.

Evolution: For this phase time complexity of the algorithm is dominated by the construction of the next solution. For QMPSO, this consists of velocity update and position update. The time complexity of velocity update is equal to the number of elements in the velocity matrix, which is $\mathcal{O}(S.M.I.|D'|)$. QMPSO uses CPPC_VA algorithm to update the particle positions. Similar to CPPC_INIT, CPPC_VA also acquires prioritised microservice instances generated from OHPP, which does not add extra computations to the algorithm. To make velocity-aware updates, algorithm iterates through devices for each prioritised microservice instance which results in $\mathcal{O}(S.M.I.|D'|)$ iterations during the worst case. This results in time complexity of $\mathcal{O}(S.M.I.|D'|)$ for the Evolution phase. The time complexity of DNCPSO and FSPP for this phase becomes, $\mathcal{O}(S.M.|D'|)$ due to the lack of throughput aware scalability of microservices.

Although the novel approaches introduced in QMPSO adds extra complexity to the algorithm, lack of these features results in a slower convergence rate, convergence to local optimums, lower QoS satisfaction, and lower resource utilisation as demonstrated in by the results in Sections 5.3.1 and 5.3.2. Thus, this trade-off between accuracy and extra computation time is vital in solving the microservices-based application placement in fog. Moreover, the added time complexity due to these improvements is limited to linearithmic increase for sorting operations and an increase by a factor of I for throughput aware scaling of microservices. Thus, QMPSO reaches a fair trade-off between performance of the placement and extra time-complexity by maximising QoS satisfaction and resource usage of the placement while avoiding a drastic increase in time complexity.

6. Conclusions and future work

Rapid growth in IoT has resulted in the emergence of diverse and complex applications developed using the microservices architecture. Due to the latency critical and bandwidth hungry nature of these applications, they are scheduled within distributed, resource-constrained and heterogeneous fog devices. To fully leverage the capabilities of fog devices to support multiple heterogeneous applications, we exploited the granularity and scalability of microservice architecture and formulated the fog application placement problem as a Lexicographic Combinatorial Optimisation Problem for batch placement of IoT applications, where QoS satisfaction and optimum resource usage are the primary and secondary objectives respectively. To solve the placement problem, we proposed an algorithm by adapting and improving the S-CLPSO technique. Extensive experiments are carried out to evaluate the effectiveness of the proposed technique under two aspects; convergence improvement against other adaptations of the S-CLPSO and efficiency of the resultant placement against state-of-the-art techniques. Obtained results depict that our approach successfully navigates large solution spaces and generates placements with higher QoS satisfaction (35% and 70% improvement in makespan and budget satisfaction respectively) while ensuring optimum fog and cloud resource usage.

As part of future work, we plan to extend our proposed approach to explore other research issues including mobility of both IoT and fog layer devices; fault tolerance under node failures and communication link failures with emphasise on the microservices architecture of applications; dynamic scheduling of microservices. We plan to implement the proposed algorithm in a real-world fog framework such as FogBus [40].

CRediT authorship contribution statement

Samodha Pallewatta: Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing. **Vassilis Kostakos:** Supervision. **Rajkumar Buyya:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Mahmud, R. Kotagiri, R. Buyya, Fog computing: A taxonomy, survey and future directions, in: Internet of Everything, Springer, 2018, pp. 103–130.

[2] M. Villari, M. Fazio, S. Dustdar, O. Rana, R. Ranjan, Osmotic computing: A new paradigm for edge/cloud integration, IEEE Cloud Comput. 3 (6) (2016) 76–83.

[3] A. Kaur, R. Kumar, S. Saxena, Osmotic computing and related challenges: A survey, in: Proceedings of the 6th International Conference on Parallel, Distributed and Grid Computing (PDGC), IEEE, 2020, pp. 378–383.

[4] M. Fowler, J. Lewis, Microservices a definition of this new architectural term, 2014, URL: https://martinfowler.com/articles/microservices.html.

[5] C.T. Joseph, K. Chandrasekaran, Straddling the crevasse: A review of microservice software architecture foundations and recent advancements, Softw. - Pract. Exp. 49 (10) (2019) 1448–1484.

[6] S. Pallewatta, V. Kostakos, R. Buyya, Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments, in: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, 2019, pp. 71–81.

[7] D.T. Nguyen, H.T. Nguyen, N. Trieu, V.K. Bhargava, Two-stage robust edge service placement and sizing under demand uncertainty, IEEE Internet Things J. (2021).

[8] R. Mahmud, S.N. Srirama, K. Ramamohanarao, R. Buyya, Profit-aware application placement for integrated fog–cloud computing environments, J. Parallel Distrib. Comput. 135 (2020) 177–190.

[9] Y. Xie, Y. Zhu, Y. Wang, Y. Cheng, R. Xu, A.S. Sani, D. Yuan, Y. Yang, A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment, Future Gener. Comput. Syst. 97 (2019) 361–378.

[10] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H.C. Cankaya, Q. Zhang, W. Xie, J.P. Jue, FogPlan: a lightweight qos-aware dynamic fog service provisioning framework, IEEE Internet Things J. 6 (3) (2019) 5080–5096.

[11] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: Proceedings of the 1st IEEE International Conference on Fog and Edge Computing (ICFEC), IEEE, 2017, pp. 89–96.

[12] A.A. Abdellatif, A. Mohamed, C.F. Chiasserini, M. Tlili, A. Erbad, Edge computing for smart health: Context-aware approaches, opportunities, and challenges, IEEE Netw. 33 (3) (2019) 196–203.

[13] R. Ke, Y. Zhuang, Z. Pu, Y. Wang, A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on IoT devices, IEEE Trans. Intell. Transp. Syst. (2020).

[14] R. Mahmud, S. Pallewatta, M. Goudarzi, R. Buyya, IFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments, 2021, arXiv preprint arXiv: 2109.05636.

[15] C. Guerrero, I. Lera, C. Juiz, Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures, Future Gener. Comput. Syst. 97 (2019) 131–144.

[16] A. Brogi, S. Forti, Qos-aware deployment of IoT applications through the fog, IEEE Internet Things J. 4 (5) (2017) 1185–1192.

[17] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, P. Leitner, Optimized IoT service placement in the fog, Serv. Orient. Comput. Appl. 11 (4) (2017) 427–443.

[18] S. Deng, Z. Xiang, J. Taheri, K.A. Mohammad, J. Yin, A. Zomaya, S. Dustdar, Optimal application deployment in resource constrained distributed edges, IEEE Trans. Mob. Comput. (2020).

[19] W.-N. Chen, J. Zhang, A set-based discrete PSO for cloud workflow scheduling with user-defined QoS constraints, in: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2012, pp. 773–778.

[20] A. Verma, S. Kaushal, A hybrid multi-objective particle swarm optimization for scientific workflow scheduling, Parallel Comput. 62 (2017) 1–19.

[21] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the ICNN'95-International Conference on Neural Networks, Vol. 4, IEEE, 1995, pp. 1942–1948.

[22] W.-N. Chen, J. Zhang, H.S. Chung, W.-L. Zhong, W.-G. Wu, Y.-H. Shi, A novel set-based particle swarm optimization method for discrete optimization problems, IEEE Trans. Evol. Comput. 14 (2) (2009) 278–300.

[23] W.-N. Chen, D.-Z. Tan, Set-based discrete particle swarm optimization and its applications: a survey, Front. Comput. Sci. 12 (2) (2018) 203–216.

[24] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295.

[25] AWS, AWS fargate pricing, 2021, URL: https://aws.amazon.com/fargate/pricing/, last accessed on 09/09/2021.

[26] Azure, Container instances pricing, 2021, URL: https://azure.microsoft.com/en-au/pricing/details/container-instances/, last accessed on 09/09/2021.

[27] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments, Softw. - Pract. Exp. 47 (9) (2017) 1275–1296.

[28] CPLEX User's Manual, Ibm ilog cplex optimization studio, 1987, pp. 1987–2018, Version 12.

[29] O. Grodzevich, O. Romanko, Normalization and other topics in multi-objective optimization, in: Proceedings of the Fields–MITACS Industrial Problems Workshop, 2006.

[30] S. Edirisinghe, C. Ranaweera, C. Lim, A. Nirmalathas, E. Wong, Universal optical network architecture for future wireless LANs, J. Opt. Commun. Netw. 13 (9) (2021) D93–D102.

[31] D. Minovski, N. Ogren, C. Ahlund, K. Mitra, Throughput prediction using machine learning in lte and 5g networks, IEEE Trans. Mob. Comput. (2021).

[32] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H.C. Cankaya, Q. Zhang, W. Xie, J.P. Jue, FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework, IEEE Internet Things J. 6 (3) (2019) 5080–5096.

[33] M. Goudarzi, H. Wu, M. Palaniswami, R. Buyya, An application placement technique for concurrent IoT applications in edge and fog computing environments, IEEE Trans. Mob. Comput. 20 (4) (2020) 1298–1311.

[34] M. Goudarzi, M.S. Palaniswami, R. Buyya, A distributed deep reinforcement learning technique for application placement in edge and fog computing environments, IEEE Trans. Mob. Comput. (2021).

[35] S. Tuli, S. Ilager, K. Ramamohanarao, R. Buyya, Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks, IEEE Trans. Mob. Comput. (2020).

[36] E. El Haber, H.A. Alameddine, C. Assi, S. Sharafeddine, UAV-aided ultra-reliable low-latency computation offloading in future IoT networks, IEEE Trans. Commun. 69 (10) (2021) 6838–6851.

[37] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, M. Gidlund, Industrial internet of things: Challenges, opportunities, and directions, IEEE Trans. Ind. Inf. 14 (11) (2018) 4724–4734.

[38] R. Mahmud, R. Buyya, Modelling and simulation of fog and edge computing environments using iFogSim toolkit, in: Fog and Edge Computing: Principles and Paradigms, 2019, pp. 1–35.

[39] Azure, Move mainframe compute to azure, 2021, URL: https://docs.microsoft.com/en-us/azure/virtual-machines/workloads/mainframe-rehosting/concepts/mainframe-compute-azure, last accessed on 09/09/2021.

[40] Q. Deng, M. Goudarzi, R. Buyya, FogBus2: a lightweight and distributed container-based framework for integration of IoT-enabled systems with edge and cloud computing, in: Proceedings of the International Workshop on Big Data in Emergent Distributed Environments, 2021, pp. 1–8.

Samodha Pallewatta is a Ph.D. student at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia. Her research interests encompass Fog/Edge Computing, Internet of Things (IoT) and Distributed Systems. She is one of the contributors of the iFogSim simulator, used extensively for resource management research in Fog/Edge computing.

Dr. Vassilis Kostakos is a professor at the School of Computing and Information Systems, University of Melbourne, Melbourne, Australia. His research includes Internet of Things, ubiquitous computing, human–computer interaction and social computing. Dr. Kostakos is a Marie Curie Fellow, a Fellow in the Academy of Finland Distinguished Professor Program, and a Founding Editor of the Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies.

Dr. Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=149, g-index=322, 116,700+ citations).