CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

ORCA – Online Research @ Cardiff

This is an Open Access document downloaded from ORCA, Cardiff University's institutional repository:https://orca.cardiff.ac.uk/id/eprint/148052/

This is the author's version of a work that was submitted to / accepted for publication.

Citation for final published version:

Kaur, Amanjot, Auluck, Nitin and Rana, Omer 2023. Real-time scheduling on Hierarchical Heterogeneous Fog Networks. IEEE Transactions on Services Computing 16 (2) , pp. 1358-1372. 10.1109/TSC.2022.3155783

Publishers page: http://dx.doi.org/10.1109/TSC.2022.3155783

Please note:
Changes made as a result of publishing processes such as copy-editing, formatting and page numbers may not be reflected in this version. For the definitive version of this publication, please refer to the published source. You are advised to consult the publisher's version if you wish to cite this paper.

This version is being made available in accordance with publisher policies. See http://orca.cf.ac.uk/policies.html for usage policies. Copyright and moral rights for publications made available in ORCA are retained by the copyright holders.

Real-Time scheduling on Hierarchical Heterogeneous Fog Networks

Amanjot Kaur, Nitin Auluck and Omer Rana, *Member, IEEE*

Abstract—Cloud computing is widely used to support offloaded data processing for various applications. However, latency constrained data processing has requirements that may not always be suitable for cloud-based processing. Fog computing brings processing closer to data generation sources, by reducing propagation and data transfer delays. It is a viable alternative for processing tasks with real-time requirements. We propose a scheduling algorithm RTH^2S (Real Time Heterogeneous Hierarchical Scheduling) for a set of real-time tasks on a heterogeneous integrated fog-cloud architecture. We consider a hierarchical model for fog nodes, with nodes at higher tiers having greater computational capacity than nodes at lower tiers, though with greater latency from data generation sources. Tasks with various profiles have been considered. For the regular profile jobs, we use least laxity first (LLF) to find the preferred fog node for scheduling. In case of "tagged" profiles, based on their tag values, the jobs are split in order to finish execution before the deadline, or the LLF heuristic is used. Using HPC2N workload traces across 3.5 years of activity, the real-time performance of RTH^2S versus comparable algorithms is demonstrated. We also consider Microsoft Azure-based costs for the proposed algorithm. Our proposed approach is validated using both simulation (to demonstrate scale up) as well as a lab-based testbed.

Index Terms—Fog computing, cloud computing, real-time scheduling, fog node hierarchy.

1 Introduction

Fog computing involves the use of a number of nodes/micro data centers located in close proximity to users and data generation sources [1]. As there could be significant propagation delays between the data generation sources and the cloud data center, fog computing provides computing capability closer to the data source. An example of such a job could be a surveillance camera at a security facility that detects an intruder and alerts relevant authorities. For such jobs, processing times need to be in the sub-second range – a constraint that a remotely located cloud data center may not be able to guarantee. In contrast, fog nodes, owing to their proximity to users and data generation sources, can execute such critical tasks with a lower invocation latency. To support the diverse execution requirements of real time applications, the fog node architecture may be hierarchical [2]. Nodes in proximity to a user (lower tier) are considered to have lower computational capability compared to fog nodes at higher tiers, but at a greater geographical distance (i.e. higher latency) from data sources. A trade-off exists therefore between processing capability of fog nodes and propagation delay to users.

"Smart" transportation provides a relevant scenario in this context – as discussed by the Open Fog Consortium [2]. The data volume generated by connected cars can become challenging to transfer to the cloud for processing, as this could result in network congestion and increase in processing times – leading to missed deadlines for tasks that process this data. A smart transportation scenario can involve a number of sensors present on roadside units, such as weather sensors for ice, snow, water, and roadside sensors for speed, volume and traffic monitoring (e.g cameras) etc. Smart cars communicate with these roadside sensors or with other cars in order to make specific decisions – offering services like infotainment, supporting collision avoidance, processing of prior information regarding poor road conditions or traffic congestion. Note that these jobs can have diverse execution requirements and deadlines. An example of a *small* job with a tight deadline includes real time "sensing" or monitoring of a parameter (or calculating averages or max./min. across these parameters over a time window), such as temperature, wind-speed, humidity, rainfall etc. These jobs are typically in the milliseconds range. These jobs need to be scheduled on a tier-1 fog node located in proximity to the user. On the other hand, jobs with less stringent deadlines but greater execution requirements could include control tasks, such as managing the properties of an infotainment system. This is an example of a *medium* sized job and may be executed at tier-2 or tier-3 fog nodes. Finally, *large* jobs with loose deadlines could be executed at the cloud. An example of such a job could be batch data processing, e.g. determining how many vehicles crossed an intersection (involving integration across multiple sensors), analysis of traffic patterns at a junction or road intersection, etc. It is pertinent to mention that these fog nodes would be distributed and would be available even in the event of a data comms. network outage. Hence, they would make the system more resilient. The key contributions of the paper are as follows:

(i) a multi-tier hierarchical fog-cloud real time scheduling algorithm RTH^2S taking account of device heterogeneity. We propose a mathematical model for an n-tier fog cloud architecture that schedules jobs onto fog/cloud processors while meeting their deadline requirements.

- *A. Kaur and N. Auluck are with the Department of Computer Science and Engineering, Indian Institute of Technology Ropar, Punjab, India, 140001. E-mail: 2017csz0014@iitrpr.ac.in, nitin@iitrpr.ac.in.*
- *O. Rana is with the School of Computer Science and Informatics, Cardiff University, Cardiff, UK, CF243AA, email: RanaOF@cardiff.ac.uk.*

Manuscript received April 19, 2005; revised August 26, 2015.

(ii) RTH^2S works for both regular and tagged job profiles. The algorithm either finds a preferred fog node for job execution, or splits the job based on a combination of its size and deadline requirements.
(iii) Using both simulation and a prototype test bed, we demonstrate the performance of the proposed algorithm RTH^2S in enhancing a key metric used to measure *benefit*: Success Ratio (SR) – while considering task load, propagation delay, heterogeneity and job profiles. Further, the impact of the tiered fog architecture on the scheduling performance is also discussed.

This paper is organized as follows. Section 2 includes a discussion of related work. The system model, notation and problem formulation is described in section 3. An orchestration protocol to support automatic system functioning is discussed in Section 4. The proposed algorithm is presented in section 5. Section 6 discusses results. Finally, section 7 concludes the paper and discusses future work.

2 Related Work

The Open Fog Consortium (involving a number of industry partners, e.g. Cisco, Intel, Microsoft, Dell etc.) has proposed a reference architecture [2] with several use cases for fog computing: smart transportation, smart buildings, airport security, and so on. The extension of network resources from cloud to fog nodes yields a rich environment which can provide storage, computation and communication resources over the network [18]. Capacity planning and optimisation of a fog-based system may be analysed using the iFogSim simulator [3], enabling various resource management strategies in fog-cloud architectures to be considered. iFogSim matches fog node capability (as Million Instructions Per Second (MIPS), memory, and network connectivity) with task capability (defined using similar metrics as fog node capability). The simulator enables understanding the trade-off between computational capability and power consumption of a fog node, and latency of executing an application task.

A survey of fog computing [4], [13] explores a number of research trends – differentiating characteristics of fog and cloud computing. In [16], the authors pitch fog computing as a crucial element for Internet of Things (IoT), and develop a mathematical model to assess the suitability of fog computing in IoT [20]. In [5], the authors observe that fog nodes/cloudlets provide an acceptable interactive response in human cognition, owing to their physical proximity and one-hop network latency. Several papers, given the context of fog-based system usage, have focused on minimising latency in such environments [21].

In our previous work, we consider the real-time scheduling of single tier fog nodes [11] on homogeneous fog nodes, i.e. all the fog nodes were assumed to have identical processing capabilities, with the interpretation that a job will have identical execution costs on all fog nodes. In [9], the authors schedule tasks in real time on identical processors based on their deadline requirements. An energy-efficient fog computing framework has been proposed in [8]. The computation resources are shared with multiple neighbor helper nodes and an optimal scheduling decision is determined for a task node. In [10], the authors proposed a real time algorithm called DEBTS for achieving a balanced system performance in terms of service delay and energy consumption. However, the authors have not considered heterogeneous fog nodes in their work. In [17], a fog based delay-optimal task scheduling algorithm has been proposed. The authors consider a heterogeneous fog network as a part of dynamic wireless networks in [19]. In [14], the placement of tasks on heterogeneous fog nodes has been explored, on the basis of privacy tags. In [33], the authors discuss resource allocation by ranking fog devices based on processing, bandwidth and latency, and assigning processors to deadline-based tasks. In [34], the authors propose a dynamic request dispatching algorithm, which minimizes energy consumption and timeliness by using the Lyapunov Optimization Technique. The authors propose an adaptive queuing weight (AQW) resource allocation and real-time offloading technique in a heterogeneous fog environment in [35]. In [36], the authors reduce the waiting time of delay-sensitive tasks by using a multilevel-feedback queue and minimizing the starvation problem of low priority tasks. However, all these approaches focus on a single tier of fog nodes between the edge and cloud systems, and cannot be applied directly to multi-tier fog cloud architectures.

There has been some work in multi-tier hierarchical fog cloud scheduling. In [24], the authors proposed a hierarchical edge computing architecture with identical resources in each tier, however without consideration of real time scheduling. In [27], the authors propose a multi-tier fog cloud architecture, and divide tasks into low & high priority. In [28], a hierarchical fog cloud architecture (limited to 2-tier cloudlets) and a workload allocation scheme is proposed, which attempts to minimise the response time of user requests. In [29], [30] the authors proposed a multi-layer heterogeneous architecture for task offloading to minimise response time, without considering real time tasks. In [31], a component based scheduler for a multi-tier fog cloud architecture is proposed. However, authors consider only two tiers of fog nodes in their work, and measure the results using simulation – without taking account of a real workload. The effective mapping of jobs to a group of heterogeneous fog nodes is no doubt a challenging problem. To the best of our knowledge, no work has looked into heterogeneous, hierarchical real time scheduling of "regular" as well as "tagged" profiled tasks on fog cloud architectures, supporting both "inter-level heterogeneity" as well as "intra-level heterogeneity".

3 System Model

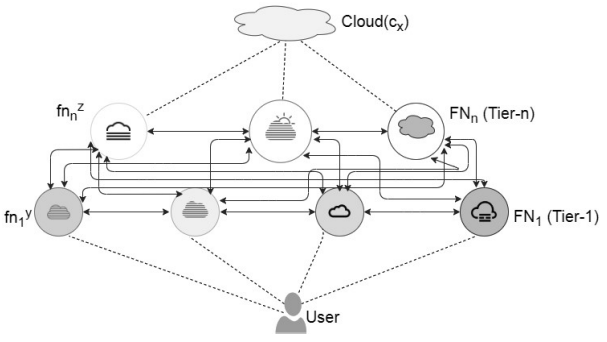

Fig. 1. Fog Architecture

TABLE 1
Key Notation

C	cloud data center set
c_x	cloud data center $c_x \in C$
FN	set of all fog nodes
FN_1	set of tier-1 fog nodes
fn_1^y	y^{th} tier-1 fog node $\in FN_1$
FN_2	set of all tier-2 fog nodes
fn_2^y	y^{th} tier-2 fog node $\in FN_2$
FN_3	set of tier-3 fog nodes
fn_3^y	y^{th} tier-3 fog node $\in FN_3$
J	set of all jobs
N	total number of jobs
J_S	set of all small jobs $\in J$
j_s^k	k^{th} small job $\in J_S$
J_M	set of all medium jobs $\in J$
j_m^k	k^{th} medium job $\in J_M$
J_L	set of all large jobs $\in J$
j_l^k	k^{th} large job $\in J_L$
$c(fn_i^j)$	capacity of j^{th} fog node at i^{th} tier
$c(c_x)$	capacity of cloud data center c_x
$d(j^k)$	deadline for k^{th} job
$pd(j^k, fn_1)$	propagation delay between job and tier-1 fog node
$pd(j^k, fn_2)$	propagation delay between job and tier-2 fog node
$pd(j^k, fn_3)$	propagation delay between job and tier-3 fog node
$pd(j^k, c_x)$	propagation delay between job and cloud data center
$\tau(j^k, fn)$	job execution cost on fog node
$pfn_z(j^k)$	z^{th} preferred fog node of job j^k

3.1 Proposed architecture

The proposed architecture is illustrated in Fig. 1. Table 1 summarises the notation used in our approach – where a set of fog nodes is given by FN. We assume that a hierarchy of fog nodes exists – as outlined in [2]. At the lowest level (i.e. closest to the user), we have tier-1 fog nodes, followed by tier-2 fog nodes at the next level, and then tier-3 fog nodes at a higher level (i.e. closer to the data center). As a general rule, as one moves up in the hierarchy, the execution capacity of the fog nodes increases. However, on the flip side, the communication distance to the data generation sources also increases, increasing the propagation delay. Note that Fig. 1 depicts n tiers of fog nodes. In this work, we consider three tiers of fog nodes i.e n = 3. Based on the system requirement, the architecture can be extended to n-tiers of fog nodes.

3.2 Notation

The set of all tier-1 fog nodes is given by FN_1, the set of all tier-2 fog nodes is given by FN_2, and the set of all tier-3 fog nodes is given by FN_3. The k^{th} fog node at tier 1, 2, and 3 is given by fn_1^k, fn_2^k, and fn_3^k respectively. At the top of the hierarchy, we have a cloud data center $c \in C$, where C is the set of all cloud data centers. Each cloud data center could potentially belong to a different cloud provider e.g. Google, Amazon, Microsoft. In our work, we consider a single cloud data center, but the proposed approach can be generalised to multiple providers. The capacity of a particular fog node or the cloud data center is given by c. This execution capacity is given in terms of Millions of Instructions per Second, or MIPS. The popular fog simulator iFogSim [3] models computational node execution capacity in MIPS, so we have chosen MIPS in our model – to make our model compatible with iFogSim. An application that needs to be executed consists of the set of all jobs in the system is denoted by J, such that $J = \{J_1, J_2, J_3, J_4, ...\}$. Throughout this paper, we use the term job and task interchangeably. The deadline of the k^{th} job is denoted by $d(j^k)$. The propagation delay between job j^k and tier-1 fog node fn_1 is given by $pd(j^k, fn_1)$. Likewise, $pd(j^k, fn_2)$, and $pd(j^k, fn_3)$ represents the propagation delay between job j^k and tier-2 fog node fn_2, job j^k and tier-3 fog node fn_3 respectively. Finally, the propagation delay between job j^k and cloud data center c_x is represented as $pd(j^k, c_x)$. Note that fn_1 could be any tier-1 fog node $\in FN_1$, fn_2 could be any tier-2 fog node $\in FN_2$, and so on. The execution cost of the k^{th} job on a tier-1 fog node is denoted by $\tau(j^k, fn_1)$.

TABLE 2
Representative smart car tasks

Job	Execution cost (MIPS)	Memory usage (GBs)	Job type	Job description
j_s^1	250	0.34	small	rainfall
j_s^2	155	0.28	small	temperature
j_m^1	3000	0.81	medium	object rec.
j_m^2	1850	0.67	medium	wiper cont.
j_l^1	6700	1.78	large	traffic patt.
j_l^2	7200	2.01	large	non-crit. updates

In order the classify the jobs based on their execution requirements, we consider three sets of jobs: small (J_S), medium (J_M), and large (J_L). So, j^k could be any small, medium, or large job $\in, J_S, J_M, J_L$. Small jobs (J_S) are the jobs that require less processing power to execute, medium jobs (J_M) are the jobs that require moderate processing power to run, and large jobs (J_L) are the jobs that need high processing power for execution. Although there are a number of smart car data sets that focus on speed, traffic patterns, car images, we could not find any data set that specifies the CPU execution requirements, or memory usage of various smart car tasks. Some representative tasks from the smart automobile use case are given in table 2. The nature of these jobs has been inspired from [22]. We consider three types of deadlines: tight(T), moderate (M), and loose(L). The deadline category is decided from Deadline Factor (DF) defined in Section VI-B Table 3 depicts the priority assignment based on the job sizes and their deadlines. We have considered 9 * 9 combinations of job sizes and deadlines. In general, as deadlines may be a directly proportional to execution costs, small jobs have lower execution costs and tight deadlines. Hence, they are assigned to tier-1 fog nodes located at the closest proximity to users. Typically, medium jobs have higher execution costs and looser deadlines than small jobs, and are assigned to the tier-2 or tier-3 fog nodes. Finally, the cloud runs large jobs with loose deadlines. All these jobs can be considered to have regular job profiles. The set of regular job profiles are denoted as R.

However, based on the user requirements, jobs may not fit into the above profiles. We call such job profiles as "tagged" – denoted as T. For example, if a medium job has a tight deadline, or a large job has a tight/ moderate deadline, despite being medium/large, they need to execute in the lower fog layers to meet their deadline requirements. We propose to split such jobs so that they can be executed on

lower capability resources. All these jobs are assigned $tag1$. Such jobs need to be assigned a high priority. As another example, small jobs may have moderate/loose deadlines and medium jobs may have loose deadlines. These jobs are tagged as $tag2$. Such jobs can be assigned a lower priority. The set T has two subsets: $T1$ subset for $tag1$ jobs, and $T2$ subset for $tag2$ jobs. Table 4 depicts the tag assignment.

We consider various kinds of heterogeneity in our model. By their nature, tier-1 fog nodes (FN_1), tier-2 fog nodes (FN_2), tier-3 fog nodes (FN_3), and the cloud data center (c_x) are heterogeneous with respect to each other. This means that execution capacity of FN_1 nodes is different from that of FN_2 and FN_3, which is further different from c_x. In general, the order of execution capacity is: $c_x > FN_3 > FN_2 > FN_1$. Higher execution capacity implies faster execution rates, so assigned tasks will finish earlier. We call such heterogeneity "inter-level-heterogeneity". In addition, we consider "intra-level-heterogeneity".

TABLE 3
Priority Level Assignment

	tight (T)	moderate (M)	loose (L)
Small job j_s	P_1	P_2	P_3
Medium job j_m	P_1	P_2	P_3
Large job j_l	P_1	P_2	P_3

TABLE 4
Tag Assignment

	tight (T)	moderate (M)	loose (L)
Small job j_s	x	$tag2$	$tag2$
Medium job j_m	$tag1$	x	$tag2$
Large job j_l	$tag1$	$tag1$	x

The total number of jobs is given by J. This is the sum of all small, medium and large jobs: $J = J_S + J_M + J_L$. The commencement time of a job $j^k \in J$ on a fog node $fn \in FN$ is denoted by $ct(j^k, fn)$. In this work, we consider that there are no precedence constraints among various jobs, so all jobs are independent of each other. However, if the jobs are split on various fog nodes, then we consider the aggregate finish time from the entry fog node to exit fog node. Hence, a job may start at time 0. Note that, j^k can be a whole job or a part of split job. The execution cost of a job $j^k \in J$ on a fog node $fn \in FN$ is given by $\tau(j^k, fn)$. Since the processing elements are heterogeneous, the execution cost of a job can be different on different processing units FN, C.

3.3 Real-time constraints

The finish time of a job $j^k \in J$ on a fog node $fn \in FN$ is denoted by $ft(j^k, fn)$. The finish time of a job j^k on a fog node fn may be modelled as:

$$ft(j^k, fn) = ct(j^k, fn) + \tau(j^k, fn) + pd(j^k, fn) \quad (1)$$

The jobs are real-time and need to finish by their deadline. For unsplit jobs, no overheads are there.

$$ft(j^k, fn) \leq d(j^k) \quad (2)$$

Since this is a heterogeneous system, we need to exercise caution while assigning jobs to fog nodes and the cloud. We use the concept of a preferred fog node pfn of job j^k – a node on which the job is most likely to meet its deadline requirements. By selecting a pfn for j^k, the algorithm takes processor heterogeneity into account.

For jobs with regular or $tag2$ profiles, we use the job laxity for finding the preferred fog node. The laxity of a job j^k is denoted by $l(j^k)$, and is the difference between its finish time and its deadline [7]. Formally:

$$l(j^k) = ft_{min}(j^k) - d(j^k) \quad (3)$$

Since each job j^k can have different finish times on different fog nodes, we consider the finish time that has the minimum value in our estimation of laxity. Hence, $pfn_1(j^k)$ is given as:

$$pfn_1(j^k) = pu : l(j^k, pu) < l(j^k, pu') \quad (4)$$

In eq. (4) above, $pu, pu' \in FN, C$ && $pu \neq pu'$. In other words, assigning a job j^k to its preferred fog node 1 pfn_1 results in the minimum laxity value of the job. Likewise, the fog node that results in the next lowest laxity value becomes pfn_2, and so on.

The following equation makes sure that the job's requirement is not more than the fog node capacity:

$$R(fn) \geq r(j^k, fn) \quad (5)$$

Here, $R(fn)$ represents the resource capability of fn and $r(j^k, fn)$ represents the resource requirement of j^k.

3.4 Job splitting constraints

For $tag1$ jobs, it may not be possible to execute the whole job on a single fog node due to their limited computational capacity and strict deadline requirement. Hence, we propose splitting such jobs and assigning the generated sub-jobs to various fog nodes. While splitting the jobs, we need to take care of the fog node's propagation delay and computation power. A fog node with less propagation delay (close to the user) may have low computation power. On the other hand, a fog node with high computation power may be located a bit far from the user. So, we consider an inverse of both parameters to calculate the value of $Y(fn)$ for fn.

$$Y(fn) = \frac{1}{pd(j^k, fn)} + \frac{1}{c(j^k, fn)} \quad (6)$$

To divide the job j^k into sub-jobs w_i, we use eq. 7:

$$w_i(fn) = j^k * Y(fn) \quad (7)$$

The sub-job size w_i is calculated by considering the propagation delay pd and capacity c of fog node. The number of sub-jobs depend upon the size of the job and characteristics of the fog node. Job j^k is calculated in eq. 8:

$$w_{left} = j^k - \sum w_i \quad (8)$$

Overheads Θ involved in splitting large jobs into smaller ones has three components: (i) delay involved in transmitting jobs to fog nodes. Job j^k is split into smaller chunks denoted by w_i; (ii) finish time ft of the job j^k; (iii) delay in receiving results. The output of each sub-job with input w_i is denoted by w_o. The bandwidth of the network connection between user u_i and fog node fn is denoted by bw.

$$\Theta(j^k) = \sum \frac{w_i}{bw(u_i, fn)} + ft(j^k) + \sum \frac{w_o}{bw(fn, u_i)} \quad (9)$$

The jobs are real-time and need to finish by their deadline. We need to take the overheads into account for split jobs.

$$\Theta(j^k) \leq d(j^k) \tag{10}$$

3.5 Job precedence model

A workflow is defined as a set of small interdependent jobs modeled as a Directed Acyclic Graph (DAG). Each DAG is defined by the tuple (J, E, τ, cc), where J is the set of jobs, and E is the set of edges defining data dependencies between them. Let $J = (j^1, j^2, ..., j^t)$ be the set of t jobs in the workflow. τ is the set of execution costs, and cost for job $j^k \in J$ is denoted by $\tau(j^k)$. The set cc consists of communication costs, and each edge from job j^k to job j^i, $e_{k,i} \in E$ has a cost $cc(j^k, j^i)$ associated with it. The data flow dependency from job j^k to job j^i is defined by an edge $e_{k,i} \in E$, with a precedence constraint that the job j^i can only start after the completion of job j^k. Suppose, a job j^k is scheduled on a fog node fn. Let $mst(j^k, fn)$ and $mct(j^k, fn)$ be the minimum start time and minimum completion time for job j^k on fn respectively, when fn is available for the execution of job j^k.

$$mst(j^k, fn) = 0, if\ pred(j^k) = \phi \tag{11}$$

$$mst(j^k, fn) = max(act(j^p) + cc(j^p, j^k)),$$
$$for\ each\ j^p \in pred(j^k) \tag{12}$$

$$mct(j^k, fn) = mst(j^k, fn) + \tau(j^k, fn) \tag{13}$$

If both the parent and child jobs are assigned to the same fog node, the cost cc will be zero. After a job is assigned to a fog node, the mst and mct become the job's actual start time (ast) and actual completion time (act), respectively. At last, the workflow's finish time (ft) is equal to the actual completion time of the last job , j^{exit}

$$ft(DAG) = act(j^{exit}) \tag{14}$$

We take a DAG as input for the dependent tasks, convert this into an ordered tasks list, and then submit these tasks for execution. For workflow/DAG execution, a tool like Pegasus [37] may be used.

3.6 Queuing delay

As the processing capability of the fog nodes FN is considerably less than the processing capability of the cdc c_x, queuing can occur in fog nodes when the jobs are large in number. The scheduled jobs on cdc c_x can generally execute without any queuing delay. Due to the limited processing capability of fog nodes, we assume that each fog node maintains a queue to buffer the jobs. The queue length of fn at $t + 1^{th}$ instance can be defined as follows [38]:

$$q(fn, t+1) = max(q(fn, t) + a(fn, t) - \mu(fn, t), 0) \tag{15}$$

Here, $q(fn, t+1)$ is the queue of fn at $(t+1)^{th}$ instance, and $q(fn, t)$ is the queue of fn at t^{th} instance. $q(fn, t)$ represents the number of jobs leaving the queue of fn in the t^{th} time slot (jobs processed by fog node). $a(fn, t)$ denotes the number of jobs arriving at fn in the t^{th} time slot. We add the queuing delay to equation (1) to calculate the finish times.

3.7 Cost and objective function constraints

Monetary Cost (MC) is estimated by considering the weighted average of the execution cost τ and propagation delay pd of j^k on a fog node fn. This weighted average is multiplied by the price of the fog node on which the jobs are being offloaded [26].

$$MC(j^k, fn) = \frac{price(fn) * (w_1 * \tau(j^k, fn) + w_2 * pd(j^k, fn))}{w_1 + w_2} \tag{16}$$

Similarly, the Monetary Cost (MC) of cloud data center c_x can be defined as follows:

$$MC(j^k, c_x) = \frac{price(c_x) * (w_1 * \tau(j^k, c_x) + w_2 * pd(j^k, c_x))}{w_1 + w_2} \tag{17}$$

The overall Monetary Cost of the system is given by:

$$MC_{system} = \sum_{k=1}^{J} \sum_{n=1}^{N} \sum_{j=1}^{m} MC(j^k, fn_n^j) + \sum_{k=1}^{J} \sum_{x=1}^{C} MC(j^k, c_x) \tag{18}$$

Success Ratio (SR) at n^{th} fog tier is the percentage ratio of n-tier jobs that finish execution before their deadline, to the total number of jobs submitted to fog nodes FN_n. The Success Ratio of n^{th} tier fog nodes is given by:

$$SR_{FN_n} = \frac{j'(FN_n)}{j(FN_n)} * 100 \tag{19}$$

Here, $j'(FN_n)$ are the total number of FN_n bound jobs that finish before their deadlines, i.e. $ft(j^i, fn_n) \leq d(j^i)$. $j(FN_n)$ are the total number of jobs submitted to the n^{th} fog tier. Here, $fn_n \in FN_n$ and $j^i, d(j^i) \in j'$.

The Success Ratio (SR_C) on the cloud data center can be defined as follows:

$$SR_C = \frac{j''(c_x)}{j(c_x)} * 100 \tag{20}$$

Here, j'' are the total number of cloud bound jobs that have finished before their deadlines, i.e. $ft(j^i, c_x) \leq d(j^i)$. $j(c_x)$ are the total number of jobs submitted to the cloud data center. Here, c_x is the only cloud data center and $j^i, d(j^i) \in j''$.

Overall, the Success Ratio of the fog nodes is given by:

$$SR_{system} = SR_{FN_1} + SR_{FN_2} + ... + SR_{FN_n} + SR_C \tag{21}$$

Given this context and set of definitions, we can formally define the research problem as:

"Given a set of jobs $J(J_S, J_M, J_L)$, a set of fog nodes $FN(FN_1, FN_2, FN_3)$ and a cloud data center c_x, with heterogeneous execution capacity schedule the jobs on their preferred fog node $pfns$, or split the job onto fog tiers according to the priority assignment of Table 3, and tag assignment of Table 4, s.t. SR_{system} is maximised".

4 Orchestration Protocol

We adopt a decentralised fog cloud architecture driven by Orchestrating agents (OAs), as proposed in [6]. Fig. 2 shows the conceptual architecture of the orchestration mechanism for the distributed fog cloud architecture. Here, FN_n represents the n^{th} fog node tier of the architecture. In this work, we considering n = 3, though n can be varied based on the application requirement. An OA is present on each

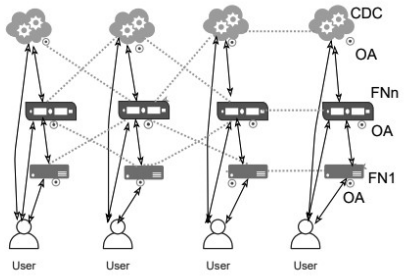

Fig. 2. Distributed fog cloud architecture

computing device. A job specific instance is created by OAs. The OAs cooperate with each other to achieve the goal of the scheduling algorithm: minimising the overall latency of the system, or increasing the success ratio of the system. As demonstrated in the figure, a user can submit jobs to the fog devices or cloud data center. Each user has a network connection to the FN_1. The FN_1 are the fog nodes which can execute jobs in the least latency. The fog node tier FN_1 is further connected to next tier of fog nodes i.e. FN_2 followed by FN_3. Finally, we have a cloud data center at the top most layer of the hierarchy.

5 Proposed Algorithm

In this section, we describe our proposed scheduling scheme RTH^2S. As mentioned in section III, we consider three types of jobs: small (J_S), medium (J_M), and large (J_L). Likewise, we have resources of diverse execution capacities, tier-1 fog nodes (FN_1), tier-2 fog nodes (FN_2), tier-3 fog nodes (FN_3), and the cloud data center (c_x), which are heterogeneous with respect to each other. For a particular job $j^k \in J$, the goal is to finish its execution within it's deadline. More specifically, for the regular profile jobs, the aim is to minimise the laxity of the job by assigning the job to its preferred fog node (pfn), while finishing the job within it's deadline. For the tagged profile jobs, we need to make a call whether the job needs to be split or not. If yes, the scaling up algorithm is invoked, which splits the jobs among various fog nodes in order to finish within the deadline. Otherwise, the jobs are assigned to the preferred fog node (pfn).

Initially, we divide the job set J into small(J_S), medium (J_M), and large (J_L) jobs. We use the k-means algorithm to partition the jobs into small j_s, medium j_m, and large j_l sizes [25]. We use job duration and memory usage of the jobs as an input data. The k-means clustering is applied by using k = 3, this results in providing the breakpoints to categorise the jobs.

The algorithm RTH^2S works as follows. The input data for the algorithm is the set of jobs. This set consists of jobs of various sizes along with their deadlines. The first step is to populate the set of jobs J into three queues $Q1$, $Q2$, and $Q3$, based on the priority level assignment of Table 3. The queues $Q1$, $Q2$, $Q3$ are sorted in ascending order of deadlines. The rationale behind this sorting is to align it with the Earliest Deadline First algorithm. We form a list named scheduledlist S, which is initially empty. We consider a queue Q_{pj}, which queues the preempted jobs. This queue is initially empty. The tags are assigned to the jobs as per Table 4. The jobs are executed according to their priority levels, i.e., P_1 being the highest priority, and P_3 being the lowest priority. Initially, the jobs present in $Q1$ are scheduled. As soon as the job arrives, we examine its tag. If the job has no tag, i.e., small jobs with tight deadline, then the preferred fog node of the algorithm is estimated. In Preferred-fn(1), 1 stands for fog tier-1. Initially, the ft of the job j^k is calculated for the tier-1 fog nodes. We calculate the minimum finish time ft_{min} among the calculated finish times. We estimate pfn by using equation 4 for j^k. In the next step, we compare two conditions: whether the task's requirement is within the preferred fog node's pfn capacity and whether its laxity is less than zero or not. The latter check implies whether the job j^k is finishing before the deadline or not. If both the conditions are satisfied, then the job j^k is scheduled on the pfn. After this, we calculate the associated Monetary cost MC on the preferred fog node

Algorithm 1: RTH^2S

Input: Set of jobs
Output: Optimal Schedule
Populate $Q1$, $Q2$, and $Q3$ with priority level P_1, P_2, and P_3 respectively;
Sort queue $Q1$, $Q2$, and $Q3$ with ascending order of deadlines;
Assign tags to the jobs;
scheduledlist $S = empty$, $Q_{pj} = empty$;
for *k = 1 to size(Q1)* **do**
  if $tag(j^k) \rightarrow x$ **then**
    Preferred-fn(1);
  end
  if $tag(j^k) \rightarrow tag1$ **then**
    Preempt the currently scheduled jobs and add the jobs to Q_{pj};
    ScaleUp();
    Resume the jobs present in Q_{pj};
  end
end
for *k = 1 to size(Q2)* **do**
  if $tag(j^k) \rightarrow x \;||\; tag(j^k) \rightarrow tag2$ **then**
    Preferred-fn(2);
    if j^k *is unscheduled* **then**
      Preferred-fn(3);
    end
  end
  if $tag(j^k) \rightarrow tag1$ **then**
    Preempt the currently scheduled jobs and add the jobs to Q_{pj};
    ScaleUp();
    Resume the jobs present in Q_{pj};
  end
end
for *k = 1 to size(Q3)* **do**
  if $tag(j^k) \rightarrow x \;||\; tag(j^k) \rightarrow tag2$ **then**
    Preferred-fn(3);
  else
    schedule j^k on cdc;
    estimate the MC using eq.(17);
    remove job j^k from queue Q, add job j^k to scheduledlist S;
  end
end
Calculate $SR(sys) \forall FN, C$

pfn. The job is added to the scheduledlist S. If the tag of the job is $tag1$, then the jobs scheduled on tier-1 are preempted and the ScaleUp algorithm is called. The preempted jobs are added to Q_{pj}. The ScaleUp algorithm works as follows. First, we find the minimum MIPS required for finishing the job before deadline. We form a variable sum which is initialised to zero. We form a loop for the fog nodes at fog node tier-1. The associated value of $Y(fn)$ is estimated using equation 6. If the fog node has spare capacity, then we estimate the sub-job w_i by using equation 7. After this step, we calculate the finish time of sub-job over the selected fog node. Once the job j^k gets the minimal MIPS required for execution, the loop breaks. The overhead for job j^k is estimated. If the job j^k finishes before deadline, then the job j^k is scheduled. The job j^k is removed from queue $Q1$ and added to the scheduled list S. Otherwise, the job j^k can't be submitted to the scheduler. The jobs in Q_{pj} are resumed on the respective fog nodes. After traversing $Q1$, the algorithm goes for P_2 priority. For the incoming job, the tag is seen. If there is no tag or tag2, then preferred-fn is run for fog tier-2. If the job is still unscheduled, then the preferred fog node is examined at tier-3. For the tag1, the preemption at fog node tier-2 is done and ScaleUp algorithm is called. For the last queue i.e $Q3$, the algorithm tries to run the jobs on fog tier-3. If the queue still has some jobs unscheduled, then they are scheduled on the $cdc-only$. Finally, SR for all the jobs is calculated.

Algorithm 2: Preferred-fn(n)

Input: Job j^k with tag $\rightarrow$ x or tag2
Output: pfn
for *y=1 to m* **do**
  estimate ft of job j^k in fn_n^y using eq.(1);
  find pfn with ft_{min} forall ft using eq.(4);
end
if $R(pfn) \geq r(j^k, pfn)$ *and laxity(pfn)* ≤ 0 **then**
  schedule job j^k on preferred fog node fn;
  estimate the MC on pfn using eq.(16);
  add job j^k to scheduledlist S;
end

6 Simulation Results

In this section, we discuss the simulation results that were carried out for the performance evaluation of the proposed algorithm RTH^2S. We consider sample scenarios that align with our Fog Architecture depicted in Fig. 1. The jobs may be run on: tier-1 fog nodes FN_1, tier-2 fog nodes FN_2, tier-3 fog nodes FN_3, or on the cloud data center c_x. In our work, we consider three tiers of fog nodes. The proposed model can be readily extended to support more tiers, based on the application requirements.

The jobs are executed on the basis of the priority assigned. Priority P_1 jobs run on FN_1 nodes, priority P_2 jobs run on FN_2 or on FN_3 nodes, priority P_3 jobs run on FN_3 nodes, or on the cloud data center c_x. This ensures that the utilization of all nodes is maximised. We compare our proposed scheduling algorithm RTH^2S with $cdc-only$ and a scheduling algorithm for Heterogeneous Fog Computing Architectures proposed in [15]. In $cdc-only$, the fog nodes have not been considered in executing jobs i.e. only the cloud data center c_x is used for executing all the jobs. In

Algorithm 3: ScaleUp

Input: Job j^k with tag $\rightarrow$ tag1
Output: Optimal Schedule
Calculate min. MIPS for job j^k to finish before deadline;
$sum \leftarrow 0$;
for *p=1 to m* **do**
  Get MIPS of p^{th} fog node;
  sum = sum + MIPS(p^{th});
  Estimate Y on p^{th} fog node using eq.(6);
  if *equation (5) holds true* **then**
    Calculate sub-job w_i on fog nodes using eq.(7);
  end
  Estimate $ft(w_i)$ on p^{th} fog node;
  if $sum \geq minimalMIPS$ **then**
    break;
  end
end
Estimate $\Theta(j^k)$ using equation (9);
if $\Theta(j^k) \leq deadline$ **then**
  schedule the job j^k on the fog nodes;
  estimate the MC on fog nodes using equation (16);
  add job j^k to scheduledlist S;
else
  job j^k can't be submitted;
end

[15], the authors propose the LTF (Longest Time First) scheduling algorithm for heterogeneous fog networks. We compare our proposed algorithm RTH^2S with LTF. The LTF algorithm schedules the jobs with the longest execution time to the fastest node. Prior to execution, LTF sorts the jobs in a descending order based on their deadlines.

6.1 Workload

We have used a real workload called HPC2N (High Performance Computing Center North) [23], [12]. This is a joint operation between various facilities and educational institutes. This workload is a result of about 3.5 years of activity. This activity was carried on the Seth cluster of the HPC center in Sweden. The Linux cluster consists of 120 dual CPU nodes. Each node in the cluster consists of 2 AMD Athlon MP2000+ CPUs, with a clock frequency of 1.67 GigaHertz. The peak performance of this cluster is 800 Gigaflops. Each node has access of 1 GB of RAM, which is shared by both CPUs. The communication framework consists of a 3D SCI intern-connect and fast Ethernet. This workload consists of over 5,00,000 jobs, which are of various lengths, and is suited to cloud, grid and fog computing. Each task has various parameters associated with it – such as Job ID, burst time (τ), memory usage, and arrival time. For each job, we take the arrival time as 0. We have divided the jobs into three categories as per the job length by using k-means: small, medium and large. The range of job lengths considered for each category are as follows - small: 1-95, medium: 96-205, large: 206-400. The fog network consists of 8 FN_1 nodes, 4 FN_2 nodes, 1 FN_3 node and 1 cdc c_x. The propagation delay (pd) from a user U_i to a tier-1 fog node is 2 milliseconds, user U_i to a tier-2 fog node is 6 milliseconds, user U_i to a tier-3 fog node is 12 milliseconds, and from U_i to a cdc is 137 milliseconds (12 milliseconds from the U_i to the proxy server and 125 milliseconds from the proxy server to cdc). The capacity of each fog node present at tier-

1, $c(fn_1^y)$ varies from 1000 MIPS to 2000 MIPS. Likewise, the capacity of each fog node present at tier-2, $c(fn_2^y)$ varies from 2500 MIPS to 4000 MIPS, the capacity of each fog node present at tier-3, $c(fn_3^y)$ has been taken as 5800 MIPS, and the capacity of the cdc, $c(c_x)$ has been taken as 70000 MIPS. The number of jobs (i.e. Job Set JS) varies from 250 to 500 and the execution costs of these jobs (i.e. τ) varies from 100 to 8500 MIPS. The size-wise break up of the jobs is as follows: small jobs make up 41% of the workload, medium jobs make up 34% of the workload, and large jobs make up 25% of the workload. Note that the values in Table 2 are representative values, and they can be changed, based on the user requirements, without having an effect on the working of the RTH^2S algorithm.

6.2 Simulation Setup and Parameters

We have used the iFogSim [3] simulator for the implementation of our proposed algorithm RTH^2S. iFogSim is rooted in CloudSim – a very widely used discrete event cloud simulator. iFogSim, therefore, allows us to model the characteristics of a cloud platform more realistically (CloudSim has >4K downloads) [32], a key basis for some of the simulation that this work is based on. We have modelled various features of fog nodes and the cdc in this simulator. By using iFogSim, one can evaluate different fog and cloud scheduling strategies. This simulator is appropriate for fog enabled devices, as it follows a representation of the sensor $\rightarrow$ processor $\rightarrow$ actuator model. A class named $HierarchicalFog$ has been implemented in the simulator. This class reads the dataset from a text file and stores the job-id, the job-length, the deadline, and the priority. In addition to this, the following quantities have also been added to the class : the propagation delay (pd) of all FN and C, execution capacity (c) and the module allocation. A $FogDevice$ class present in iFogSim contains a function named $updateAllocatedMips$. The task of this function is to allocate the MIPS requirements of various execution modules. In order to take job deadlines into account, certain modifications have been made to this class. We have created job queues $Q1$, $Q2$, and $Q3$ in the simulator. The queues are sorted in increasing of the deadlines, e.g. the task with the tightest deadline appears at the head of the queue. As per the priority assignment of Table 3, jobs are be allocated to tier-1 FN_1, tier-2 FN_2, tier-3 FN_3, and cdc c_x. Note that each data point is an average of five simulation runs. A 95% confidence interval is used in the graphs. We now describe the parameters used in our simulations:

1) Success Ratio (SR): This is defined as $(\frac{N'}{N})$*100, i.e. the percentage of the number of jobs finishing execution before their deadlines to the total number of jobs considered for scheduling.

2) Task Load (TL): There is a MIPS requirement associated with all jobs considered for scheduling. The MIPS value of each job was uniformly selected from the range (100, 8500). Next, we calculated the average MIPS value for all jobs. In order to get a range of Task loads, this MIPS value is multiplied by 1 to 5.

3) Propagation Delay (PD) : This quantity is defined as the range of delay factor between the jobs the fog nodes and the cloud data centers (cdc). A lower value indicates smaller delay. We set the delay factor (pd) of 2, 6 and 12 milliseconds between the user and fog tier-1, tier-2, and tier-3 respectively. A value of 137 milliseconds was set between the user and the cdc. These values are added by 10 milliseconds in each iteration to get new values.

4) Deadline Factor (DF) : Job deadlines are changed over a range to observe the effect of tight and loose deadlines on performance. A higher value implies tight deadlines, and vice versa. A job's initial deadline is considered. Next, we calculate the average of all such deadlines. To get a range of deadline values, we divide this average deadline with a factor of 1 to 5. The tight deadlines lie in the range 1-24, moderate deadlines lie in the range 25-74, and loose deadlines lie in the range 75+.

5) Heterogeneity Level (HL) : Heterogeneity Level (HL) signifies the degree of heterogeneity of fog nodes – measuring the variation in computational capacity of fog nodes within each level. A low HL value implies that the execution capacities of the fog nodes are similar. The Heterogeneity level of any n^{th} tier fog node is given by:

$$HL_{FN_n} = \frac{c(fn_n^{max}) - c(fn_n^{min})}{average(c(fn_n^j))} \tag{22}$$

$c(fn_n^{max})$ represents a tier-n fog node with the maximum capacity.

$$fn_n^{max} = fn_n^j : c(fn_n^j) > c(fn_n^X) \tag{23}$$

In eq. (23), $fn_n^j, fn_n^X \in FN_n$ && $X \neq j$. $c(fn_n^{min})$ represents a tier-n fog node with the minimum capacity.

$$fn_n^{min} = fn_n^j : c(fn_n^j) < c(fn_n^X) \tag{24}$$

In eq.(24), $fn_n^j, fn_n^X \in FN_n$ && $X \neq j$. We can replace n in FN_n to get the heterogeneity level of a fog node. Finally, the heterogeneity level of the system is given by:

$$HL_{system} = HL_{FN_1} + HL_{FN_2} + ... + HL_{FN_n} + HL_C \tag{25}$$

6) Monetary Cost (MC) : This quantity is defined as the cost associated with executing the job on fog nodes FN, or on cloud data center cdc. This metric depends upon the execution cost and propagation delay of j^k on fn.

6.3 Results and Discussion

In this section we describe results of various experiments to evaluate and compare our approach across both real world & synthentic datasets.

Effect of fog resources on Performance: We evaluate the capacity improvement of using fog nodes with the cloud data center cdc, using Success Ratio(SR) as the performance metric. Scheduling algorithms: RTH^2S, $1TF$ (1-tier fog), $2TF$ (2-tier fog), $cdc-only$, LTF [15], and WALL [28] are compared. In $cdc-only$, we forward all the jobs to the cloud data center for execution. In RTH^2S, the number of fog nodes at tier-1, tier-2 and tier-3 has been fixed at 8, 4 and 1 respectively. We assume one cdc.

LTF considers two kinds of fog nodes: fast and slow. In order to achieve parity, we consider one fast node and two slow nodes in this section of the simulation. The computation power of the fast nodes is more than that of the

slow nodes. However, the fast nodes consume more power. Our algorithm RTH^2S dispatches the jobs in an increasing order of deadlines. On the other hand, LTF sorts the jobs in decreasing order of deadlines and sends the jobs with the largest execution time to the fastest node. The Workload ALLocation algorithm, $WALL$ is a hierarchical cloudlet network that assigns jobs to suitable cloudlets/fog nodes, such that the average response time is minimized. It sorts the users based on decreasing workload size and schedules the jobs to the cloudlet so as to minimise the response time. We also consider various fog node tiers in our simulation. The propagation delay (pd) from a user U_i to cdc c_x has been fixed at 125 milliseconds. In 1-tier fog i.e. $1TF$, we consider only one tier of fog nodes (FN_1) and a cloud data center (c_x). The $c(fn_1^y)$ of tier-1 fog nodes has been fixed at 3500 MIPS. The propagation delay (pd) from a user U_i to tier-1 FN_1 in $1TF$ has been fixed at 2 milliseconds. In 2-tier fog i.e. $2TF$, we consider two tiers of fog nodes (FN_1), (FN_2) and a cloud data center (c_x). The propagation delay (pd) from an user U_i to tier-1 FN_1, and from user U_i to tier-2 FN_2 has been fixed at 2 milliseconds, and 6 milliseconds respectively. The propagation delay (pd) from an user U_i to tier-1 FN_1, from user U_i to tier-2 FN_2, and from user U_i to tier-3 FN_3 in $3TF$ has been fixed at 2 milliseconds, 6 milliseconds, and 12 milliseconds respectively.

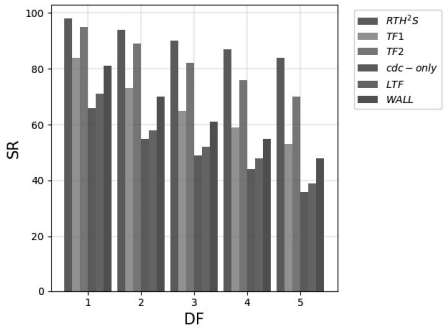

Fig. 3. Effect of DF on SR

In the first simulation scenario, we increase the Deadline Factor (DF), and observe it's impact on the Success Ratio (SR). The delay factor (pd) is taken as 2, 6, and 12 milliseconds between the user and fog tier-1, tier-2, and tier-3 respectively. A value of 125 milliseconds is taken between the user and the cdc in RTH^2S. The deadline factor (DF) of the jobs has been varied from 1 to 5. Tasks have the loosest deadlines when DF = 1, and the tightest deadlines when DF = 5. The results of this simulation are shown in Fig. 3. It is observed that as we increase the DF value, deadlines become more "tight", and we notice a complementary decrease in the SR value for all scheduling algorithms. As such, a large number of jobs are unable to finish their execution before their deadlines, which reflects in the decreased SR.

Our proposed algorithm RTH^2S provides a higher SR values than the $cdc - only$ algorithm. RTH^2S schedules the jobs as per their size, priority and deadlines. On the other hand, in case of the $cdc - only$ algorithm, all jobs, irrespective of their size and priority are forwarded to the cloud data center (c_x) for execution. This has an adverse effect on both the tight or moderate deadline jobs, due to the large propagation delay between the user U_i and cloud data center (c_x). Hence, the SR values of our proposed algorithm RTH^2S are higher than those offered by $cdc - only$. The LTF algorithm sorts the jobs from longest to shortest, and then assigns the long jobs to the fast nodes and the short jobs to the slow nodes. Hence, short jobs will be executed at the end, due to which they may have already missed their deadlines. The large jobs execute at the fast nodes, whereas the medium and small jobs execute at the slow nodes. Due to the modest power of the slow nodes, a small number of jobs are accommodated at the lower level. Hence, LTF offers a smaller Success Ratio SR value than RTH^2S. In $2TF$, jobs with P_1 priority are executed at tier-1, jobs with P_2 priority are executed at tier-2, and jobs with P_3 priority may get execute on the cloud data center c_x, depending on the size and deadline requirements. In the absence of the third tier, more jobs are sent to the cloud for execution, which leads to smaller SR values for $2TF$. On the other hand, $1TF$ offer low SR values, despite having reduced propagation delay between U_i and fn_1^y. This happens due to the reduced overall computation capacity of tier-1 fog nodes, which results in transferring more jobs to the cloud data center c_x. The $WALL$ algorithm takes the user with a maximum job size and assigns it to the fog node that offers the minimum response time. This approach negatively affects small and medium jobs with tight deadlines. Also, the computation power of fog nodes is relatively modest for executing large jobs, which further increases the finish times, leading to deadline misses. In RTH^2S, we split the large jobs to complete jobs within the deadlines. The SR ratio provided by different algorithms is as follows : RTH^2S >2TF>1TF> $WALL > LTF > cdc - only$.

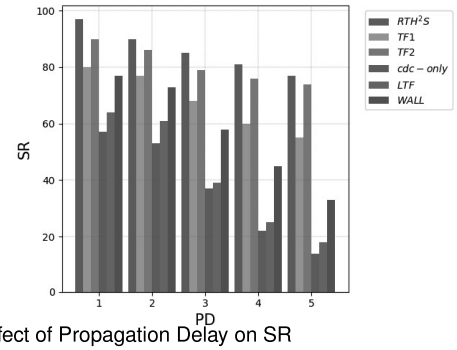

Fig. 4. Effect of Propagation Delay on SR

In the second simulation, we examine the impact of Propagation Delay (pd) on the SR. The initial pd from the users to fog nodes has been fixed as follows: 2 milliseconds to tier-1, 6 milliseconds to tier-2, 12 milliseconds to tier-3 and 125 milliseconds to the cdc. In order to increase this delay, we have added 10 milliseconds at tier-1, tier-2, tier-3 and cdc in each iteration. Fig. 4 depicts the results. It is observed that with an increase of the pd value, more time is spend in communication. This results in an increase in the commencement time (ct) at the fog nodes present at tier-1 (FN_1), tier-2 (FN_2), tier-3 FN_3 and at the cloud data center (c_x). Hence, the finish time (ft) of the jobs often overshoots their deadlines (d), so a lesser number of jobs finish execution before their deadlines, which results in a low Success Ratio SR in all tiers. We observe similar results in LTF. The induced delay between slow and fast fog nodes results in smaller values for SR. Likewise, the increased pd effects the SR in $WALL$. The pd added at each iteration

increases the completion time of the jobs in both tiers and cdc. Overall, we observe that an increase in the pd reduces the SR in all six scheduling strategies.

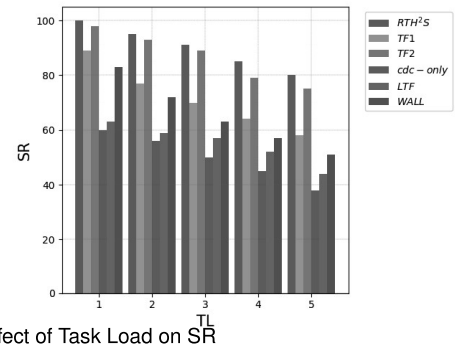

Fig. 5. Effect of Task Load on SR

In the next simulation, we show the impact of Task load (TL) on Success Ratio (SR). Fig. 5 depicts the results for this simulation. We increase the task load (TL) from 1 to 5. As we increase the TL value, more tasks are added to the system. This results in reducing the SR, as a large number of jobs start missing their deadlines. This behaviour is shown by all six scheduling strategies: RTH^2S, $1TF$, $2TF$, $cdc{-}only$, LTF, and $WALL$. However, RTH^2S takes advantage of the fog nodes present at tier-1, tier-2, and tier-3 due to which, a larger number of jobs are able to meet their deadlines. Note that these jobs are unable to meet their deadlines on $cdc - only$. This happens as the fog nodes are in closer proximity to the end users, and hence, the propagation delay (pd) from user to fog nodes is less. Contrarily, jobs which are using cdc to execute face significant propagation delays (pd), which results in deadline misses. The LTF algorithm sorts jobs in a decreasing order of deadlines. It's SR values are lower than those of the proposed algorithm's SR values, as we sort in the opposite order: small deadline $\rightarrow$ large deadline. Hence, a larger number of jobs are able to meet their deadlines in a given time interval. The $WALL$ algorithm sorts jobs in descending order of sizes, which effects the tight/moderate deadlines of small and medium jobs. For $1TF$ node, due to less computation power, these fog nodes are not able to finish the jobs before the deadlines. It is tough for a single tier to finish the P_1 or P_2 priority jobs before their deadlines. On the other hand, in $2TF$, due to addition of one more tier, more number of jobs can be executed before the deadlines. However, once the tiers don't have sufficient capacity to execute, the jobs are transferred to cloud data center $cdc\ c_x$. Due to the significant propagation delay between a user and the cloud data center, the jobs start missing their deadlines. For 3-tier fog node i.e RTH^2S, more jobs can be accommodated on the fog node tiers with less propagation delays which leads to higher success ratios SR. It is important to note that as we add fog node tiers, there is an addition of fog nodes in the network, leading to an increase in the total computation power. Though, the propagation delay increases as well, but this delay is smaller as compared to sending jobs to the cloud.

We have observed that the 3-tier fog based algorithm RTH^2S outperforms all compared scheduling strategies, for all metrics considered. The $2TF$ network and $1TF$ network offer lesser computation power. Though, there is an increased communication delay due to the presence of more fog tiers in RTH^2S, this delay is smaller as compared to sending jobs to the cloud data center for execution. Our proposed algorithm outperforms $cdc - only$ owing to the large communication delay involved in sending the jobs to $cdc - only$. It outperforms LTF due to their sorting of jobs in an opposite direction, which leads to small jobs being scheduled too late. RTH^2S outperforms $WALL$ as it provides the splits of the large jobs with a tight deadline rather than assigning them as a whole to the fog node, which increases the finish time of the jobs. Also, $WALL$ selects the users with the maximum job size first, giving less priority to small/medium jobs with tight deadlines.

Effect of Heterogeneity Level (HL) on Success Ratio (SR): We examine the impact of fog node heterogeneity on the system performance. The results of this simulation are shown in Fig. 6. We increase the Heterogeneity Level HL from 0 to 1.2. The number of fog nodes at tier-1, tier-2, and tier-3 have been fixed at 8, 2 and 1 respectively. The capacity of fog nodes has been varied from 300 MIPS to 6000 MIPS. We compare the performance of six scheduling algorithms: RTH^2S, $1TF$, $2TF$, LTF, $WALL$ and $cdc - only$. As $cdc - only$ does not employ fog nodes, a significant number of jobs miss their deadlines. Moreover, we observe a constant Success Ratio for $cdc - only$, i.e. increasing fog node heterogeneity has no effect on $cdc - only$'s SR. This is because we consider only 1 cdc in our approach, so there is no heterogeneity in the cdc. Our proposed model can be easily extended to consider heterogeneity in the cdc. We omit this experiment due to space constraints. For RTH^2S, we observe that increasing HL leads to an increase in the SR. This is because increasing HL increases the variation in the execution capacity of fog nodes. Hence, the probability of picking a faster fog node increases. This behaviour is observed in $1TF$, $2TF$, $WALL$ and LTF.

However, we observe that RTH^2S offers a higher SR than LTF. This is because LTF orders the jobs from longest $\rightarrow$ shortest, i.e. it is non real-time. Hence, the short jobs start late, and miss their deadlines. On the other hand, RTH^2S sorts jobs from smallest $\rightarrow$ largest, in terms of deadlines. So, the number of jobs meeting deadlines is maximised. $WALL$ performs better than LTF, as $WALL$ has two tiers, while LTF has just one-tier. Hence, RTH^2S performs better than $1TF$, $2TF$, $cdc - only$, $WALL$ and LTF as it provides higher SR values.

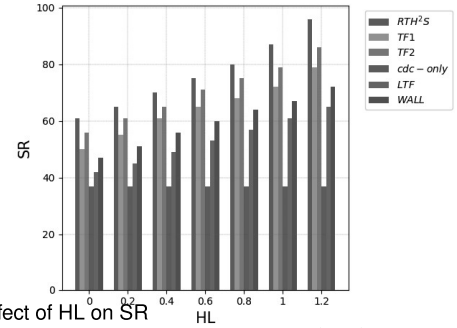

Fig. 6. Effect of HL on SR

Effect of Tag mix on Success Ratio (SR): we study the impact of tag assignment on the success ratio SR of RTH^2S, $WALL$, LTF, and $cdc - only$. We consider two separate tag mixes in this simulation: **Tag Mix 1:** the number of untagged i.e. regular profiles & $tag2$ jobs are constant, and the number of $tag1$ jobs are periodically injected by $1/4$ at every

x-axis data point. Based on Table 4, all three types of $tag1$ jobs are increased in equal proportion by $1/12$. **Tag Mix 2:** the number of $tag1$ & un-tagged jobs are constant, and the number of $tag2$ jobs are periodically injected by $1/4$ at every x-axis data point. Based on Table 4, all three types of $tag2$ jobs are increased in equal proportion by $1/12$.

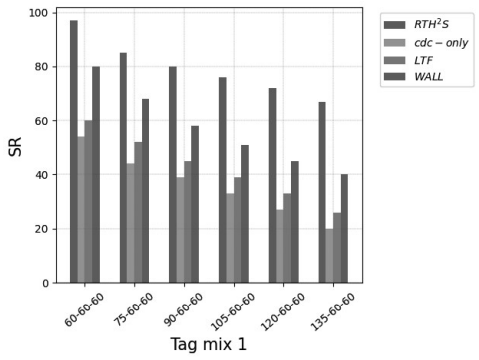

Fig. 7. Effect of tag mix 1 on SR

Initially, we considered 160 jobs. The results for tag mix 1 & mix 2 are shown in Fig. 7 and Fig. 8 respectively. The format of each x-axis data point is as follows: (# of $tag1$ jobs, # of $tag2$ jobs, # of no tag jobs). Fig. 7 shows the result of tag mix 1. Increasing larger & medium jobs ($tag1$) with tight deadlines, and large jobs with moderate deadlines, we are increasing the load on the lower tiers of fog nodes. Due to an increase in tag1 jobs, the algorithm preempts the currently scheduled jobs. This negatively impacts regular profile jobs: small jobs with tight deadlines, or medium jobs with moderate deadlines. As large size tasks cannot be directly accommodated on a single fog node, they need to be split before scheduling on fog nodes. This increases the commencement time of the jobs which may lead to a deadline miss. This decreases overall success ratio SR of RTH^2S. The minimum SR is exhibited by the $cdc-only$ owing to the distance between user U_i and cloud data center c_x. Also, the cloud data center is not suitable for handling jobs with tight deadlines. On the other hand, LTF sorts the job in decreasing order of deadlines which results in missing most of the tight deadlines. $WALL$ sorts the users in decreasing order of workloads. Moreover, it doesn't do any splitting of the jobs. Due to the modest capacity of fog nodes, it takes more time to finish the large jobs. This results in missing most of the job deadlines.

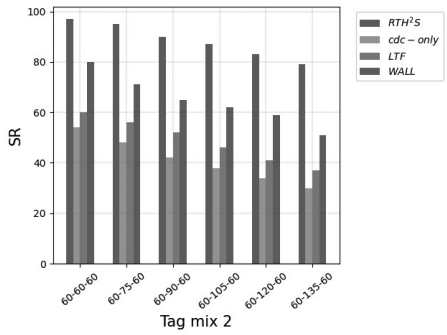

Fig. 8. Effect of tag mix 2 on SR

Fig. 8 shows the result of tag mix2, where we increase $tag2$ jobs, i.e., small jobs with moderate deadlines, small jobs with loose deadline and medium jobs with moderate deadline. This results in decreasing the performance of the RTH^2S algorithm due to an increase in the number of jobs. However, in this case, no job preemption is necessary as job sizes are not so large and deadlines are not so tight. The $cdc-only$ algorithm performs the worst as it does not employ any fog node tier for offloading the computation. On the other hand, LTF executes large jobs first by employing fast fog nodes for job execution. This results in missing jobs of small and moderate sizes. We observe a similar pattern in $WALL$, as it gives preference to large jobs, resulting in the performance degradation due to misses in tight deadlines of small and medium jobs. We observe that as we increase the number of tag1 jobs, there is significant decrease in the SR values. This happens because tagged jobs with tight/moderate deadlines and large/medium sizes lead to regular profile jobs being unable to execute before their deadlines.

TABLE 5
Cost ($/hour, May 2021, Asia Pacific Region) of Microsoft Azure

Instance type	Cost per hour($)
tier-1	$0.034
tier-2	$0.34
tier-3	$3.4
cdc	$0.08

Effect of task load on monetary cost (MC): We consider task load on monetary cost MC for RTH^2S using Microsoft Azure pricing in our simulations, as shown in Table 5, with results in Table 6. We have taken weights w_1 and w_2 as 0.75 and 0.25, respectively. As we increase the task load of the system, we observe an increase in the monetary cost. As more jobs are added to the system, more work has to be done by the FN and cdc. With the increase in the task load, the jobs are sent to the higher tiers for execution. The price of higher-tier fog nodes is more than the lower-tier fog nodes. Also, this increases the propagation delay in the system. As monetary cost is directly proportional to the execution cost and propagation delay of jobs, this is reflected in the results of RTH^2S, $WALL$ and LTF. The monetary cost of LTF is more than RTH^2S as LTF sends jobs with the largest execution time to the fastest node. The monetary cost of $WALL$ is less than LTF, as LTF sends more jobs to cdc as task load increases. This increases the propagation delay, which increases the overall monetary cost of LTF. The monetary cost of $WALL$ is higher than RTH^2S, as $WALL$ prefers the larger workloads initially. This algorithm can finish a very number of small and medium jobs.

TABLE 6
Effect of task load on monetary cost

TL	RTH^2S (TL↑)	$WALL$ (TL↑)	LTF (TL↑)
1	$1.02	$2.41	$3.29
2	$2.28	$4.02	$5.12
3	$3.53	$5.69	$7.93
4	$5.31	$8.23	$9.89
5	$7.31	$10.72	$13.23

Task deadline and monetary cost (MC): We investigate the effect of deadline factor on monetary cost MC for the proposed algorithm RTH^2S, $WALL$ and LTF. The results are shown in Table 7. With the increase in the deadline factor DF, the deadline becomes more tight and the jobs have to run in the fog tiers to finish their execution before

deadlines. As more and more jobs run on the fog tiers, the monetary cost MC increases. Besides, as deadline become tighter with increase in deadline factor, less number of jobs are able to finish with cdc. This happens as jobs have to travel farther to execute on the cdc. By the time the tight deadline jobs reach the cdc, it is already too late. RTH^2S utilises the fog tier's resources to finish the job execution before the deadlines. This further increases the overall MC of our proposed algorithm RTH^2S. RTH^2S outperforms LTF and $WALL$ due to the usage of better heuristics. Both LTF and $WALL$ execute large jobs first, and the execution cost of large jobs is higher than small and medium jobs. With deadlines becoming tighter, both algorithms run a significant portion of large jobs, leading to higher monetary costs.

TABLE 7
Effect of deadline factor on monetary cost

DF	RTH^2S (DF↑)	$WALL$ (DF↑)	LTF (DF↑)
1	$1.13	$1.91	$2.98
2	$1.5	$2.53	$4.08
3	$2.33	$3.45	$5.91
4	$4.98	$6.43	$7.18
5	$6.05	$8.49	$10.23

Task Success Ratio (SR) and queuing delay: We investigate the effect of task load on system performance, while considering queuing delay. In Fig. 9, we compare the performance of three scheduling algorithms: RTH^2S, $WALL$, and LTF, by increasing task load TL from 1 to 5. We calculate the finish time of the jobs with and without queuing delay (q). Increasing queuing delay leads to jobs being unable to meet their deadlines, and therefore a decreasing system success ratio SR proportional to the number of jobs. RTH^2S offers better performance, as it considers job priority, size and deadline. On the other hand, LTF sorts jobs from largest to smallest, leading to higher deadline misses. $WALL$ chooses a user having the largest job among all the users – leading to short jobs missing most of the deadlines.

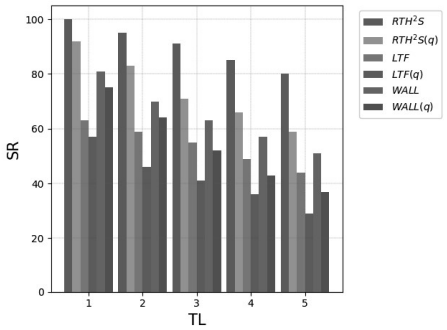

Fig. 9. Effect of queuing delay on SR

6.4 Performance analysis using synthesized dataset

We consider synthesized datasets to evaluate the performance of our proposed algorithm RTH^2S, LTF, $WALL$, and $cdc-only$. Our dataset comprises [100-300] jobs in job set JS. We randomly generate the job between 2000–45000 MI, with memory usge between 0.15GB–2.5GB and Deadline range of 250ms–10000ms, based on [39], [40]

DF and Success Ratio (SR): We consider the impact on SR while increasing DF from 1 to 5, higher values of DF makes it difficult for jobs to finish execution within their deadlines. From Fig. 10 we observe a similar trend in the performance of RTH^2S, $cdc-only$, LTF, and $WALL$.

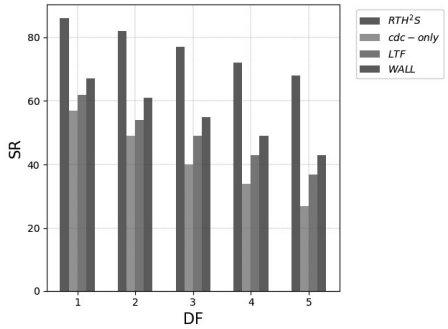

Fig. 10. Synthetic dataset: Effect of DF on SR

Propagation delay (pd) and Success Ratio (SR): We illustrate the impact of pd on SR. We take the delay factor (pd) as 2, 6, and 12 milliseconds between the user and fog tier-1, tier-2, and tier-3, respectively. We consider a value of 125 milliseconds between the user and the cdc. As we increase the delay factor, we see a decrease in the success ratio for all the jobs in all four scheduling strategies: RTH^2S, LTF, $WALL$, and $cdc-only$. This happens as the jobs' finish time increases with the increase in the pd. This is visible in the results shown in Fig. 11. Due to the reasons mentioned in the previous sections, we observe the following SR among the algorithms: $RTH^2S > WALL > LTF > cdc-only$.

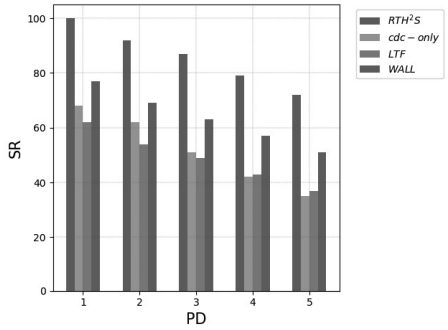

Fig. 11. Synthetic dataset: Effect of pd on SR

6.5 Fog cloud test-bed

Our prototype considers a single tier of fog nodes FN_1 followed by the cloud data center c_x. Tier-1 FN_1 consists of two heterogeneous fog nodes. We used a RPi4 Model B with 4GB RAM, and a desktop with Ubuntu 16.04 operating system as fn_1^1, and fn_1^2 respectively. We used a VM instance on Amazon EC2 as c_x. The first fog node is a RPi4 (Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC 1.5GHz running raspbian OS). The second fog node is an Intel i7-7700 CPU, 3.60GHz × 8, 64-bit OS with 7.7GiB RAM. We used a cloud VM instance with 1 vCPU, 1 GiB memory as c_x. A regular IPv4 Internet connection is used to connect the user to the cloud. We do not show the results for the WALL algorithm, as its performance is similar to LTF, due to the consideration of 1 tier of fog nodes.

Effect of TL on Success Ratio SR: In the first experiment, we study the real time performance of proposed algorithm RTH^2S, LTF and c_x. Three kinds of job priorities are considered i.e. $P1$, $P2$, $P3$. Initially, we consider eight jobs. We gradually increase the number of jobs (upto 48), and observe the impact on the Success Ratio SR. The results

are shown in Fig. 12. The SR decreases with the increase of jobs in all three cases. However, our proposed algorithm RTH^2S outperforms the others due to the usage of superior heuristic. It incorporates fog nodes in job execution, while following the earliest deadline first EDF algorithm. Moreover, the jobs are assigned according to their priority. The $cdc-only$ performs worst owing to the significant propagation delay from user u_i to c_x.

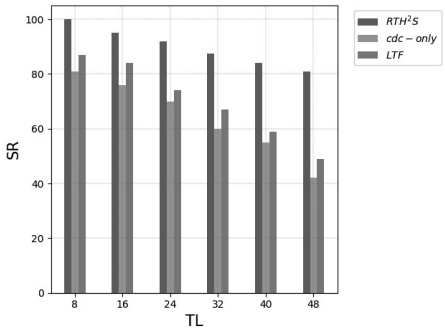

Fig. 12. Effect of TL on SR

Effect of DF on Success Ratio SR: The deadline factor for the tight, moderate and large deadline is 0s, 2s, and 10s respectively. We increase all three deadlines by 1s in each iteration. As shown in Fig. 13, the best performance is offered by RTH^2S. With a loose deadline, more jobs are able to finish their execution leading to an increase in the SR for all three approaches.

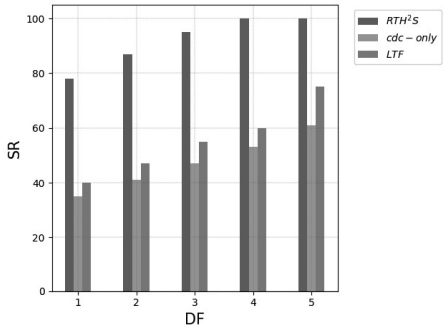

Fig. 13. Effect of DF on SR

Effect of TL on Average Response Time: We estimate the average response time by varying task load for RTH^2S, LTF and $cdc-only$. Initially, we consider 7 jobs, followed by an increase of 5 per iteration. The average response time is the sum of execution time & communication delay. With the increase in the number of jobs, the average response time increases. The results of this experiment are shown in Fig. 14. The highest average response time is provided by $cdc-only$ owing to the high communication delay from user U_i to c_x. The lowest average response time is exhibited by RTH^2S followed by LTF, as LTF sorts the deadlines in descending order leading to short deadline jobs executing last. This also increases the average response time of the all jobs. The average improvement in Success Ratio (SR) offered by RTH^2S over $cdc-only$ and LTF is shown in Table 8.

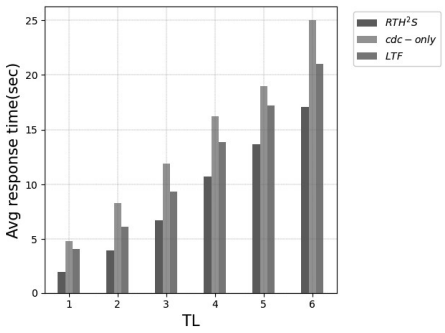

Fig. 14. Effect of TL on Average Response Time

TABLE 8
Average Improvement

Performance metric	$cdc-only$	LTF	$WALL$
Deadline Factor (DF)	81%	69%	46%
Propagation Delay (PD)	134%	107%	39%
Heterogeneity Level (HL)	106%	43.5%	29%
Task Load (TL)	81%	64%	42%

7 Conclusion

Significant propagation delays between users and the cloud data center may act as a deterrent for executing deadline driven real-time jobs. This delay can be reduced by employing fog nodes for the execution of such jobs. In addition, it may very well be the case that there is a hierarchy of fog nodes [2]. Typically, fog nodes in various tiers (and even within a particular tier) are heterogeneous. In this paper, we propose RTH^2S, an algorithm that schedules real-time jobs on a multi-tiered fog network by taking diverse job profiles into account. Using a real-life workload, RTH^2S is validated using a simulator as well as a prototype. We observe that RTH^2S offers better real-time results in terms of higher Success Ratios, and reduced Monetary Costs. We also observe that job profiles impact the real-time system performance. An increase in number $tag1$ profile jobs impact the regular profile jobs, leading to deadline misses and lower SR values. Our future work involves the use of multiple cloud data centers. We also plan to develop "schedulability" and performance bounds for real-time tasks on such multi-tier fog-cloud architectures.

References

[1] Fog Computing and the internet of things: Extend the Cloud to where the Things are, Cisco White Paper, 2015.
[2] openfogconsortium.org, OpenFog Reference Architecture for Fog Computing, 2017, [online], available at: https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf.
[3] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in Internet of things, edge and fog computing environments", Available: http://arxi.org/abs/1606.02007.
[4] R. K. Naha, S. Garg, D. Georgakopoulos, P. R. Jayaraman, Y. Xiang and R. Ranjan, "Fog computing: survey of trends, architectures, requirements, and research directions", IEEE Access, vol, 6, pp. 47980-48009, 2018.
[5] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", in IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, Oct - Dec, 2009.
[6] N. Auluck, O. Rana, S. Nepal, A. Jones and A. Singh, "Scheduling Real Time Security Aware tasks in Fog Networks", IEEE Trans. on Services Computing, 2019.

[7] S. Han and H. Park, "Predictability of Least Laxity First Scheduling Algorithm on Multiprocessor Real-Time Systems", Int. Conf. on Embedded & Ubiquitous Computing (EUC), Seoul, South Korea, 2006, pp. 755 - 764.
[8] Y. Yang, K. Wang, G. Zhang, X. Chen, X. Luo, M. T. Zhou, "MEETS: Maximal energy efficient task scheduling in homogeneous fog networks", IEEE Internet of Things Journal, vol.5, no. 5, 2018, pp. 4076 - 4087.
[9] K. Fizza, N. Auluck, A. Azim, "Improving the Schedulability of Real-Time Tasks using Fog Computing", IEEE Trans. on Services Computing, 2019.
[10] Y. Yang, S. Zhao, W. Zhang, Y. Chen, X. Luo, J. Wang, "DEBTS: Delay energy balanced task scheduling in homogeneous fog networks", IEEE Internet of Things Journal, vol. 5, no. 3, 2018, pp. 2094 - 2106.
[11] A. Singh, N. Auluck, O. Rana, A. Jones, S. Nepal, "RT-SANE: Real Time Security Aware Scheduling on the Network Edge", The Tenth IEEE/ACM Int. Conf. on Utility & Cloud Computing, Austin, USA, 2017.
[12] I.A. Moschakis and H.D. Karatza, "A meta-heuristic optimization approach to the scheduling of Bag-of-Tasks applications on heterogeneous Clouds with multi-level arrivals and critical jobs", Simulation Modelling Practice &d Theory 57, 2015, pp. 1 - 25.
[13] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow and P. A. Polakos, "A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges", in IEEE Communications Surveys & Tutorials, vol. 20, no. 1, 2018, pp. 416 - 464.
[14] K. Fizza, N. Auluck, O. Rana, and L. Bittencourt, "PASHE: Privacy aware scheduling in a heterogeneous fog environment", in IEEE 6th Int. Conf. on Future Internet of Things and Cloud (FiCloud), 2018, pp. 333 – 340.
[15] H. Wu and C. Lee, "Energy Efficient Scheduling for Heterogeneous Fog Computing Architectures", IEEE 42nd Annual Computer Software and Applications Conference, Tokyo, 2018.
[16] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the Internet of Things", in Proc. IEEE Int. Conf. Edge Comput., Honolulu, USA, 2017.
[17] G. Zhang, F. Shen, Y. Zhang, R. Yang, Y. Yang and E. A. Jorswieck, "Delay Minimized Task Scheduling in Fog-Enabled IoT Networks", 10th Int. Conf. on Wireless Communications and Signal Processing (WCSP), Hangzhou, 2018, pp. 1 - 6.
[18] N. Chen, Y. Yang, T. Zhang, M. Zhou, X. Luo and J. K. Zao, "Fog as a Service Technology", in IEEE Communications Magazine, vol. 56, no. 11, November 2018, pp. 95 - 101.
[19] S. Zhao, Y. Yang, Z. Shao, X. Yang, H. Qian and C. Wang, "FEMOS: Fog-Enabled Multi-tier Operations Scheduling in Dynamic Wireless Networks", in IEEE Internet of Things Journal, vol. 5, no. 2, April 2018, pp. 1169 - 1183.
[20] S. Sarkar, S. Chatterjee and S. Misra, "Assessment of the Suitability of Fog Computing in the Context of Internet of Things", in IEEE Trans. on Cloud Computing, vol. 6, no. 1, 2018.
[21] A.-C. Pang, W.-H. Chung, T.-C. Chiu, and J. Zhang, "Latency-driven cooperative task computing in multi-user fog-radio access networks", in Proc. IEEE Thirty-Seventh Int. Conf. on Distributed Computing Systems (ICDCS), 2017, pp. 615 – 624.
[22] Malik, S., Ahmad, S., Kim, B.W., Park, D.H., Kim, D., "Hybrid Inference Based Scheduling Mechanism for Efficient Real Time Task and Resource Management in Smart Cars for Safe Driving". Electronics 2019, 8, 344.
[23] N'takpé T., Suter F. "Don't Hurry Be Happy: A Deadline-Based Backfilling Approach", Job Scheduling Strategies for Parallel Processing. LNCS10773, Springer.
[24] Tong, L., Li, Y. and Gao, W., 2016, April. A hierarchical edge cloud architecture for mobile computing. In IEEE INFOCOM 2016-The 35th Annual IEEE Int. Conf. on Computer Communications.
[25] Mishra, A.K., Hellerstein, J.L., Cirne, W. and Das, C.R., 2010. Towards characterizing cloud backend workloads: insights from google compute clusters. ACM SIGMETRICS Performance Evaluation Review, 37(4), pp.34-41.
[26] Han, P., Du, C., Chen, J. and Du, X., 2020. Minimizing Monetary Costs for Deadline Constrained Workflows in Cloud Environments. IEEE Access, 8, pp.25060-25074.
[27] Chekired, D.A., Khoukhi, L. and Mouftah, H.T., 2018. Industrial IoT data scheduling based on hierarchical fog computing: A key for enabling smart factory. IEEE Transactions on Industrial Informatics, 14(10), pp.4590-4602.
[28] Fan, Q. and Ansari, N. Workload allocation in hierarchical cloudlet networks. IEEE Communications Letters, 22(4), 2018.
[29] Wang, P., Zheng, Z., Di, B. and Song, L., 2019. HetMEC: Latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing. IEEE Transactions on Wireless Communications, 18(10), pp.4942-4956.
[30] El Haber, E., Nguyen, T.M. and Assi, C., 2019. Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds. IEEE Transactions on Comms., 67(5), pp.3407-3421.
[31] Peixoto, M., Genez, T. and Bittencourt, L.F., 2021. Hierarchical Scheduling Mechanisms in Multi-Level Fog Computing. IEEE Trans. on Services Computing.
[32] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Prac. & experience, vol. 41, no. 1, pp. 23–50, 2011.
[33] Naha, R.K., Garg, S., Chan, A. and Battula, S.K., 2020. Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment. Future Generation Computer Systems, 104, pp.131-141.
[34] Karimiafshar, A., Hashemi, M.R., Heidarpour, M.R. and Toosi, A.N., 2021. An Energy-Conservative Dispatcher for Fog-Enabled IIoT Systems: When Stability and Timeliness Matter. IEEE Trans. on Services Computing.
[35] Li, L., Guan, Q., Jin, L. and Guo, M., Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system. IEEE Access 7, 2019.
[36] Adhikari, M., Mukherjee, M. and Srirama, S.N., 2019. DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing. IEEE Internet of Things Journal, 7(7), pp.5773-5782.
[37] Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., Da Silva, R.F., Livny, M. and Wenger, K., 2015. Pegasus, a workflow management system for science automation. Future Generation Computer Systems, 46, pp.17-35.
[38] Li, L., Guo, M., Ma, L., Mao, H. and Guan, Q., 2019. Online workload allocation via fog-fog-cloud cooperation to reduce IoT task service delay. Sensors, 19(18), p.38-30.
[39] Sonmez, C., Ozgovde, A. and Ersoy, C., 2019. Fuzzy workload orchestration for edge computing. IEEE Trans. on Network & Service Management, 16(2), 2019, pp.769-782.
[40] Almutairi, J. and Aldossary, M., 2021. A novel approach for IoT task offloading in edge-cloud environments. Journal of Cloud Comp., 10(1), 2012, pp.1-19.

Amanjot Kaur is currently pursuing a PhD from Indian Institute of Technology, Ropar, India. Her research interests are Cloud Computing, Fog and Edge Computing.

Nitin Auluck is an associate professor in the Dept. of Computer Science & Engineering at the Indian Institute of Technology Ropar, Punjab, India. His research interests include fog computing, real-time systems, and parallel and distributed systems.

Omer Rana is professor of performance engineering at the School of Computer Science & Informatics at Cardiff University, UK.