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

I
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 ⇔ mdcs ⇔ 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
From this work. Finally, section 7 provides conclusions that can be drawn from the results of various simulations that have been carried out. 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. We consider 16 data privacy controls from [37] to support the inclusion of such security tags for both mdcs and cdc.

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 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 cdc.

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

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 centralized 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 $m_1$), 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 ($c_{fp}$) 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 cdcs 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 : $m_1$ takes control of the execution of the job (b) local cdc : $c_{fp}/c_{fp}$ takes care of the execution of the job and (c) a remote mdc : $m_2$ 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>set</td>
</tr>
<tr>
<td>$c_1$, $c_2$, $c_p$, $c_{fp}$, $c_{fp}$</td>
<td>local, remote, private &amp; public cdcs</td>
</tr>
<tr>
<td>$M$</td>
<td>set</td>
</tr>
<tr>
<td>$m_1$, $m_2$</td>
<td>Local and remote mdcs</td>
</tr>
<tr>
<td>$JS$</td>
<td>job set</td>
</tr>
<tr>
<td>$I$</td>
<td>interactive task set</td>
</tr>
<tr>
<td>$B$</td>
<td>batch task set</td>
</tr>
<tr>
<td>$D$</td>
<td>device set</td>
</tr>
<tr>
<td>$D_1$</td>
<td>particular device $D_1 \in D$</td>
</tr>
<tr>
<td>$t_j$</td>
<td>specific interactive task $\in I$</td>
</tr>
<tr>
<td>$b_j$</td>
<td>specific batch task $\in B$</td>
</tr>
<tr>
<td>$T_t$</td>
<td>tag set allocated to tasks</td>
</tr>
<tr>
<td>$T_r$</td>
<td>tag set allocated to resources</td>
</tr>
<tr>
<td>$t_{fp}, t_{sp}, t_{fp}$</td>
<td>private, semi-private, public task tags</td>
</tr>
<tr>
<td>$t_{hp}, t_{at}, t_{at}$</td>
<td>trusted, semi-trusted, untrusted resource tags</td>
</tr>
<tr>
<td>$c_{fp}(m)$</td>
<td>capacity of mdc $m \in M$</td>
</tr>
<tr>
<td>$c_{fp}(c)$</td>
<td>capacity of cdc $c \in C$</td>
</tr>
<tr>
<td>$st$</td>
<td>start time of task $t_j$ or $b_j$</td>
</tr>
<tr>
<td>$ct$</td>
<td>completion time of task $t_j$ or $b_j$</td>
</tr>
<tr>
<td>$d$</td>
<td>deadline for task $t_j$ or $b_j$</td>
</tr>
<tr>
<td>$cd$</td>
<td>communication delay</td>
</tr>
<tr>
<td>$DC_{system}$</td>
<td>Deployment cost</td>
</tr>
<tr>
<td>$U_{system}$</td>
<td>System utilization</td>
</tr>
</tbody>
</table>

In this section, we provide a formulation of the research problem (using the notation in Table 1) and a possible system model. We consider $C = \{c_1, c_2, c_3, ..., c_P\}$, 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 $c_{fp}$), local public (denoted by $c_{fp}$), foreign private (denoted by $c_{fp}$) or foreign public (denoted by $c_{fp}$). 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 center (denoted by $m_f$).
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 $cdc$ and each $mdc$. 

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.

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 $cdc$, $c_{lp}/c_{fpu}$. Tag $t_{st}$ is meant for “semi-trusted” resources. These would be the local public $cdc$, $c_{fpu}$ 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 $ct$ 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, $ct(i_{jp}) = st(i_{jp}) + ct(i_{jp})$. Owing to the privacy constraints, a private job $i_{jp}$ can be executed only on a local $mdc$ $m_l$.  

---

**TABLE 2:** Assignment of jobs to processors

<table>
<thead>
<tr>
<th>Job Type</th>
<th>Tag</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_{sj}$</td>
<td>$t_{ht}$</td>
<td>Yes</td>
</tr>
<tr>
<td>$i_{sp}$</td>
<td>$t_{st}$</td>
<td>Yes</td>
</tr>
<tr>
<td>$i_{jpu}$</td>
<td>$t_{ut}$</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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.

\[
\begin{align*}
\text{cd}(i_{jp}, m_l) &= t + sc(i_{jp}, m_l) + \frac{s(i_{jp})}{bw} \\
\text{st}(i_{jp}) + \text{et}(i_{jp}) + \text{cd}(i_{jp}, m_l) &\leq d(i_{jp})
\end{align*}
\]

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.

\[
\begin{align*}
\text{st}(i_{jp}) + \text{et}(i_{jp}) + \text{cd}(i_{jp}, m_l) &\leq d(i_{jp})
\end{align*}
\]

Similarly the semi-private & public interactive jobs \( (i_{jp} \& j_{pu}) \) will be executed. Job \( i_{jp} \) can be executed on \( m_l \) or \( c_{ip} \) or \( c_{ip} \), while job \( j_{pu} \) 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 \( c_{ip} \) for execution. The quantity \( \text{cd}(b_{jp}, c_{ip}) \) represents the communication latency between \( b_{jp} \) and \( c_{ip} \). Formally this is represented as equation 3.

\[
\begin{align*}
\text{cd}(b_{jp}, c_{ip}) &= t + \text{sc}(b_{jp}, c_{ip}) + \frac{s(b_{jp})}{bw}
\end{align*}
\]

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

\[
\begin{align*}
\text{st}(b_{jp}) + \text{et}(b_{jp}) + \text{cd}(b_{jp}, c_{ip}) &\leq d(b_{jp})
\end{align*}
\]

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

Note that the \( mdc \) 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 \( \text{et}(i_{jp}) \) 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.

\[
\forall i_{jp} \in J(m_l), \forall m_l \in M, \text{st}(i_{jp}) + \text{et}(i_{jp}) + \text{cd}(i_{jp}, m_l) \leq d(i_{jp})
\]

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 \( mdc \) and \( cdcs \). These costs are denoted by \( DC(m) \) and \( DC(c) \). This corresponds to the monetary 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 \( mdc \) 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 \( \text{cd}(i_{jp}, m_l) \), 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 tasks 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 \( mdc, m_l \in M \):

\[
\begin{align*}
DC(m_l) &= \sum_{j=1}^{N(j,p,m)} \text{cd}(i_{jp}, m_l) + t(m_l)
\end{align*}
\]

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.

\[
\begin{align*}
t(m_l) &= \sum_{j=1}^{N(j,p,m)} et(i_{jp}, m_l)
\end{align*}
\]

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 \( mdc, m_X \in M \) is represented below as equation 8. Here is the set of all \( mdc \), local and foreign.

\[
\begin{align*}
DC(M) &= \sum_{X=1}^{Z} DC(m_X)
\end{align*}
\]

In a similar manner, equation 9 models the deployment cost of a local \( cdc, c_{ipu} \).

\[
\begin{align*}
DC(c_{ipu}) &= \sum_{j=1}^{N(j,p,c_{ipu})} \text{cd}(b_{jp}, c_{ipu}) + t(c_{ipu})
\end{align*}
\]
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 \( t(c_{jpu}) \) 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:

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

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

Similar to \( DC(c_{jpu}) \) 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.

\[
DC(C) = \sum_{X=1}^{O} DC(c_{X})
\] (11)

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

\[
DC_{system} = DC(M) + DC(C)
\] (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 utilization needs to be maximized. We model the utilization of a particular \( mdc, m_l \in M \) using equation 13.

\[
UT_{m_l} = \frac{\sum_{j=1}^{N_{jpu, m_l}} et(j_{pu}, m_l)}{cp(m_l)}, \forall j_{pu} \in J(m_l)
\] (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.

\[
UT_{c_{jpu}} = \frac{\sum_{j=1}^{N_{jpu, c_{jpu}}} et(b_{jpu}, c_{jpu})}{cp(c_{jpu})}, \forall b_{jpu} \in J(c_{jpu})
\] (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.

\[
UT(M) = \sum_{X=1}^{O} UT_{m_X}, \forall m_X \in M
\] (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.

\[
UT(C) = \sum_{X=1}^{O} UT_{c_X}, \forall c_X \in C
\] (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.

\[
UT_{system} = UT(M) + UT(C)
\] (17)

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

Maximize \( SR \), i.e. maximize \( \frac{UT_{system}}{\sum_{i=1}^{n} t(i_{pu})} \) 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: \( \forall i_{jp} \in I, \forall b_{jpu} \in B, \forall m_X \in M, \forall c_X \in C, \forall u \in U \), the following equations A and B need to hold:

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

\[
st(b_{jpu}) + et(b_{jpu}) + cd(b_{jpu}, c_{jpu}) \leq d(b_{jpu}) \quad (B)
\]

Similar equations (numbers A - B) would need to hold \( \forall j_{sp}, i_{jpu} \), as well as for \( \forall b_{jpu} \) 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 \( t_{sp} \) 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 \( i \in I \), or \( b \in B \), it is assumed that \( m_l, c_{ip} \) & \( c_{fp} \) are trusted. Additionally, \( c_{ip} \) is semi-trusted, and other \( mdcs \) and \( cdcs \) fall under the untrusted category. Note that \( m_l, c_{ip} \) and \( c_{ip} \) 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** (C1): \( st(i_{jp}) + et(i_{jp}) + cd(i_{jp}, m_l) \leq d(i_{jp}) \).
- **CDC deadline condition** (C2): \( st(b_{jpu}) + et(b_{jpu}) + cd(b_{jpu}, c_{jpu}) \leq d(b_{jpu}) \).
- **MDC spare capacity condition** (C3): \( \forall i_{jp}, \forall m_l, et(i_{jp}) \leq (cp(m_l) - \sum_{j=1}^{N_{jpu, m_l}} et(i_{jp}, m_l)) \).
- **CDC spare capacity condition** (C4): \( \forall b_{jpu}, \forall c_{jpu}, et(b_{jpu}) \leq (cp(c_{jpu}) - \sum_{j=1}^{N_{jpu, c_{jpu}}} et(b_{jpu}, c_{jpu})) \).

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 \( cdcs \).
that the total number of jobs is much greater than the total number of mdc and cdc, so $y \gg w$. Therefore, the above equation evaluates to $\sim O(y^2)$.

Algorithm 1 RT-SANE

1: procedure RT-SANE
2: Compute quantities $st, ct, cd, \forall i_j \in I, \forall b_j \in B$. 
3: Fill $Q_1$ & $Q_0$ with $t_{pu}$, $t_{sp}$ & $t_{pu}$ jobs. 
4: Order $Q_1$ & $Q_0$ in non-decreasing deadline sequence. 
5: $\forall i_j, b_j$ with tag $t_p$: 
6: if ($C_1$ holds for $m_l$) $\&\&$ ($C_3$ holds for $m_l$) then 
7: schedule $i_j$ on $m_l$. 
8: else 
9: try job submission in future. 
10: end if 
11: if ($C_2$ holds for $c_{ip}$) $\&\&$ ($C_4$ holds for $c_{ip}$) then 
12: schedule $b_j$ on $c_{ip}$. 
13: else 
14: try job submission in future. 
15: end if 
16: $\forall i_j, b_j$ with tags $t_{sp}$ or $t_{pu}$: 
17: if ($C_1$ is met on $m_l$) $\&\&$ ($C_3$ is met on $m_l$) then 
18: schedule $i_j$ on its local mdc $m_l$. 
19: end if 
20: if ($C_2$ is met on $c_{ipu}/c_{ip}$) $\&\&$ ($C_4$ is met on $c_{ipu}/c_{ip}$) then 
21: schedule $b_j$ on its local cdc $c_{ipu}/c_{ip}$. 
22: end if 
23: if ($C_1$ is met on $m_f$) $\&\&$ ($C_3$ is met on $m_f$) then 
24: schedule $i_j$ on a remote mdc $m_f$. 
25: end if 
26: if ($C_2$ is met on $c_{fpu}/c_{fp}$) $\&\&$ ($C_4$ is met on $c_{fpu}/c_{fp}$) then 
27: schedule $b_j$ on a remote cdc $c_{fpu}/c_{fp}$. 
28: end if 
29: Compute quantities $DC, UT, \forall m, c \in M, C$. 
30: 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 $c_{ip}$ or local public cdc $c_{ipu}$. 
- 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, $m_l$, $c_{ip}$ are trusted, $c_{ipu}$ is semi-trusted, and $m_f$, $c_{fp}$ 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 (\(D_u\)). 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 → processor → 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 total number of tasks that were reviewed for scheduling. Formally, \(SR = \frac{\sum y}{\sum y} \times 100\).

**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 \(d(i_j) = ett(i_j)\). 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 \textit{mdcs}, and 20\% are batch jobs executing on the \textit{cdc}. Specifically, we observe the impact of the following metrics on the \textit{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.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig3.png}
\caption{Effect of DF on SR}
\end{figure}

The communication delay between a device & an mdc is set as 5 milliseconds, and between a device & the \textit{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. \textit{SR} = 100\%. We also observe that \textit{RT – SANE} offers higher SR values than \textit{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 \textit{RT – SANE} tries to execute the interactive jobs on the \textit{mdcs} and the batch jobs on the \textit{cdc}. On the other hand, in the \textit{cdc – only} algorithm, all jobs, whether interactive or batch are sent to the \textit{cdc} for execution. This is detrimental to the interactive jobs, as owing to the large communication delays between the device & the \textit{cdc}, they would not be able to meet their “tight” deadlines. Therefore, the SR values for \textit{RT – SANE} are higher than those for \textit{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 \textit{mdcs}, and a delay of 105 milliseconds from users to the \textit{cdc}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig4.png}
\caption{Effect of TL on SR}
\end{figure}

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 \textit{RT – SANE} & \textit{cdc – only}.

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

In the third simulation, we observe the effect of Delay Factor (DLF) on the \textit{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 \textit{mdcs} has been taken as 5 milliseconds and delay from the users to the \textit{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 (\textit{st}) at the \textit{mdcs} & the \textit{cdc} increases. This, in effect, causes the completion time of jobs (\textit{ct}) to exceed their deadlines (\textit{d}). Hence, the SR value reduces for increase in DLF values.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.png}
\caption{Effect of DLF on SR}
\end{figure}

However, \textit{RT – SANE}, owing to the employment of \textit{mdcs}, is able to ensure that more jobs finish before their deadline. Hence, it’s SR values are higher than those offered by \textit{cdc – only}, which does not employ \textit{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 mdc for scheduling jobs, in addition to the cdc. Both algorithms differ in the way that the tight deadline jobs are sent to the mdc. 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 mdc on the basis of mdc’s 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 mdc has 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 mdc 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 its 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 mdc, 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 meet 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 mdc. Ultimately, the sum of execution costs of all jobs becomes greater than the execution capacities of the mdc. 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 its 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

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 mdc 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 \textit{mdcs} on Performance

Next, we investigate the effect of varying the number of \textit{mdcs} on the system performance (SR).

The results of this study are shown in figure 9. As the number of \textit{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 \textit{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 \textit{RT – SANE} demonstrates a higher SR value as compared to \textit{cdc – only}. This is because of the fact that \textit{RT – SANE} employs \textit{mdcs} in addition to the \textit{cdc} for job execution. Expectedly, increasing the number of \textit{mdcs} will not have any impact on SR for \textit{cdc – only} as the algorithm does not employ \textit{mdcs} for job execution. Figure 10 shows the same simulation, this time comparing \textit{RT – SANE} with iFogStor. Both these algorithms use \textit{mdcs}, in addition to the \textit{cdc}, for job execution. Once again, \textit{RT – SANE} shows higher SR values, owing to the superior real-time heuristic of earliest deadline first, that it employs.

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. \textit{RT – SANE} performs better than \textit{cdc – only}, as in the latter, all jobs (even interactive) are sent to the \textit{cdc}. Hence, the performance is poor. \textit{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.

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

6.6 Effect of Security mix on Performance

In this section we study the effect of security tag assignment on the performance of \textit{RT – SANE}, iFogStor & \textit{cdc – only}. Specifically, we consider three different security mixes:
- Security Mix 1: the number of private (\textit{tp}) & semi-private (\textit{tsp}) 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.

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.

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.

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.
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 mdc1 (Sm1), one of the mdcs is slow, i.e. has a low MIPS value. In case Sm2, two mdcs are slow. Likewise, case Sm3 and Sm4 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 Sm1 to Sm4, 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>
<thead>
<tr>
<th>Case</th>
<th>SR(RT – SANE)</th>
<th>SR(RT – SANE(JM))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sm1</td>
<td>83.3%</td>
<td>93%</td>
</tr>
<tr>
<td>Sm2</td>
<td>72</td>
<td>81.3</td>
</tr>
<tr>
<td>Sm3</td>
<td>72.5</td>
<td>76</td>
</tr>
<tr>
<td>Sm4</td>
<td>67</td>
<td>71</td>
</tr>
</tbody>
</table>

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 cdcs respectively. Semi private interactive and batch jobs are sent to the foreign mdc and private cdcs respectively. Finally, public interactive and batch jobs are executed at the foreign mdc and foreign public cdcs 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 cdcs – 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
