Scheduling Real-Time Security Aware tasks in Fog Networks

Anil Singh, Nitin Auluck, Omer Rana, *Member, IEEE*, Andrew Jones, *Member, IEEE*, Surya Nepal, *Member, IEEE*

Abstract—Fog computing brings the cloud closer to a user with the help of a micro data center (mdc), leading to lower response times for delay sensitive applications. *RT-SANE* (Real-Time Security Aware scheduling on the Network Edge) supports batch and interactive applications, taking account of their deadline and security constraints. RT-SANE chooses between an mdc (in proximity to a user) and a cloud data center (cdc) by taking account of network delay and security tags. Jobs submitted by a user are tagged as: private, semi-private and public, and $mdcs$ and $cdcs$ are classified as: trusted, semi-trusted and untrusted. RT-SANE executes private jobs on a user's local $mdcs$ or pre-trusted $cdcs$, and semi-private and public jobs on remote $mdcs$ and $cdcs$. A security and performance-aware distributed orchestration architecture and protocol is made use of in RT-SANE. For evaluation, workload traces from the CERIT-SC Cloud system are used. The effect of slow executing straggler jobs on the Fog framework are also considered, involving migration of such jobs. Experiments reveal that *RT-SANE* offers a higher "success ratio" (successfully completed jobs) to comparable algorithms, including consideration of security tags.

Index Terms—Fog computing, mobile data center, cloud data center, security aware services, real-time systems.

1 Introduction

Internet of Things (IoT) enabled devices are now increasingly generating large amounts of data. In real-time systems, there is a requirement for this data to be processed within a specified deadline [9]. Although numbers may vary (across Gartner, Cisco and other market forecasts), it is estimated that we will have multi-billion Internet enabled devices by 2020 [26]. Hence, there is a strong need to construct distributed systems that can be successful in analyzing "big data" produced from these IoT devices. There has also been significant hardware innovation over recent years, with servers of the not so distant past being of comparable performance to current day mobile phones. Availability of such user-owned devices have enabled processing of data intensive applications in geographical proximity to users. A cloud data center (cdc) can be used to process the data that is generated by such IoT devices [15], and is the most dominant execution mode currently being used. The downside is the significant network latency involved between the cdc and the IoT devices. Real-time applications would however miss their processing deadlines by the time their data reaches the cdc [18]. Multi-user gaming, image/video rendering, audio/video content streaming, smart & autonomous cars, etc. are some examples of real-time applications that have such latency-sensitive processing requirements.

By using the network edge to perform as much computation as possible [3], one can get around the latency that would be involved if application data was sent across the network to the cloud. Switches, routers, gateways are examples of some edge devices that can execute jobs that would have been scheduled to run at the cloud data center. Bittencourt, Lopes, Petri and Rana [15] and Dastjerdi, Gupta, Calheiros, Ghosh and Buyya [17] discuss the edge computing paradigm, where access points are used by applications to retrieve and transfer the data to a cdc. These access points may be enhanced to provide both storage and computation capacity at the edge of the network, and would be referred to as $mdcs$ (also known as cloudlets). The communication between phones, $mdcs$ and the cdc is as follows: smart phones $\Leftrightarrow$ $mdcs$ $\Leftrightarrow$ cdc. Moreover, peer-to-peer communication between the various $mdcs$ is also possible. This communication is needed for the storage of the execution states of applications, and may be used to preempt applications from their local mdc, on the account of mobility, and to later resume applications on a new mdc.

Additionally, based on their characteristics and QoS requirements, applications may broadly be classified into two categories: interactive and batch [29]. Interactive tasks are typically less compute intensive, and require real-time performance, i.e. the task should be finished by a specified deadline. Batch jobs, on the other hand, are more compute intensive, and may not have a real-time requirement. Intuitively, interactive tasks may be executed on the edge, i.e. on the mobile data centers ($mdcs$) – to limit the latency associated with sending data to a cloud data center. Two types of $cdcs$ have been considered for the execution of batch jobs: private and public $cdcs$. Private $cdcs$ are more secure than public $cdcs$, but this is provided at an extra cost [32]. Applications that require high security and can tolerate latency can be run on these secure $cdcs$ [33], [34].

Privacy and security capabilities in Cloud computing depend on security controls offered by a provider. These can range from the types of encryption algorithms they support, facilities for data anonymisation to hosting locations of data centres employed as part of their deployment strategy. Rahulamathavan and others [37] investigated risk of a data

- *A. Singh and N. Auluck are with the Department of Computer Science and Engineering, Indian Institute of Technology, Ropar, India. E-mail: {anil, nitin}@iitrpr.ac.in*
- *O. Rana and A. Jones are with the School of Computer Science and Informatics, Cardiff University, UK. E-mail: {ranaof, jonesacf}@cardiff.ac.uk*
- *S. Nepal is with the Commonwealth Scientific and Research Organization, Sydney, Australia. E-mail: surya.nepal@data61.csiro.au*

breach (i.e. user data privacy concern) associated with data center hosting. The security tags identified in this work are based on the analysis undertaken in [37] – whereby data privacy is associated with the types of capabilities offered by a provider and included as a tag to support discovery of an mdc or cdc (depending on the audited security capability based on these security controls). A similar approach is adopted by the Cloud Security Alliance, which makes use of self-certification through a "Cloud Controls Matrix" security methodology for Cloud providers [1]. We consider 16 data privacy controls from [37] to support the inclusion of such security tags for both $mdcs$ and $cdcs$.

In this paper, we introduce a scheduling algorithm called *RT-SANE*, that addresses both the privacy/security and real-time performance requirement of application jobs amalgamating an mdc and a cdc. In *RT-SANE*, interactive applications that are private to a user are constrained to run only on their local mdc, while private batch applications are constrained to be run on the private cdc. Applications that are semi-private (i.e. those that involve use of a data set held at the cloud data center) are sent to the local cdc (either private or public). Finally, applications that belong to the public category, may be executed at a remote mdc or a cdc (either private or public). As interactive jobs have stringent deadline requirements, they are executed on the local or foreign mdcs, provided one with spare capacity is available. Batch jobs are assumed to have *loose*/flexible or no deadlines, and are executed on the $cdcs$.

The rest of this paper is structured as follows. Related work covering combined use of fog/edge resources and a cloud system is discussed in section 2. Section 3 introduces a novel distributed orchestration architecture and protocol. Section 4 discusses the system model and provides a formulation of the research problem. The proposed algorithm *RT-SANE* is described in section 5. Section 6 talks about the results of various simulations that have been carried out. Finally, section 7 provides conclusions that can be drawn from this work.

2 Related Work

Edge computing infrastructure can provide benefit for applications with stringent latency and response time requirements, such as gaming and stream processing, enabling some initial processing to be carried out closer to the user device/data generation source. Additionally, where the network connecting a user device to a cloud data center can fail or have a variable availability profile (i.e. network Quality of Service can change significantly over time, in unpredictable ways), edge resources can either: (i) support an approximate version of capability that would be carried out within a data center [30], or (ii) enable adaptation of a pre-generated model to be carried out [27], enabling subsequent re-synchronisation of this model with the cloud once the network connection is re-established. The use of edge resources also has a bearing on issues around data ownership and *trust* in a cloud data center provider, as data shared with a cloud provider can be directly viewed and searched.

Scheduling across edge computing resources has also been explored by a number of authors. In [25], *iFogStor* and *iFogStorZ* are proposed to support scheduling – the first uses an Integer Linear Programming-based approach to find an optimal result, whereas the second uses a heuristic to create an approximate result at lower computational cost. However many of these approaches [24], [25] do not consider application deadlines. A mobility-aware scheduling algorithm (and a survey) is proposed in [12], but no support for deadline-centric tasks is provided. This aspect also aligns with focus on understanding how services can be mapped to edge resource, considering a group of possible edge nodes on which such resources can be hosted. Skarlat and others [19] show how this can be modelled as an optimisation problem, focusing on the reduction of communication delay between different services within a workflow (using a genetic algorithm to find possible solutions to this problem) and realised through the use of a cloud-fog middleware.

Some researchers have characterized and compared "real-life" workloads [28], [29]. In [29], Google's data center workloads have been studied, including over 25 million tasks, spread over 12,500 hosts. The following characteristics were studied – job length, job submission frequency, job resource utilization (both CPU & memory). Two kinds of jobs were studied: short interactive jobs and long grid jobs. These workloads may be fed as input to algorithms that schedule jobs on fog networks.

Approaches used in real-time scheduling also align with the focus of this work [9], [10], [11], with limited coverage of support for edge networks [14]. This work is an extension of [31], where we considered jobs whose characteristics were randomly generated. In this paper, we consider "real-life" workloads from the Czech CERIT Scientific Cloud (CERIT-SC) [28], consisting of both interactive & batch jobs. Interactive jobs require fast response times, e.g. a keyword search. On the other hand, a batch job is computationally intensive, with little or no I/O. This workload is typical of other data centers, where this mix of interactive & batch jobs can be observed.

3 Distributed Orchestration Architecture & Protocol

3.1 Distributed Orchestration Architecture

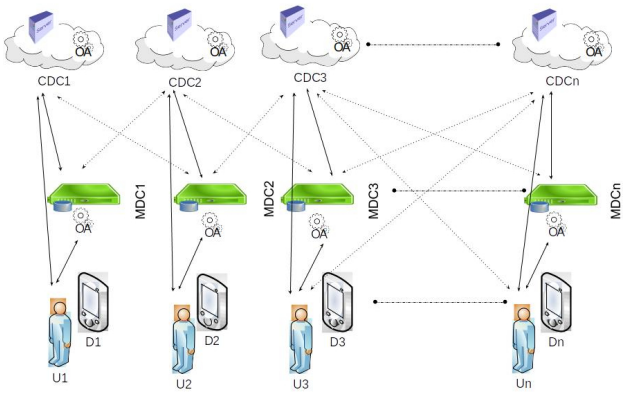

Fig. 1. Distributed Orchestration Architecture (DOA) for Edge Computing

A centralized architecture for orchestration is much simpler and easier to implement. However, such an architecture has a number of shortcomings from the perspectives of

1. https://cloudsecurityalliance.org/working-groups/cloud-controls-matrix/

security and performance. First, a centralized architecture is less resilient to failure due to a single point of failure. An attacker can launch a Denial of Service (DoS) attack. Second, the centralised orchestrator is often deployed in a cloud. The consequence of such cloud deployment is that the frequent communication needed between the edges and cloud introduces extra latency. To overcome these challenges, a fully-decentralized distributed orchestration supported by an underlying collaborative multi-agent system is introduced. In our distributed architecture, an orchestration agent resides on each computing device. Across the different nodes being used in the system, these agents create instances that are specific to each job. These agents cooperate with each other with the aim of attaining a system wide goal. For example, a goal could be the completion of a job on-time with minimum cost and meeting the specified security and privacy requirements. Deploying Orchestration Agents (OAs) and proxies can be a time consuming process however, and we assume that these either already exist, or their one-off deployment time is lower than processing requirements over an application lifetime.

Figure 1 shows a conceptual architecture of our proposed distributed orchestration mechanism for edge computing. As illustrated in the figure, a user has a mobile device (D_x) which can perform a number of activities; for example, it could generate and process data, execute a computational job, receive the output as a result of the execution of jobs, and trigger actions on the output received. We refer to a unit of work as a job/task. Both terms, job and task, are used interchangeably in this paper. In our illustration, we consider that each user device has a network connection to its local mdc (referred to as m_l), which is capable of executing jobs with low latency, and is trusted by its local (home) devices D_x. We also assume that each mdc has a connection with at least one cdc. The home $public\ cdc$ (c_{lpu}) is semi-trusted. The implication for this is that it can execute the requested jobs. However, what it cannot do is guarantee that the privacy requirements of jobs and data is satisfied. Private local and remote $cdcs$ are assumed to be trusted. Other non-local/remote mdcs : m_f and $public\ cdc$s : c_{fpu} are assumed to be untrusted.

3.2 Best Effort Orchestration Protocol

Based on the components of our distributed orchestration architecture above, a protocol for interaction between these components is proposed. We consider three different cases: (a) local mdc : m_l takes control of the execution of the job (b) local cdc : c_{lp}/c_{lpu}, takes care of the execution of the job and (c) a remote mdc : m_f is responsible for the execution of the job. A sequence diagram for these three different cases is provided in Figure 2. The aim here is to demonstrate a best effort orchestration protocol, without taking account of exception scenarios such as failures, attacks, message losses, etc.

In the first case, a device (D_1) submits a job submission request to its local mdc (mdc_1). We assume that mdc_1 meets all the specified requirements of cost, deadline and security for the submitted job. The OA sends positive acknowledgement to D_1. By default, this job is executed at the local mdc, with results sent back to D_1 on the completion of the job. Since a device trusts its local mdc, the job execution occurs at the highest level of security.

In the second case, we assume that a device D_1 submits a job request that cannot be executed on the local mdc. This could happen for a number of reasons – such as a busy mdc executing another job, the mdc does not have enough resources to execute the job or it is cheaper to execute the job at the cdc. For instance, in the scenario where the mdc is busy, the job to be executed cannot meet the deadline if it waits in a queue. In such cases, the local instance of orchestration agent OA interacts with a cdc (cdc_1) to create a proxy OA. The OA forwards the request of D_1 to cdc. The cdc agent takes over the responsibility of the completion of the job. The job is directly submitted to cdc. When the cdc completes the job, the result is returned to D_1. This is an example of a scenario in which a job is executed on a resource that is semi-trusted. Moreover, the job is also able to meet the dual requirements of deadline and cost.

In the third case, we assume that the home cdc (cdc_1) cannot meet the requirements for the job to be executed locally. Then the request of D_1 is forwarded to other $cdcs$ or $mdcs$. If any of them is capable of executing the job at that time the acknowledgement is sent to D_1 directly and then the job is submitted to it.

4 System Model

TABLE 1
Mathematical Symbols

C	cdc set
$c_{lp}, c_{fp}, c_{lpu}, c_{fpu}$	Local, remote, private & public $cdcs$
M	mdc set
m_l, m_f	Local and remote $mdcs$
JS	Job set
I	Interactive task set
B	Batch task set
D	Device set
D_i	Particular device $D_i \in D$
i_j	Specific interactive task $\in I$
b_j	Specific batch task $\in B$
T_t	Tag set allocated to tasks
T_r	Tag set allocated to resources
t_p, t_{sp}, t_{pu}	Private, semi-private, public task tags
t_{ht}, t_{st}, t_{ut}	Trusted, semi-trusted, untrusted resource tags
$cp(m)$	capacity of $mdc\ m \in M$
$cp(c)$	capacity of $cdc\ c \in C$
et	Execution cost of task i_j or b_j
st	Start time of task i_j or b_j
ct	Completion time of task i_j or b_j
d	Deadline for task i_j or b_j
cd	Communication delay
DC_{system}	Deployment cost
UT_{system}	System utilization

In this section, we provide a formulation of the research problem (using the notation in Table 1) and a possible system model. We consider $C = \{c_1, c_2, c_3, ..., c_O\}$, which is a set of O $cdcs$. Based on its functionality, a cloud data center c can be in one of four categories: local private (denoted by c_{lp}), local public (denoted by c_{lpu}), foreign private (denoted by c_{fp}) or foreign public (denoted by c_{fpu}). A cdc is linked to a set of Z $mdcs$, given by $M = \{m_1, m_2, m_3, ..., m_Z\}$. The resources available to support job execution consist of the various mdcs and cdcs. Similar to a cdc, an mdc $m \in M$ can also fall into one of two categories: a local micro data center (denoted by m_l) or a foreign micro data

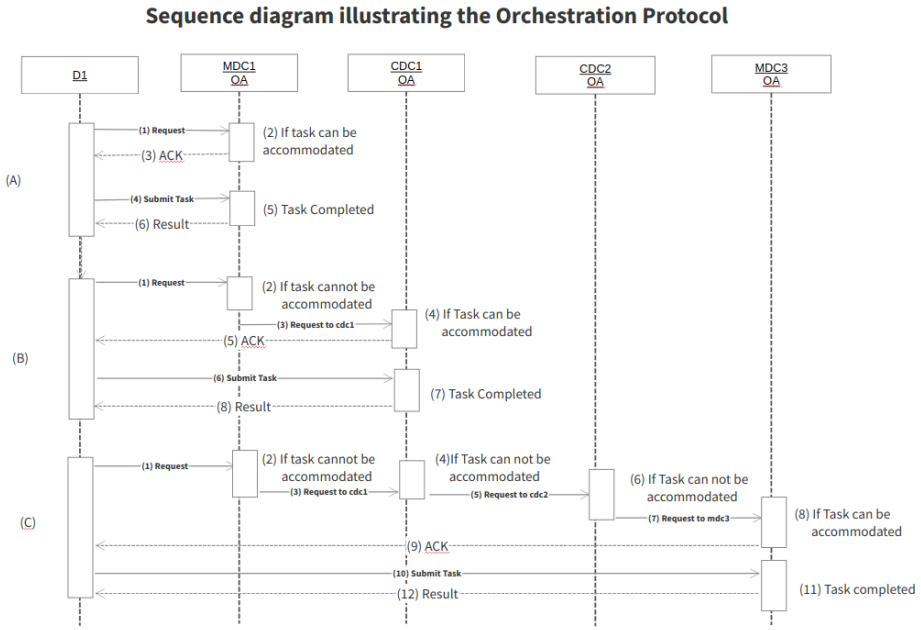

Fig. 2. Sequence diagram illustrating the Orchestration Protocol

center (denoted by m_f). Every one of the $mdcs$ has an execution rate/capacity $cp(m)$. The units of this is MIPS (i.e. Millions of Instructions per Second). For evaluation, we have used a simulator called iFogSim [6] – which also uses MIPS to specify computational capacity – this unit is therefore used to provide compatibility with the simulator. Besides MIPS, other alternatives metrics can also be used to measure computational capacity and job execution capacity of a cloud or micro data center. We also assume that all $mdcs$ demonstrate identical execution profiles. This means that a particular job needs the same amount of time on all the $mdcs$. Our model can however be easily extended to handle the case of "heterogeneous" $mdcs$. We assume a network connection, with bandwidth bw, exists between each cdc c and each mdc m.

The set of all user devices is represented by: $D = \{D_1, D_2, D_3, ..., D_n\}$. There is a set of jobs/applications, i.e. JS, that need to be executed on a cdc or an mdc, whether local or foreign. The set JS consists of sets for interactive and batch jobs, i.e. $JS = \{I, B\}$. Two categories of jobs are assumed, let $I = \{i_1, i_2, i_3, ...i_g\}$ be the set of interactive jobs. These jobs demand a short response time from the fog network. Let $B = \{b_1, b_2, b_3, ..., b_h\}$ represent the set of batch jobs. These are computationally intensive tasks, with little or no I/O involved. Such jobs aim to maximize resource utilization. The jobs in this work are real-time in nature. Hence, they need to finish their execution before the allocated deadline. Moreover, we consider the case of non-preemptive jobs. This means that a job, once started, may not be preempted by another job.

In order to model the security aspect, each job $i_j \in I$ and $b_j \in B$, is assigned a security tag. Such a set of tags is given by $T_t = \{t_p, t_{sp}, t_{pu}\}$. Tag t_p is meant for "private" jobs. Such private jobs can be run only on the local mdc m_l associated with a user or private cdc. Tag t_{sp} is meant for "semi-private" jobs. These are to be run on the local mdc m_l or the local cdc c_{lp}/c_{lpu} of a user. Lastly, tag t_{pu} is meant for "public" jobs. These tasks can be run on any execution resource - i.e. and $cdc, c_{fp}/c_{fpu}$ or mdc, m_f. According to the job tag assigned, the notation of an interactive job $i_j \in I$ becomes i_{jp}, i_{jsp} or i_{jpu} respectively. Likewise, the notation of a batch job $b_j \in B$ becomes b_{jp}, b_{jsp} or b_{jpu} respectively.

TABLE 2
Assignment of jobs to processors

	t_{ht}	t_{st}	t_{ut}
t_p	Yes	No	No
t_{sp}	Yes	Yes	No
t_{pu}	Yes	Yes	Yes

Likewise, we allocate a security tag to every execution resource. Such a set of tags is given by $T_r = \{t_{ht}, t_{st}, t_{ut}\}$. Tag t_{ht} is meant for "highly trusted" resources. These would be the local mdc, m_l of a user and private $cdc, c_{lp}/c_{fp}$. Tag t_{st} is meant for "semi-trusted" resources. These would be the local public cdc, c_{lpu} of a user. Lastly, tag t_{ut} is meant for "untrusted" resources, for example, mdcs, m_f and public cdcs, c_{fpu}. These are outside the home coverage area of users. The allocations between jobs and resources is depicted in Table 2. A value of 'Yes' means that we can allocate a particular job to a particular execution resource. On the other hand, a value of 'No' implies that the allocation between a job and an execution resource is not allowed.

The quantity et represents the execution cost of a job. The start time of task is denoted by st. In this work, we assume that all the jobs are "independent". This means that there is no precedence among jobs. Therefore, we can safely assume that all jobs may start at time = 0. Each job has a completion time, which is represented by ct.

For a particular job with a private security tag, it follows that, $ct(i_{jp}) = st(i_{jp}) + et(i_{jp})$. Owing to the privacy constraints, a private job i_{jp} can be executed only on a local mdc m_l.

The quantity $cd(i_{jp}, m_l)$ represents the communication delay associated between job i_{jp} and m_l, which is its local mdc. The bandwidth of the network connection between a user and her m_l is denoted by bw. Each user transmits data of a particular size, which is represented by $s(i_{jp})$. It takes an amount of time to initialize the network link, and this is represented at t. The cost associated with conveying the state of job i_{jp} is represented as $sc(i_{jp}, m_l)$. The communication cost between a task i_{jp} and its m_l can now be modeled as shown in equation 1. Note that equations 1 and 3 are general in nature and the parameters and their values may depend on the specific execution environment.

$$cd(i_{jp}, m_l) = t + sc(i_{jp}, m_l) + \frac{s(i_{jp})}{bw} \tag{1}$$

Since the tasks assigned to m_l are real-time, it is imperative that they finish their execution before the assigned deadline. This constraint may be represented as shown in equation 2.

$$st(i_{jp}) + et(i_{jp}) + cd(i_{jp}, m_l) \leq d(i_{jp}) \tag{2}$$

Similarly the semi-private & public interactive jobs (i_{jsp} & i_{jpu}) will be executed. Job i_{jsp} can be executed on m_l or c_{lp} or c_{lpu} while job i_{jpu} can be executed on any device that has enough capacity to accommodate it. However, priority for job execution will be given to the nearest available device.

Private batch jobs may be offloaded to the local cloud data center c_{lp} for execution. The quantity $cd(b_{jp}, c_{lp})$ represents the communication latency between b_{jp} and c_{lp}. Formally this is represented as equation 3.

$$cd(b_{jp}, c_{lp}) = t + sc(b_{jp}, c_{lp}) + \frac{s(b_{jp})}{bw} \tag{3}$$

In this case, we run task b_{jp} on its local cdc, c_{lp}. Note that the b_{jp} needs to finish prior to the expiry of its deadline. Equation 4 represents this condition:

$$st(b_{jp}) + et(b_{jp}) + cd(b_{jp}, c_{lp}) \leq d(b_{jp}) \tag{4}$$

Similarly the semi-private & public batch jobs (b_{jsp} & b_{jpu}) will be executed. Job b_{jsp} can be executed on c_{lp} or c_{lpu} while b_{jpu} can be executed on c_{lp} or c_{lpu} or c_{fp} or c_{fpu}. However, priority for job execution will be given to the nearest available device.

Note that the $mdcs$ may already be executing tasks prior to dispatching more tasks to them. Hence, it should not be the case that dispatching i_{jp} to m_l leads to the missing of deadlines of tasks(s) that are already running or are scheduled to run. The quantity $J(m_l)$ represents the set of tasks that are already executing on an mdc, m_l. Let us denote $et(i_{jp})$ as the run cost/time of a task $i_{jp} \in J(m_l)$. Now, we may pick a task i' for execution on an mdc, m_l, if and only if equation 5 holds.

$$\forall i_{jp} \in J(m_l), \forall m_l \in M, st(i_{jp}) + et(i_{jp}) + cd(i_{jp}, m_l) \leq d(i_{jp}) \tag{5}$$

Note that similar equations (numbers 1 - 5) need to hold for each interactive jobs $i_{jsp}, i_{jpu} \in I$, and for batch job $b_{jp}, b_{jsp} \in B$ as well.

The number of interactive tasks is denoted by g and the number of batch tasks is denoted by h. This includes the sum of all private, semi-private and public batch jobs. Let the sum of both interactive and batch jobs be given by y, such that $y = g + h$. Further, we denote the number of tasks that finish their execution before their deadlines as y'. Note that $y' \leq y$. A popular metric to evaluate the performance of a real-time system is called Success Ratio (SR), which is simply the number of tasks that finish execution before their deadline divided by all the jobs examined for execution. Formally, we may state that $SR = \frac{y'}{y}$. The Success Ratio assumes significance in a distributed system, as it has an impact on the correctness of running an application.

An important metric that needs to be considered is the cost of deployment of $mdcs$ and $cdcs$. These costs are denoted by $DC(m)$ and $DC(c)$. This corresponds to the monetory cost of keeping these devices in operation, for running the various jobs. These devices may be leased from various service providers. The costs due to power and cooling also need to be considered. The deployment cost would therefore depend on how long the $mdcs$ and $cdcs$ are up and running. Now, $DC(m_l)$ denotes the deployment cost of a particular $m_l \in M$. There are multiple parts in $DC(m_l)$. One part is $cd(i_{jp}, m_l)$, which is the network communication latency between a private task i_{jp} and a local mdc, m_l. The mdc needs to wait till the job is received over the communication channel. Next is, $t(m_l)$, which is the sum of execution costs of all jobs assigned to m_l. This is directly proportional to the operation time of the server, while it is running the tasks. On combining the above parts, we come up with equation 6 that models the deployment cost for each $mdc, m_l \in M$:

$$DC(m_l) = \sum_{j=1}^{N(j_p, m_l)} cd(i_{jp}, m_l) + t(m_l) \tag{6}$$

The total number of jobs assigned to an mdc is denoted by $N(j, m)$. Here, j may be interactive or batch, private or semi-private or public. Likewise, m may be local or foreign. Similar notation would exist for jobs assigned to $cdcs$ as well. Next, $et(i_{jp}, m_l)$ models the execution cost of a task i_{jp} assigned to m_l. Here, $t(m_l)$ can be represented in the form of equation 7.

$$t(m_l) = \sum_{j=1}^{N(j_p, m_l)} et(i_{jp}, m_l) \tag{7}$$

Likewise, there will be a deployment cost $DC(m_f)$ for all $m_f \in M$. These equations are similar to equations 6 and 7, and have been omitted owing to space constraints.

Similar to $DC(m_l)$ described in equation 6, the deployment costs of all $mdcs$ $m_X \in M$ is represented below as equation 8. M here is the set of all mcs, local and foreign.

$$DC(M) = \sum_{X=1}^{Z} DC(m_X) \tag{8}$$

In a similar manner, equation 9 models the deployment cost of a local cdc, c_{lpu}.

$$DC(c_{lpu}) = \sum_{j=1}^{N(j_{pu}, c_{lpu})} cd(b_{jpu}, c_{lpu}) + t(c_{lpu}) \tag{9}$$

Here, cd is the communication delay between a public batch job b_{jpu} and the cdc. Note that the cdc needs to wait till it receives the job over the communication network. The quantity $t(c_{lpu})$ represents the sum of execution costs of all jobs executed on the cdc. This is directly proportional to the operation time of the cdc, and is given by the following equation 10:

$$\sum_{j=1}^{N(j_{pu},c_{lpu})} et(b_{jpu},c_{lpu}), \forall b_{jpu} \in J(c_{lpu}) \quad (10)$$

Likewise, there will be a deployment cost $DC(c_{fpu})$, $DC(c_{lp})$, $DC(c_{fp})$ for all $c_{fpu}, c_{lp}, c_{fp} \in C$. These equations are similar to equations 9 and 10, and have been omitted owing to space constraints.

Similar to $DC(c_{lpu})$ shown above in equation 9, the deployment costs of all $c_X \in C$ is represented below as equation 11. C here is the set of all $cdcs$, local and foreign.

$$DC(C) = \sum_{X=1}^{O} DC(c_X) \quad (11)$$

Lastly, we represent the total cost of deployment of all cdcs & mdcs, in the form of equation 12.

$$DC_{system} = DC(M) + DC(C) \quad (12)$$

Since the deployment costs are directly related to the time for which the $mdcs$ are up and running, the goal is to minimize these costs. This is done by minimizing the number of $mdcs$ used. Another mdc is considered for job execution only if the local mdc is overloaded.

Next, we define utilization (UT) , which is the percentage of time that an execution resource (mdc or cdc) is busy executing jobs. The value of UT will be in range $\{0-1\}$. In order to justify leasing/purchase costs, the utlization needs to be maximised. We model the utilization of a particular mdc, $m_l \in M$ using equation 13.

$$UT_{m_l} = \frac{\sum_{j=1}^{N(j_p,m_l)} et(i_{jp},m_l)}{cp(m_l)}, \forall i_{jp} \in J(m_l) \quad (13)$$

Similarly, we calculate the utilization of an $mdc, m_f \in M$, which is given by UT_{m_f}.

Likewise, we can represent the utilization of a cdc, $c_X \in C$ in the form of equation 14.

$$UT_{c_{lpu}} = \frac{\sum_{j=1}^{N(j_{pu},c_{lpu})} et(b_{jpu},c_{lpu})}{cp(c_{lpu})}, \forall b_{jpu} \in J(c_{lpu}) \quad (14)$$

The total utilization of all the $mdcs$ is obtained by adding the utilization for all individual $mdcs$, and is given by equation 15. Here, mdc m_X can be local or foreign.

$$UT(M) = \sum_{X=1}^{Z} UT_{m_X}, \forall m_X \in M \quad (15)$$

Likewise, the total utilization of all $cdcs$ is obtained by adding the utilization for all individual $cdcs$, and is given by equation 16. Here, cdc c_X can be local or foreign.

$$UT(C) = \sum_{X=1}^{O} UT_{c_X}, \forall c_X \in C \quad (16)$$

The total system utilization is obtained by adding $UT(M)$ and $UT(C)$, and can now be represented in the form of equation 17.

$$UT_{system} = UT(M) + UT(C) \quad (17)$$

We now represent the optimization problem that needs to be solved:

Maximize SR, i.e. maximize $\frac{y'}{n}$ and UT_{system}, while minimizing $DC_{system}, \forall m_X \in M, \forall c_X \in C$. Note that this is based on the constraints between tasks and execution resources, that are depicted in table 2. Note that this takes care of the privacy issues of the users. By maximizing the Success Ratio (SR), we are ensuring that as many tasks as possible meet the following conditions: $\forall i_{jp} \in I, \forall b_{jpu} \in B, \forall m_X \in M, \forall c_X \in C, \forall u_i \in U$, the following equations A and B need to hold:

$$st(i_{jp}) + et(i_{jp}) + cd(i_{jp},m_l) \leq d(i_{jp}) \quad (A)$$

$$st(b_{jpu}) + et(b_{jpu}) + cd(b_{jpu},c_{lpu}) \leq d(i_{jpu}) \quad (B)$$

Similar equations (numbers A - B) would need to hold $\forall i_{jsp}, i_{jpu}$, as well as for $\forall b_{jp}$ and $b_{jsp} \in B$.

5 PROPOSED SCHEME: *RT-SANE*

In this section, we describe in detail our proposed scheduling scheme *RT-SANE*. As mentioned earlier, on page 4, jobs may be assigned one of three security labels: – the label t_p is assigned to tasks that are private, the label t_{sp} is allocated to tasks that are semi-private, and finally, public tasks are assigned a label of t_{pu}. In a similar fashion, the execution resources are also assigned one of three security labels: – the label t_{ht} is reserved for resources that are highly trusted, the label t_{st} is for resources that are semi-trusted, and finally, untrusted resources are allocated a label of t_{ut}. Note that either an mdc, or a cdc could represent a resource. For a particular task $i_j \in I$, or $b_j \in B$, it is assumed that m_l, c_{lp} & c_{fp} are trusted. Additionally, c_{lpu} is semi-trusted, and other $mdcs$ and $cdcs$ fall under the untrusted category. Note that m_l, c_{lp} and c_{lpu} stand for the local mdc, private cdc and public cdc respectively. The sets of tasks that need to be executed - I & B, are partitioned into 2 queues. All interactive jobs go to queue Q_i, and all batch jobs are sent to queue Q_b. There are some schedulability conditions that need to hold in our proposed algorithm. These are as follows:

- ***MDC deadline condition (C_1)***: $st(i_{jp}) + et(i_{jp}) + cd(i_{jp},m_l) \leq d(i_{jp})$.
- ***CDC deadline condition (C_2)***: $st(b_{jpu}) + et(b_{jpu}) + cd(b_{jpu},c_{fpu}) \leq d(b_{jpu})$.
- ***MDC spare capacity condition (C_3)***: $\forall i_{jp}, \forall m_l$, $et(i_{jp}) \leq (cp(m_l) - \sum_{j=1}^{N(j_p,m_l)} et(i_{jp},m_l))$.
- ***CDC spare capacity condition (C_4)***: $\forall b_{jpu}, \forall c_{fpu}$, $et(b_{jpu}) \leq (cp(c_{fpu}) - \sum_{j=1}^{N(j_{pu},c_{fpu})} et(b_{jpu},c_{fpu}))$.

Note that similar schedulability conditions would need to hold for all semi private, public interactive and batch jobs, as well as for all local and foreign $mdcs$ and $cdcs$.

Conditions C_1 & C_2 ensure that all job deadlines are met. Conditions C_3 & C_4 ensure that jobs are assigned to resources for execution only if there is sufficient spare capacity available on the resources.

The proposed algorithm is non-preemptive. This means that once a job starts running on a mdc or cdc, we cannot block the execution of this job in order to run some other job. In the algorithm, job $i_j \in I$ could be private, semi private or public - i_{jp}, i_{jsp}, i_{jpu}. Likewise, job $b_j \in B$ could also be private, semi private or public - b_{jp}, b_{jsp}, b_{jpu}. The steps of *RT-SANE* are as follows. First, we calculate the following mathematical quantities for all $i_j \in I$, and for all $b_j \in B$: st (start time), ct (completion time), cd (communication delay). The scheduling queues Q_i and Q_b are populated with the interactive and batch jobs, respectively. Next, based on the EDF algorithm, we sort all jobs in the queues in a non-decreasing order of their deadline values. Hence, the first job in the queue has the least deadline. In the event that more than one job has the same deadline, the execution sequence is done on a random basis. The scheduler now makes an attempt to execute all private jobs on their m_l, subject to the the meeting of conditions C_1 and C_3 (deadline and spare capacity conditions). If this does not happen, the jobs have no choice but to wait and try execution at a later instance of time. Note that in the case of private jobs, they can only be executed on their local mdc, and nowhere else. The preemption of job execution is carried out by the distributed orchestrator. The schedule tries to execute all jobs that are semi-private and public on m_l or c_l. However, in case these are already executing jobs to their full capacity, the scheduler sends these jobs to a remote mdc or cdc for execution. If, for a remote mdc, both conditions C_1 and C_3 are successfully satisfied, the job may be executed there. In case this is not feasible, the scheduler sends the job to a remote cdc for execution. Note that irrespective of the mdc or cdc used to execute jobs, both the "deadline" and the "spare capacity" conditions need to be satisfied.

5.1 RT-SANE Analysis

Now, we carry out the analysis of the proposed algorithm RT-SANE. Let y, be the total number of jobs that need to be scheduled in the system, such that $y = g + h$. Here, g is the number of interactive jobs, and h is the number of batch jobs. Likewise, let w be the total number of $mdcs$ and $cdcs$, such that $w = Z + O$. Here, Z is the number of $mdcs$ and O is the number of $cdcs$. Here, y includes all private, semi-private and public jobs, and w includes all highly trusted, semi trusted and un-trusted $mdcs$ and $cdcs$. In the first stage of RT-SANE, the quantities st, ct and cd are calculated for all jobs. The complexity of this stage is $O(y * w)$. The next stage involves the addition of all jobs to the queues Q_i and Q_b. The complexity of this stage is $O(y)$. In the next stage, both queues are sorted in increasing order of job deadlines. The complexity of this stage is $O(y^2)$. In the next stage, we check the spare capacity and deadline condition for all jobs. The complexity of this stage is $O(y * w)$. The next step involves assigning the jobs to the $mdcs$ and $cdcs$. The complexity of this step is $O(y + w)$. Finally, we calculate the DC and UT values, for a complexity of $O(y + w)$. On adding all these terms, the complexity of RT-SANE becomes $O(y*w)+ O(y) + O(y^2) + O(y * w) + O(y + w) + O(y + w) = O(2 * y * w) + O(2 * (y + w)) + O(y) + O(y^2)$. Now, we also know that the total number of jobs is much greater than the total number of $mdcs$ and $cdcs$, so $y \gg w$. Therefore, the above equation evaluates to $\sim O(y^2)$.

Algorithm 1 RT-SANE

procedure RT-SANE
  Compute quantities st, ct, cd, $\forall i_j \in I, \forall b_j \in B$.
  Fill Q_i & Q_b with t_p, t_{sp} & t_{pu} jobs.
  Order Q_i & Q_b in non-decreasing deadline sequence.
  $\forall i_j, b_j$ with tag t_p:
  if (C_1 holds for m_l) && (C_3 holds for m_l) **then**
    schedule i_j on m_l.
  else
    try job submission in future.
  end if
  if (C_2 holds for c_{lp}) && (C_4 holds for c_{lp}) **then**
    schedule b_j on c_{lp}.
  else
    try job submission in future.
  end if
  $\forall i_j, b_j$ with tags t_{sp} or t_{pu}:
  if (C_1 is met on m_l) && (C_3 is met on m_l) **then**
    schedule i_j on its local mdc m_l.
  end if
  if (C_2 is met on c_{lpu}/c_{lp}) && (C_4 is met on c_{lpu}/c_{lp}) **then**
    schedule b_j on its local cdc c_{lpu}/c_{lp}.
  end if
  if (C_1 is met on m_f) && (C_3 is met on m_f) **then**
    schedule i_j on a remote mdc m_f.
  end if
  if (C_2 is met on c_{fpu}/c_{fp}) && (C_4 is met on c_{fpu}/c_{fp}) **then**
    schedule b_j on a remote cdc c_{fpu}/c_{fp}.
  end if
  Compute quantities DC, UT, $\forall m, c \in M, C$.
end procedure

6 Simulation Results

Now, we elaborate on the simulation results that we have done with the goal of performance evaluation of the proposed algorithm RT-SANE. We have used sample situations that agree with our proposed Distributed Orchestration Architecture (DOA), explained in figure 1. The proposed algorithm *RT-SANE* has its roots in this architecture and protocol. The following three cases can arise in the orchestration protocol:

- Tasks (whether interactive or batch) are executed on their local mdc m_l.
- Tasks are run on their local private cdc c_{lp} or local public cdc c_{lpu}.
- Tasks are run either on a foreign mdc m_f, or on a foreign cdc c_{fp} or c_{fpu}.

As far as the security requirements are concerned, m_l, c_{lp} are trusted, c_{lpu} is semi-trusted, and m_fs, c_{fpu}s are untrusted.

Different cloud systems vary in their performance profiles, and the results are likely to change depending on the system we use. This has also been demonstrated through

benchmarking of cloud systems in the past, where different choice and combinations of VMs lead to differing application execution profiles [35], [36]. Based on this analysis, our focus in this paper has been to investigate bounds on performance using various scheduling strategies. Our results align with trends observed in the above publications.

We have compared RT_SANE with $iFogStor$ [25], another scheduling algorithm for the edge-cloud architecture. $iFogStor$ is a non-real time data placement technique in the fog environment that uses heterogeneity and geo-location to assign tasks to the edge and the cloud hosts so that the overall system latency is minimised. $iFogStor$ divides the fog nodes into zones. A zone contains the fog devices that are in the same geographical area. The task placement is done in the same zone to reduce the total delay. Heterogeneity is used to find a suitable host, as a job may only execute on a specific kind of host.

6.1 Workload

For the simulations, we have considered the workload from the CERIT-SC system [28], which is a scientific cloud from Czechoslovakia, hosted by Masaryk University. This cloud has a capacity of roughly 5500 CPU cores & 5 Petabytes of storage. Roughly 75% of the CPU cores are virtualized, and the remaining cores are used for "bare metal" applications.

The workload was collected for a period of one year, i.e. 2016. This workload is mixed and consists of two types of jobs – cloud VMs and grid jobs. By definition, the grid jobs are computationally intensive, and request more CPUs. On the other hand, cloud VMs are less demanding of the CPU. Hence, grid jobs may be executed on the $cdcs$, and the VMs may be executed on the $mdcs$.

The Fog environment considered for simulations consists of 12 $mdcs$ (i.e. Z = 12) and 1 cdc (i.e. O = 1). The communication delay (cd) from a device D_i to an mdc is 5 milliseconds and from D_i to a cdc is 105 milliseconds (5 milliseconds from the D_i to the proxy server and 100 milliseconds from the proxy server to cdc). The capacity of each mdc, $cp(m)$ has been taken as 3500 MIPS and the capacity of the cdc, $cp(c)$ has been taken as 60000 MIPS. The bandwidth (bw) from user to $mdcs$ is 1000 $mbps$ while from proxy server to the cdc is 4000 $mbps$. The number of jobs (i.e. Job Set JS) varies from 180 to 300 and the execution costs of these jobs (i.e. et) varies from 155 to 9786 MIPS. Each job represents a user with mobile device (D_x). Unless stated explicitly, we assume that 60% of the jobs are interactive, and 40% of the jobs are batch. iFogSim does not provide the means to model the network link initializing time t, or the cost to transfer the state of a job sc. Hence, we set both of these to 0. Finally, note that this is one particular representation of the job characteristics. Users may vary the characteristics, based on the requirement.

6.2 Simulation Setup & Parameters

For carrying out the simulations, we chose iFogSim [6]. This simulator has allowed us to model various features of $mdcs$ and $cdcs$. The iFogSim simulator is capable of evaluating various fog and cloud environment scheduling strategies, hence it has been extremely useful to us. This simulator follows the sensor $\rightarrow$ processor $\rightarrow$ actuator representation. Hence, it becomes pertinent for various devices that are edge-enabled. We have created a class named *MultipleApps*. This class has been used to store both the deadlines of the tasks, as well as their $MIPS$ capacity needs. Additionally, the following have also been added to this class: execution capabilities of all the mdcs and cdcs, the communication network delay cd, and the allocation of modules. There is a function called *updateAllocatedMips* (this is located in the class *FogDevice*), that allocates the $MIPS$ requirements of all the different execution modules. We have made suitable modifications to this class so that it is now possible to factor in the task deadlines. The "time shared" method has already been coded in the simulator. In the simulator, there is a job priority queue. This queue contains the modules in an increasing deadline sequence, which is from the head of the queue to its tail, or in a first come first serve sequence. Additionally, we have coded a function that can tell us if a module has run to completion. If this is the case, we take away that module from the queue, to give a chance to the remaining jobs in the queue to run on the mobile and cloud data centers. The following parameters have been used in our simulations:

Success Ratio (SR): This is defined as the percentage of tasks that finish execution before their deadline, divided by the the total number of tasks that were reviewed for scheduling. Formally, $SR = \frac{y'}{y} \times 100$.

Throughput: defined as the number of jobs that complete their execution within a particular time frame.

Task Load (TL): defined as the $MIPS$ requirement of all jobs. The average $MIPS$ value was calculated, which was then multiplied by a factor of 1 to 6 to get a range of Task loads.

Deadline Factor (DF): provides a range over which the task deadline are varied. A small DF would be a sign of deadlines that are tight & a large value indicates looser deadlines. For calculating the DF values, we obtained a lower bound on the task deadline $d(i_j) = ect(i_j)$. Next, we calculated the average deadline of the system, and multiplied this average value from 1 to 6 to get a range of DF values.

Delay Factor (DLF): defines communication delay between tasks, the micro data centers ($mdcs$) and the cloud data centers ($cdcs$). Initially, we set the delay value (cd) to 5 msec. between the user and an mdc, and 100 msec. between the user and the cdc. We increment these initial values by 5 msec.

6.3 Results & Discussion

6.3.1 Effect of including Edge capacity on Performance

The motivation for this section is to demonstrate the benefit of employing Fog Devices ($mdcs$), to augment the capability of the Cloud data-center (cdc). The performance has been measured in terms of the Success Ratio (SR) for the following two scheduling schemes: $RT - SANE$ and $cdc - only$. In the $cdc - only$ algorithm, we dispatch all tasks to the cdc for their execution. The number of $mdcs$ has been fixed at 12. Among all the jobs considered, 80% are interactive jobs

executing on $mdcs$, and 20% are batch jobs executing on the cdc. Specifically, we observe the impact of the following metrics on the SR, for both algorithms.

1) Deadline Factor (DF).
2) Task Load (TL).
3) Delay Factor (DLF)

In the first simulation, the Deadline Factor (DF) has been increased, & it's effect on the Success Ratio (SR) has been observed.

Fig. 3. Effect of DF on SR

The communication delay between a device & an mdc is set as 5 milliseconds, and between a device & the cdc has been set as 105 milliseconds. The DF of the jobs has been increased from 1 to 6. Figure 3 shows the results of this simulation. We observe that in general, increasing the DF value results in a corresponding increase in the SR value. We observe that increasing the DF generally makes the job deadlines more "loose". Hence, both algorithms are able to ensure that a larger number of jobs meet their deadlines, & this is reflected in the increased SR. In fact, beyond a specific DF value, the deadlines are so loose, that both algorithms are able to ensure that all the jobs meet their deadlines, i.e. SR = 100%. We also observe that $RT-SANE$ offers higher SR values than $cdc-only$. Note that the interactive jobs have tight deadlines, while in comparison, the batch jobs have looser deadlines. This is because the batch jobs are much more computationally intensive. The Orchestration Agent (OA) in $RT-SANE$ tries to execute the interactive jobs on the $mdcs$ and the batch jobs on the cdc. On the other hand, in the $cdc-only$ algorithm, all jobs, whether interactive or batch are sent to the cdc for execution. This is detrimental to the interactive jobs, as owing to the large communication delays between the device & the cdc, they would not be able to meet their "tight" deadlines. Therefore, the SR values for $RT-SANE$ are higher than those for $cdc-only$.

In the next simulation, we show the effect of task load (TL) on system performance (SR). The results of this simulation are shown in Figure 4. We assume a communication delay of 5 milliseconds from users to $mdcs$, and a delay of 105 milliseconds from users to the cdc.

The task load (TL) has been increased from 1 to 6. In general, increasing the TL value puts more load on the system. This causes more and more jobs to miss their deadlines, resulting in a decrease in the SR value. Note that this behavior holds for both $RT-SANE$ & $cdc-only$.

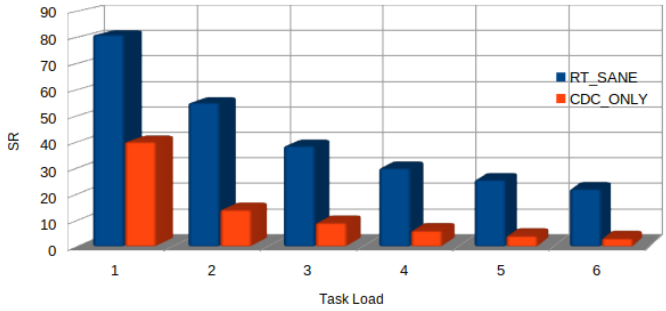

Fig. 4. Effect of TL on SR

However, $RT-SANE$ also employs $mdcs$ for the execution of the jobs. A number of tight deadline jobs, which, on being sent to the cdc, would miss their deadlines, are able to now meet their deadlines on the $mdcs$. This is due to the fact that the $mdcs$ are in proximity to the users, and hence, the user to mdc delay is less. On the other hand, jobs scheduled using $cdc-only$ are not able to take advantages of the $mdcs$, and so, a number of these tight deadline jobs end up missing their deadlines. Hence, $RT-SANE$ performs better than $cdc-only$, due to higher Success Ratio (SR) values.

In the third simulation, we observe the effect of Delay Factor (DLF) on the SR. As explained earlier, DLF is a measure of the overall communication delay in the system. As with previous simulations, the initial communication delay from the users to $mdcs$ has been taken as 5 milliseconds and delay from the users to the cdc has been taken as 105 milliseconds. In each iteration of the simulation this initial delay has been increased by 5 milliseconds. The results of this simulation are depicted in Figure 5. We observe that as the DLF value increases, more delays are introduced in the system, causing jobs to spend more time in communication. Due to this, the start time of jobs (st) at the $mdcs$ & the cdc increases. This, in effect, causes the completion time of jobs (ct) to exceed their deadlines (d). Hence, the SR value reduces for increase in DLF values.

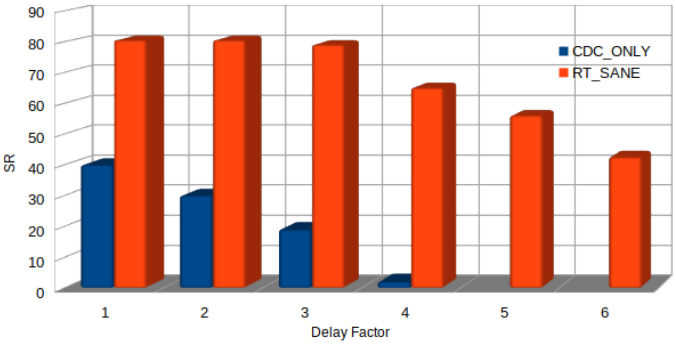

Fig. 5. Effect of DLF on SR

However, $RT-SANE$, owing to the employment of $mdcs$, is able to ensure that more jobs finish before their deadline. Hence, it's SR values are higher than those offered by $cdc-only$, which does not employ $mdcs$.

6.3.2 Effect of Deadline Heuristic on Performance

In this section, we evaluate and contrast $RT-SANE$ with $iFogStor$ [25] . Both algorithms employ $mdcs$ for scheduling jobs, in addition to the cdc. Both algorithms differ in the way that the tight deadline jobs are sent to the $mdcs$. In $RT-SANE$, jobs are sent in increasing order of deadlines, i.e. a job with the smallest deadline is sent first. On the other hand, in $iFogStor$, jobs are sent to the $mdcs$ on the basis of $mdcs'$ location with respect to the user in first come first serve order. As was the case with the previous set of simulations, we observe the impact of - deadline factor (DF), task load (TL), & delay factor (DLF) on system performance (SR). For this set of simulations, the number of $mdcs$ have been fixed at 12. Out of all the jobs considered, 80% are assumed to interactive and 20% are assumed to be batch. The orchestration agent sends the interactive jobs to the $mdcs$ and the batch jobs to the cdc for execution.

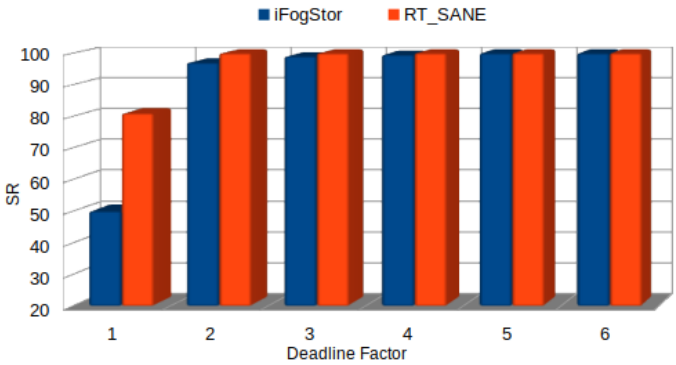

Fig. 6. Effect of DF on SR

First, we evaluate the effect of deadline factor (DF) on success ratio (SR). The communication delays from user to mdc & from user to cdc have been assumed to be 5 milliseconds & 105 milliseconds respectively. The DF has been increased from 1 to 6 & the effect on SR has been plotted. Figure 6 shows the results. From the figure, we observe that, in general, increasing the DF values makes the deadlines looser. Hence, a larger number of jobs are successful in finishing their execution before their deadlines, leading to an increase in the SR value. However, $RT-SANE$, due to it's superior heuristic of earliest deadline first, is able to ensure that a larger number of jobs are able to finish execution before the expiry of their deadlines. Note that beyond a particular DF value, the performance of both algorithms is similar. This is because, at this stage, the deadlines are so loose that the heuristic becomes irrelevant to the scheduling. As the deadline increases, so does the number of jobs that meet their deadline. $RT-SANE$ performs better than $iFogStor$ because it uses the earliest deadline first heuristic, so the jobs with smaller deadlines will execute first, while $iFogStor$ uses the nearest location $mdcs$, but in a first come first serve order. Hence jobs with loose deadline may execute first and jobs with tight deadline may execute later. As the deadlines become too loose, both algorithms perform equally well because all jobs meets their deadline.

Next, we evaluate the effect of varying task load TL on performance SR. Figure 7 shows the results of this simulation. In general, increasing TL leads to a reduction in SR. This trend can be explained as follows. An increasing TL puts more pressure on the execution elements of the system, specifically, the $mdcs$. Ultimately, the sum of execution costs of all jobs becomes greater than the execution capacities of the $mdcs$. All this leads to an increased number of jobs missing their deadlines, which is reflected in a decreasing SR. However, $RT-SANE$ outperforms $iFogStor$ because of it's superior heuristic of scheduling the job with the earliest deadline first. It is interesting to note that after a specific value of TL, the performance of both algorithms is similar. At this point, the load has become so much that a majority of jobs miss their deadlines, irrespective of which scheduling heuristic is employed. Note that the SR value is not zero, as the batch jobs are still able to meet their deadlines.

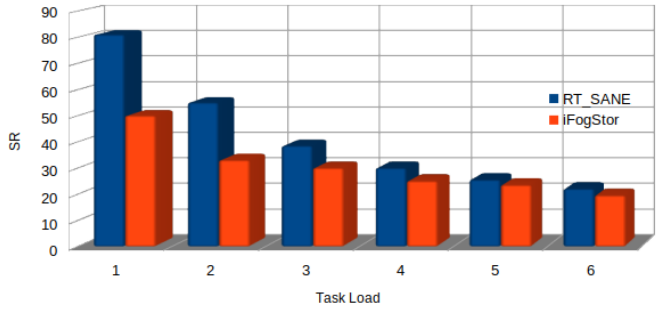

Fig. 7. Effect of TL on SR

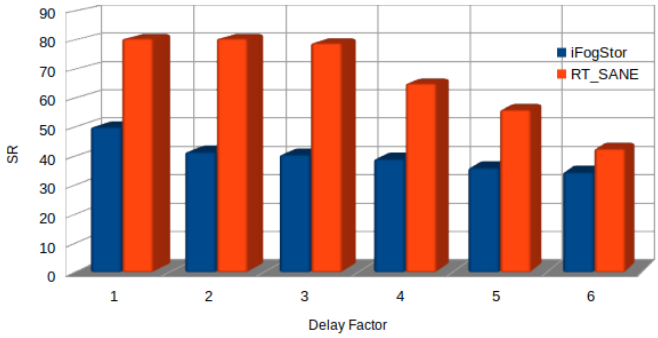

Fig. 8. Effect of DLF on SR

The next simulation studies the effect of delay factor (DLF) on the success ratio (SR). An initial communication delay of 5 milliseconds from users to $mdcs$ has been assumed. A delay of 105 milliseconds from users to the cdc has also been assumed. This initial delay is gradually increased in each iteration by 5 milliseconds. Figure 8 depicts the result for this simulation. Intuitively, increasing DLF induces more delay in the system, which leads to more number of jobs spending extended time in communication. This advances the start times & completion times of the jobs. Importantly, the completion times exceed the deadline values, and hence, the success ratio (SR) reduces. Once again, the superior heuristic used by $RT-SANE$ is the

cause of it's offering a higher SR versus $iFogStor$.

6.4 Effect of number of $mdcs$ on Performance

Next, we investigate the effect of varying the number of $mdcs$ on the system performance (SR).

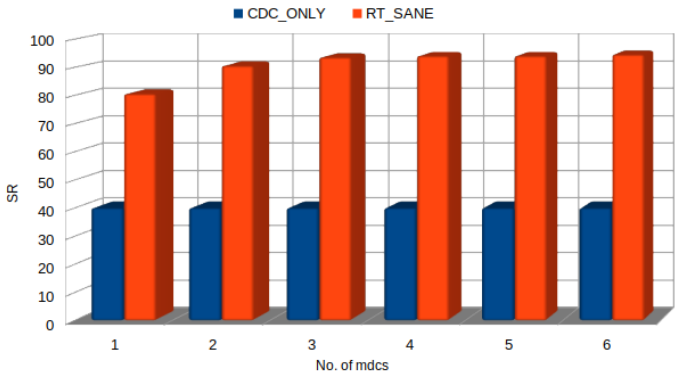

Fig. 9. Effect of No. of $mdcs$ on performance

The results of this study are shown in figure 9. As the number of $mdcs$ increases, the waiting time for jobs will decrease. As the computing capacity of the system increases, the jobs will get quicker access to an mdc, and hence, more number of jobs are able to finish before their deadlines, leading to an increase in the SR value. We observe from figure 9 that $RT-SANE$ demonstrates a higher SR value as compared to $cdc-only$. This is because of the fact that $RT-SANE$ employs $mdcs$ in addition to the cdc for job execution. Expectedly, increasing the number of $mdcs$ will not have any impact on SR for $cdc-only$ as the algorithm does not employ $mdcs$ for job execution. Figure 10 shows the same simulation, this time comparing $RT-SANE$ with $iFogStor$. Both these algorithms use $mdcs$, in addition to the cdc, for job execution. Once again, $RT-SANE$ shows higher SR values, owing to the superior real-time heuristic of earliest deadline first, that it employs.

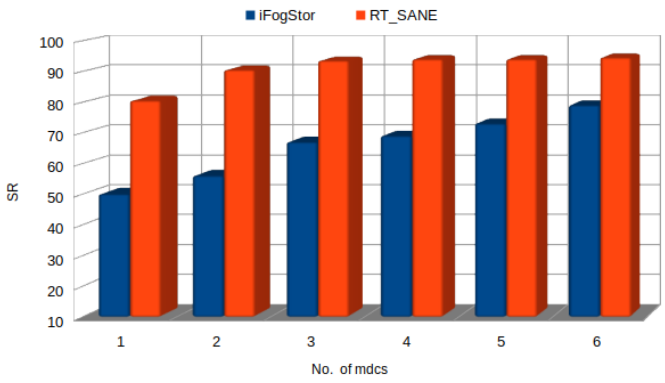

Fig. 10. Effect of No. of $mdcs$ on performance

6.5 Effect of job mix on Performance

In this section, the effect of job mix on the Success Ratio has been studied. The results are shown in figures 11 & 12. The format of the data points on the x-axis of both graphs is: *(% of interactive jobs, % of batch jobs)*. Initially, for the first data point, all jobs are batch jobs i.e. 100% of the jobs are batch and 0% of the jobs are interactive. In each iteration (data point), the number of batch jobs has been decreased by 10% and the number of interactive jobs has been increased by 10%. After repeating this process 10 times, the number of batch jobs becomes 0% and the number of interactive jobs becomes 100%. In general, as we go from one data point to the next, the SR increases. $RT-SANE$ performs better than $cdc-only$, as in the latter, all jobs (even interactive) are sent to the cdc. Hence, the performance is poor. $RT-SANE$ performs better than $iFogStor$, as it schedules jobs using the earliest deadline first heuristic, so, during a given time period, a larger number of interactive jobs are able to meet their deadlines.

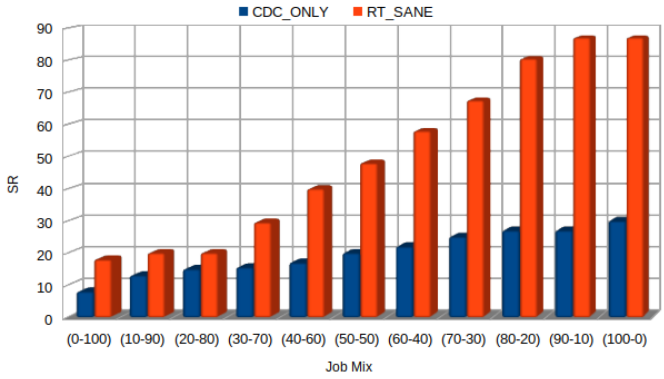

Fig. 11. Effect of job mix on performance

For the initial data points, the number of batch jobs is pretty high. As the $mdcs$ are not suitable for executing batch jobs, the performance of $RT-SANE$, $iFogStor$ and $cdc-only$ is almost similar. After increasing the number of interactive jobs, the success ratio increases, especially in $RT-SANE$ and $iFogStor$, because the Orchestration Agent tries to executes the interactive jobs on the $mdcs$ as much as possible.

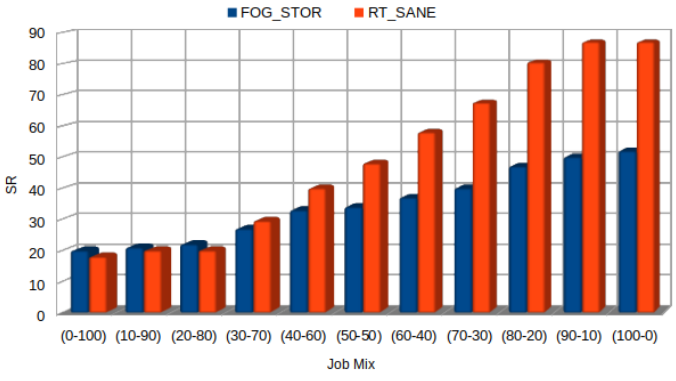

Fig. 12. Effect of jobs mix on performance

6.6 Effect of Security mix on Performance

In this section we study the effect of security tag assignment on the performance of $RT-SANE$, $iFogStor$ & $cdc-only$. Specifically, we consider three different security mixes:

- Security Mix 1: the number of private (t_p) & semi-private (t_{sp}) jobs are constant, and the number of

public (t_{pu}) jobs is repeatedly increased by $\frac{1}{3}$ every x-axis data point.
- Security Mix 2: The number of public (t_{pu}) & semi-private (t_{sp}) jobs are constant, and the number of private (t_p) jobs is increased by $\frac{1}{3}$ every x-axis data point.
- Security Mix 3: The number of public (t_{pu}) & private (t_p) jobs are constant, and the number of semi-private (t_{sp}) jobs is increased by $\frac{1}{3}$ every x-axis data point.

For this set of simulations, 160 jobs were considered initially. The results for all three security mixes are shown in figures 13 - 15. For all these figures, the format of each x-axis data point is as follows: (# of t_{pu} jobs, # of t_p jobs, # of t_{sp} jobs). Figure 13 shows the results for security mix 1.

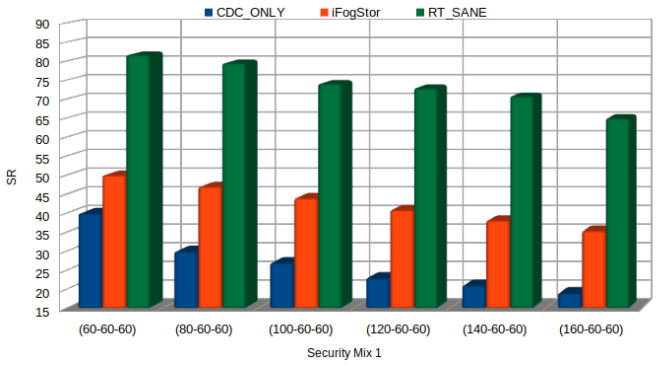

Fig. 13. Effect of security mix-1 on performance

As number of public jobs are increased, more and more load is placed on the cdc, as most of the public jobs are sent there for execution, if there is not enough spare capacity in the $mdcs$. In general, when the load is increased, more jobs will miss their deadlines. The $cdc - only$ algorithm perform the worst because of the large communication delays involved between the user & the cdc. $RT - SANE$ performs better than $iFogStor$ as it gives a higher priority to jobs with smaller deadlines.

The results for security mix 2 are shown in figure 14. In the second case, we increased the number of private jobs, & kept the number of other job types constant.

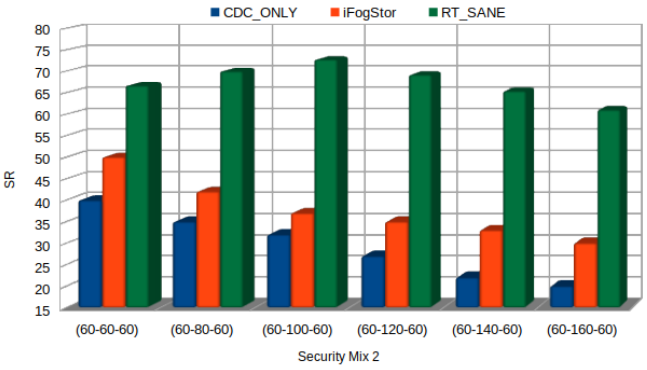

Fig. 14. Effect of security mix-2 on performance

As the private jobs are sent to local $mdcs$ only by Orchestration Agent, the $cdc - only$ approach performs the worst, as it is unable to execute the increased number of private jobs. On the other hand, the performance of the $iFogStor$ algorithm goes down gradually, because of an increase in private job traffic on $mdcs$, which results in more waiting time for jobs. Hence, more jobs start missing their deadlines, & the Success Ratio is reduced. The performance for $RT - SANE$ may be explained as follows. In this algorithm, the jobs are executed in an earliest deadline first fashion. As the number of private jobs is increased, more jobs are sent to local $mdcs$ by the Orchestration Agent. Initially, the SR increases, because there is spare capacity available in the local $mdcs$ for jobs to occupy. As soon as the local $mdcs$ reach their full capacity, the waiting time of the private jobs increases and they start missing their deadlines. This leads to a reduction in the SR value.

In the third case of security mix 3, the number of Semi-private jobs was increased, & the number of other job types was kept constant. The results of this simulation are shown in figure 15.

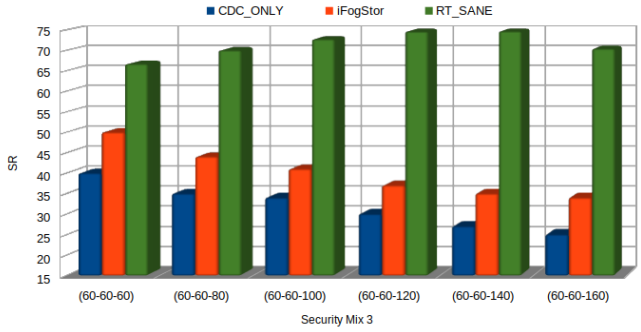

Fig. 15. Effect of security mix-3 on performance

Semi-private jobs exhibit very different characteristics, as compared to public jobs or private jobs. This is because these jobs can use either the $mdcs$, or the cdc, or both. The semi private jobs are executed on the $mdcs$ (local or foreign), until the $mdcs$ have sufficient spare capacity available. Once the $mdcs$ spare capacity is not sufficient, these jobs are assigned to the cdc by the Orchestration Agent. In case of $cdc - only$, the Success Ratio decreases gradually because of the increased job load and the high communication delay between the user and the cdc. $RT - SANE$ demonstrates better performance than $iFogStor$ due to it's superior heuristic of earliest deadline first, which ensures that a higher number of jobs finish execution before their deadline.

6.7 Effect of Task Load on Throughput

In this section, we study the effect of task load on throughput. Throughput here is defined as the number of jobs completed per unit time. A unit time of 100 milliseconds has been considered. Since our focus in this simulation is on the throughput, and not the meeting of deadlines, we have considered the deadlines of jobs to be loose enough, so that no deadlines are missed. The results are shown in figure 16. Not surprisingly, increasing the task load reduces the throughput. This is because increasing the task load leads to more pressure on the $mdcs$, which results in higher job wait times, & lower throughput. However, $RT - SANE$ offers a higher throughput than $cdc - only$, due to it's feature of employing $mdcs$ for job execution.

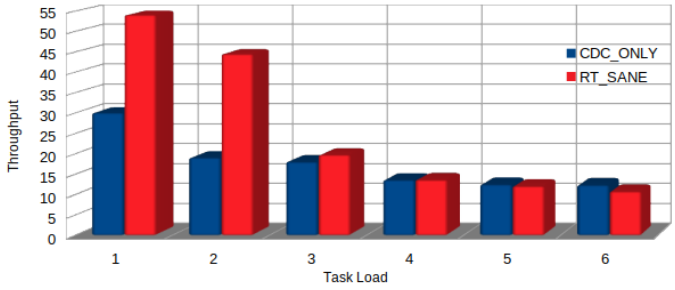

Fig. 16. Effect of task load on throughput

6.8 The case of Straggler Jobs

The goal of this simulation is to analyze the effect of straggler tasks on the system performance. By definition, straggler tasks are slow running & poor performing tasks that delay the execution of other tasks [30]. This could be due to faulty hardware, or misconfiguration. For this simulation, we considered 200 tasks and 12 $mdcs$. We consider four cases. In the first case Slow mdc_1 (Sm_1), one of the $mdcs$ is slow, i.e. has a low $MIPS$ value. In case Sm_2, two $mdcs$ are slow. Likewise, case Sm_3 and Sm_4 have 3 and 4 slow $mdcs$ respectively. An mdc is slow if its $MIPS$ value is less than the average $MIPS$ value of all $mdcs$ that have been considered. We evaluate the SR for two algorithms: $RT-SANE$, which is the basic algorithm described earlier, and $RT-SANE(JM)$, which is $RT-SANE$ with job migration enabled. Basically, we migrate all jobs from the slow $mdc(s)$ to an mdc that is free, i.e. it has its full computing capacity available.

The results for this simulation are given in Table 3. From the table, we observe that as we move from case SM_1 to SM_4, the Success Ratio (SR) offered reduces. This is due to the fact that the number of slow $mdcs$ is increasing, causing the number of straggler jobs to also increase. Hence, only a fraction of the jobs that we migrate to a free mdc are able to meet their deadlines. We also observe that the SR offered by $RT-SANE(JM)$ is higher than that offered by $RT-SANE$. This can be attributed to the feature of job migration, wherein straggler jobs on the slow $mdc(s)$ are migrated to a free mdc, which leads to a larger number of jobs being able to meet their deadlines.

TABLE 3
The case of Straggler Jobs

	$SR(RT-SANE)$	$SR(RT-SANE(JM))$
Sm_1	85.5%	93%
Sm_2	79	83.5
Sm_3	72.5	76
Sm_4	67	71

7 Conclusion

Communication latency in executing applications on the cloud is unsuitable for real-time applications. It is intuitive to execute such applications at the edge of the network as: (i) it leads to better performance (higher Success Ratio); (ii) addresses security concerns of private data. We propose *RT-SANE*, a scheduling algorithm that addresses both performance and security. We consider two types of jobs: interactive and batch. Private interactive and batch jobs are sent to the local mdc and private cloud cdc respectively. Semi private interactive and batch jobs are sent to the foreign mdc and private cdc respectively. Finally, public interactive and batch jobs are executed at the foreign mdc and foreign public cdc respectively. The security of jobs are taken into account by executing private jobs on local $mdcs$ and $cdcs$. A distributed orchestration architecture and protocol is proposed. We also consider the case of straggler jobs, which are migrated to other free $mdcs$ in the system, thereby improving overall application performance. Simulations comparing $RT-SANE$ with $iFogstor$ and $cdc-only$ on real workload data on the CERIT system [28] and Google [29] show that *RT-SANE* provides a better system performance, due to a higher value of Success Ratio.

References

[1] Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are, Cisco White Paper, 2015.
[2] A. V. Daesterjerdi and R. Buyya, Fog Computing: Helping the Internet of things to realize their potential, IEEE Computer, vol. 49, no. 8, pp. 112-116, 2016.
[3] L. F. Bittencourt, O. Rana, and I. Petri, Cloud Computing at the Edges, Springer International Publishing, 2016, pp. 3-12.
[4] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, Fog Computing: A platform for Internet of things and analytics, Springer International Publishing, 2014, pp. 169-196.
[5] B. Jennings and R. Stadler, Resource management in clouds: survey and research challenges, Journal of Network and Systems Management, vol. 23, no. 3, 2015, pp. 567-619.
[6] 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, CORR abs, vol. 1606.02007, 2016. [Online]. Available: http://arxi.org/abs/1606.02007.
[7] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. Da Fonseca, Scheduling in hybrid clouds, IEEE Communications Magazine, vol. 50, no. 9, 2012, pp. 42-47.
[8] J. D. Montes, M. Abdelbaky, M. Zhou, and M. Parashar, Comet-cloud: Enabling software-defined federations for end-to-end application work-flows, IEEE Internet Computing, vol. 19, no. 1, 2015, pp. 69-73.
[9] C. L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM, vol. 20, no. 1, 1973, pp. 46-61.
[10] Z. Guo and S. Baruah, A neurodynamic approach for real-time scheduling via maximizing piecewise linear utility, IEEE Transactions on Neural Networks and Learning Systems, vol. 27, no. 2, 2016, pp. 238-248.
[11] J. Singh, S. Betha, B. Mangipudi, N. Auluck, Contention aware energy efficient scheduling on heterogeneous multiprocessors, IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 5, 2014, pp. 1251-1264.
[12] B. A. Hridita, M. Irfan & M. S. Islam, Mobility aware task allocation for mobile cloud computing, International Journal of Computer Applications, Vol. 137, No. 9, 2016, pp. 35-41.
[13] X. D. Pham & E. N. Huh, Towards task scheduling in a cloud fog computing system, The 18^{th} Asia-Pacific Network Operations and Management Symposium, APNOMS, Kanazawa, Japan, October 5-7, 2106, pp. 1-4.
[14] M. Shojafar, N. Cordeschi and E. Baccarelli, Energy-efficient adaptive resource management for real-time vehicular cloud services, IEEE Transactions on Cloud Computing, preprint, April 6, 2016, pp. 1-14.
[15] L. F. Bittencourt, M. M. Lopes, I. Petri and O. Rana, Towards virtual machine migration in fog computing, The 10^{th} International conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), November 4-6, 2015, Krakow, Poland, pp. 1-8.
[16] F. Xia, L. T. Yang, L. Wang and A. Vinel, Internet of Things, Editorial, International Journal of Communication Systems, Vol. 25, 2012, pp. 1101-1102.

[17] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh and R. Buyya, Fog computing: principles, architectures, and applications, Internet of Things: Principles and Paradigms, R. Buyya and A. Dastjerdi (eds), Morgan Kaufmann, ISBN: 978-0-12-805395-9, Burlington, Massachusetts, USA, May 2016.
[18] J. W. S. Liu, Real-Time Systems, Prentice Hall, April 2000, 592 pages.
[19] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, P. Leitner, "Optimized IoT service placement in the fog", Journal of Service Oriented Computing and Applications (SOCA), Dec. 2017, Vol. 11, Issue 4, Springer.
[20] S. Sharif, P. Watson, J. Taheri, S. Nepal, and A. Zomaya, Privacy-aware scheduling SaaS in high performance computing environments, IEEE Transactions on Parallel & Distributed Systems, vol. 28, no. 4, April 2017, pp. 1176-1188.
[21] W. Shu, J. Cao, Q. Zhang, Y. Li, and L. Xu, Edge computing: vision and challenges, IEEE Internet of Things Journal, vol. 3, no. 5, October, 2016, pp. 637-646.
[22] M. Satyanarayanan, G. Lewis, E. J. Morris, S. Simanta, J. Boleng, K. Ha, The role of cloudlets in hostile environments, IEEE Pervasive Computing, October, 2013, pp. 40-49.
[23] M. Satyanarayanan, Augmenting cognition, IEEE Pervasive Computing, April, 2004, pp. 4-5.
[24] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. Gokhale, INDICES: Exploiting edge resources for performance aware cloud hosted services, First IEEE/ACM International Conference on Fog & Edge Computing, Madrid, Spain, May 14, 2017, pp. 75-80.
[25] M. I. Naas, P. R. P. Orange, J. Boukhobza, L. Lemarchand, iFogStor: an IoT data placement strategy for fog infrastructure, First IEEE/ACM International Conference on Fog & Edge Computing, Madrid, Spain, May 14, 2017, pp. 97-104.
[26] D. Evans, The Internet of Things, how the next evolution of the Internet is changing everything, Cisco White Paper, April 2011.
[27] A. Javed, O. Rana, C. Marmaras, and L. Cipcigan, "Fog paradigm for local energy management systems". Proc. of 2nd EAI Int. Conf. on Cloud, Networking & Internet-of-Things (CN4IoT), Brindisi, Italy, 20-21 April 2017, Springer, pp. 162-176
[28] D. Klusacek, and B. Parak, Analysis of mixed workloads from shared cloud infrastructure, the Twenty First Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), June 2017, Orlando, USA.
[29] S. Di, D. Kondo, and W. Cirne, Characterization and comparison of cloud versus grid workloads, International Conference on Cluster Computing (CLUSTER), September 2012, Beijing, China, pp. 230 - 238.
[30] C. Li, H. Shen, and T. Huang, Learning to diagnose stragglers in distributed computing, Ninth workshop of many tasks computing on clouds, grids, and supercomputers, November 2016, Salt Lake City, USA, pp. 1 - 6.
[31] A. Singh, N. Auluck, O. Rana, A. Jones, and S. Nepal, RT-SANE: Real Time Security Aware Scheduling on the Network Edge, The Tenth IEEE/ACM International Conference on Utility and Cloud Computing (UCC), December 5 - 8, 2017, Austin, USA.
[32] A. R. Zamani, I. Petri, J. D. Montes, O. F. Rana, M. Parashar, "Edge-Supported Approximate Analysis for Long Running Computations", Proc. of 5th IEEE Int. Conf. on Future Internet of Things and Cloud (FiCloud 2017), pp 321-328, Prague, Czech Republic, August 21-23, 2017. IEEE Computer Society Press.
[33] Z. Kerravala, How Organizational Efficiency Can Be Improved with Private Cloud", White Paper, ZK Research, 2017.
[34] K. Hwang, D. Li, Trusted Cloud Computing with Secure Resources and Data Coloring, IEEE Internet Computing, Vol. 14, issue 5, 2010.
[35] D. Chen, H. Zhao, Data Security and Privacy Protection Isues in Cloud Computing, International Conference on Computer Science and Electronics Engineering, Hanhzhou, China, March 2012, pp. 647 - 651.
[36] J. O'Loughlin and L. Gilliam, A Performance Brokerage for Heterogeneous Clouds, Future Generation Computer Systems, Vol. 87, Oct 2018, pp. 831 - 845.
[37] J. O'Loughlin and L. Gilliam, Should Infrastructure clouds be priced entirely on performance? An EC2 case study, International Journal of Big Data Intelligence, vol. 1, no. 4, 2014.
[38] Y. Rahulamathavan, M. Rajarajan, O. Rana, M. Awan, P. Burnap and S. Das, Assessing Data Breach Risk in Cloud Systems, IEEE CloudCom, Vancouver, Canada, 2015, pp. 363 - 370.

Anil Singh received his B. Tech degree from Uttarakhand Technical University, Uttarakhand, India and M. Tech from National Institute of Technology, H.P., India. He is currently pursuing PhD from Indian Institute of Technology, Ropar, India. His research interests are Cloud Computing and Fog & Edge Computing.

Nitin Auluck received the BE degree in electrical and electronics engineering from Gulbarga University, Gulbarga, India, in 1998 and the PhD degree from the University of Cincinnati, Cincinnati, in 2005. He is an associate professor in the Department of Computer Science and Engineering at the Indian Institute of Technology Ropar, Punjab, India. He was an assistant professor in the Department of Computer Science, Quincy University, Quincy, USA from 2004 to 2010. His research interests include fog computing, real-time systems, and parallel and distributed systems.

Omer Rana is professor of performance engineering in the School of Computer Science and Informatics at Cardiff University and leads the Complex Systems research group. He received his PhD from the Imperial College (London University) in Neural Computing & Parallel Architectures. His research interests include performance modelling & simulation, IoT, and edge analytics. Contact him at ranaof@cardiff.ac.uk.

Andrew Jones was until recently a senior lecturer in the School of Computer Science and Informatics at Cardiff University, Cardiff, UK. His research interests lie primarily in the areas of novel cloud architectures and applications and the design and interoperability of heterogeneous distributed information systems, particularly in the field of biodiversity informatics.

Surya Nepal is a Principal Research Scientist at CSIRO Data61 and leads the distributed systems security research group. He has been with CSIRO since 2000. Over the last 17 years, his main research focus has been in the development and implementation of technologies in the area of distributed systems (including cloud, IoT and edge computing) and social networks, with a specific focus on security, privacy and trust. He obtained his BE from National Institute of Technology (NIT) Surat, India; ME from Asian Institute of Technology (AIT), Thailand; and PhD from RMIT University, Australia. He has more than 200 peer-reviewed publications to his credit; his papers are published in international journals such as IEEE Trans. Parallel and Distributed Systems, IEEE Trans. on Service Computing, ACM Trans. on Internet Technologies, and IEEE Trans. on Computers. He has co-edited three books including security, privacy and trust in cloud systems by Springer, and co-invented 3 patents. He currently serves as an associate editor in an editorial board of IEEE Transactions on Services Computing.