SenSchedule: Scheduling Heterogeneous Resources in Sensor-Cloud Infrastructure

Sunanda Bose, *Member, IEEE,* Nandini Mukherjee, *Senior Member, IEEE*

Abstract—In a sensor-cloud infrastructure scheduling and allocation of resources are challenging tasks. Unlike computational cloud infrastructure, sensor may have varying capabilities and contexts of services. Moreover, some sensors may be available for a fixed time duration periodically. Depending on some dynamic factors, performance of a sensor may also vary. Such non-uniform performance may be predicted from previous observations. This paper proposes novel scheduling algorithms for heterogeneous resources having varying capabilities, contexts of usage and non-uniform performance over time. Simulation results show that the algorithms perform efficiently.

Index Terms—Sensor Cloud, IoT, Virtual Sensors, Resource Model, Selection, Allocation

1 Introduction

With the advent of sensor, cloud and communication technologies, it has been possible to communicate with remotely located sensors from applications hosted inside cloud and vice versa. In order to capture the semantics of functioning of heterogeneous sensors, in our earlier work [1], we defined context and capabilities of sensors. Sensors can have different sensing contexts, such as healthcare and environment. They can also have varying parameters describing their capabilities, such as "a temperature sensor measures from 90^0F to 110^0F" or "a humidity sensor has a detection range from 5% to 95% RH (relative humidity)". Thus, a pool of such sensors can create a heterogeneous inventory of resources which can be made available for the use of a Sensor-Cloud Infrastructure (SCI). The resources can be provisioned like computing resources available at the IaaS layer of a computational cloud. Sensor-owners can provision services to the applications running on SCI. Earlier we proposed an architecture of SCI to incorporate remotely located sensing resources as citizens of generic resource family, like cpu and memory [2]. Thus, instead of confining a sensor for a specific application, an open system is proposed, where the same sensor can be used by multiple applications as long as its context and capabilities match with their sensing requirements.

Sensing resources may be available periodically for a fixed duration. For example, pollution sensing can be done by sensors mounted on the cars parked in a designated area. A car is parked periodically for a certain duration. Hence, these sensors are available periodically only for a certain length of time. Performance of sensors may also vary with time [3] [4]. Often applications need data from multiple sensors in order to satisfy their requirements. All these contribute to non-uniform spatio-temporal distribution of sensing resources based on availability, applicability, appropriateness and sensing performance. It is therefore necessary to fulfil the requirements of any sensor-aware application by provisioning one or multiple sensors with appropriate context, capabilities and performance from the available resource pool. It should also be noted that the requirements of multiple applications running simultaneously on an SCI must also be handled. The task of scheduling sensing resources from the pool of heterogeneous sensors to requesting applications gives rise to a decision problem of mapping sensing resources to sensing requests. A resource manager in computational cloud schedules computing resources to the requesting applications from a resource pool. Similarly in SCI, sensing resources are to be scheduled and allocated from the resource pool as shown in Figure 1. However, for efficient mapping, a model is required for the entire scenario. This paper addresses the issues related with modeling of resources and the resource requests in order to appropriately schedule the resources to requests. In an SCI, bottleneck situation arises when large number of consumer applications generate requests for few sensing resources. On the other hand, in a participation based cloud [5], where different resource providers may contribute sensing resources, different revenue generation models can exist. In all such cases, the cloud should offer methodologies to minimize resource starvation, so that all participants get fair chance.

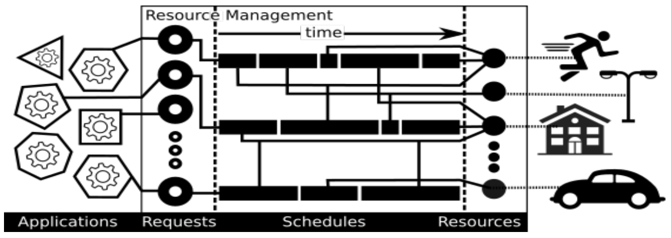


Fig. 1: Resource Management System

The key contributions of this paper are as follows: 1) proposing a generalised model for sensing resources and application requests, 2) developing an approach for scheduling sensing resources to applications considering availability, appropriateness and performance of the resources, and 3) presenting modified approaches to minimize resource starvation and to make the system fault tolerant. The paper is organized as follows: Section 2 describes an

• *S. Bose, and N. Mukherjee are with the Department Computer Science and Engineering, Jadavpur University Kolkata, India.*
E-mail: sunanda.bose@msn.com, nmukherjee@cse.jdvu.ac.in

TABLE 1: Symbols

ω	type of the resource (e.g. temperature, humidity)
Ω	set of all resource types
λ	context of the resource (e.g. health, environment)
Λ	set of all resource contexts
γ	a matrix of contextual parameters where row i is a range (min, max) of contextual parameter γ_i^λ
γ^λ	vector of parameters supported by the context γ
μ	a matrix of resource capabilities where row i is a range (min, max) of capability μ_i^ω
μ^ω	vector of capabilities for resource type ω
$T = (s, \delta, \kappa)$	availability pattern of a resource where s, κ, δ are the start time, duration and recurrence respectively
t_s	beginning time of a request
t_e	ending time of a request
$M_{m\times n}$	matrix having m rows and n columns
R	set of all resources
Q	set of all requests
$R_i(T)$	availability pattern (s, κ, δ) of a resource R_i
$Q_k(t)$	time slot (t_s, t_e) of a request Q_k
$\mathcal{L}_i$	set of performance partitions of resource R_i
$\overleftarrow{\mathcal{L}_{ij}}$	beginning of j^{th} partition of R_i
$\overrightarrow{\mathcal{L}_{ij}}$	ending of j^{th} partition of R_i
$\lvert\mathcal{L}_{ij}\rvert$	size of the j^{th} partition of R_i ($\overrightarrow{\mathcal{L}_{ij}} - \overleftarrow{\mathcal{L}_{ij}}$)
$F_{i,j}$	feasibility of R_i for Q_j
$W_{i,j}$	metric of appropriateness between R_i and Q_j
$P_{i,j,k}$	Favourability of j^{th} partition of R_i for Q_k
$H_{ij\to pq}$	Logistic function to model migration from j^{th} partition of R_i to the q^{th} partition of R_p.
ϵ	maximum switch over cost for resource migration.

SCI infrastructure. Section 3 proposes a model to describe the characteristics of resources. In Section 4 three scheduling strategies, compact, balancing and fault tolerant are discussed. Experimental results are presented in Section 5. In Section 6 we present our study on related works. Section 7 concludes the paper.

2 Sensor-Cloud Infrastructure

A cloud is essentially a pool of resources which are leased by consumers in advance or on demand basis. As our resources have varying *contexts* and *capabilities*, requests for resources must correspond to the diversities of resource family. Sensing resources specific to healthcare are tagged with health sensing context with *contextual parameters* for identifying its attachment, i.e. patient id. Environment sensors can have a different context and may include its location as contextual parameter. Models of the resources and requests are defined in Tuples 1a and 1b respectively. Symbols are shown in Table 1 along with their definitions. Sensing context is also specified in resource request. Only those resources which match with the context specified in request can be selected to satisfy the request.

$$\{\omega \in \Omega, \lambda \in \Lambda, \gamma = M_{2\times|\gamma^\lambda|}(\mathbb{Z}), \mu = M_{2\times|\mu^\omega|}(\mathbb{Z}), T\} \tag{1a}$$

$$\{\omega, \lambda, \gamma, \mu, (t_s, t_e)\} \tag{1b}$$

Different sensors have different sensing capabilities, such as *sensitivity*, *accuracy* etc. A capability is defined as a pair of minimum and maximum values. Vector μ^ω (Table 1) defines capabilities of resource type $\omega \in \Omega$, of which each element is a pair of minimum and maximum values. Thus, it is a matrix of two columns and $|\mu^\omega|$ rows as shown below. Due to the mobility and other reasons, unlike computing

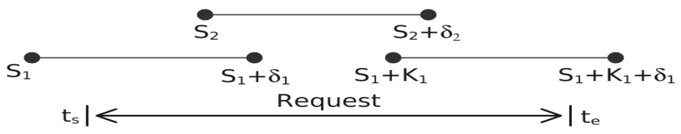


Fig. 2: Two resources having different availability slots serving a request for $t_s \to t_e$

resources, sensing resources may be available periodically. The availability pattern is expressed as a tuple of s (start time), δ (duration) and a recurrence $\kappa > \delta$. Resource i appears at every $s_i + n\kappa_i$ and serves till δ_i time unit $\forall n \in \mathbb{Z}^+$. Time can be represented as unix time stamp *.

A resource request has one time span defined as (t_s, t_e) indicating the start and end times of the request time span as shown in Figure 2. A single resource may not be available during the requested time span. In such scenario multiple physical resources (shown as S_1, S_2 in Figure 2) may be leased in order to provide uninterrupted sensing service. Sensing services are provided to consumers as virtual sensors abstracting sensing time of one or more physical sensor(s) which are leased in response to a request during the resource selection phase.

Once a resource is available, its performance may vary over time (e.g. due to instability in transmission network). In some previous experiments [3], it was observed that throughput of mobile sensors vary greatly while passing through tunnels in road, or even in busy hours when network load is high. However as the resources are leased for multiple times, we may observe its throughput pattern and avoid allocating requests to the resources in different times or localities (in case of mobile sensors) when their throughput is expected to be low.

So to satisfy a request, we need to search for a set of appropriate resources that match best with the requested *capabilities* and *context*. At the same time the selected resources need to be available during the time span of the request. Further, to provide a good sensing experience we may prefer switching to a better resource when the currently allocated resource cannot perform efficiently or has some fault. But too much switching over may cause delay, So decisions are to be taken to minimize the delay, while maintaining the required efficiency of the sensing services. On the other hand, if a large number of requests are served by small number of resources, then dependency on the sensing resources increases. If one such sensing resource fails, it will impact a large number of requests. Moreover, other resource providers will starve, thereby loosing motivation in business in a participation based sensor cloud. So to balance resource sharing, it is also important to minimize resource dependency and starvation. In the subsequent sections we move towards solution of these problems one by one.

3 Modeling Resource Characteristics

To solve the problems discussed in the previous section we first address heterogeneity and select resources that are feasible to satisfy the request. However all resources of the

*. e.g. a resource available every 2 to 11PM since Jan 1 2017 (UTC) may be represented using $T = \{s : 1483279200, \delta : 32400, \kappa : 86400\}$.

same type (ω) are not uniformly favored. Some are more favored than the others due to varying capabilities. So we create an appropriateness matrix between resources and requests where a higher value implies that a resource is more favorable for the request. As the sensing resources may not perform uniformly we also create a performance vector for each resource based on the past observations.

Heterogeneity

In order to allocate a request, a set of feasible resources are selected that match with the type and capability required by the request. Each capability parameter $\mu_i^\omega = \{\mu_{i,min}^\omega, \mu_{i,max}^\omega\}$ is a pair of min and max values, $\forall \mu_i^\omega \in \mu^\omega$, as shown in Example 3.1. While matching, in order to reduce the computational requirement, we create a pair of scalar values representing the minimum and maximum bounds. Instead of matching all parameters in capabilities matrix μ^ω ,we check against that pair of minimum, maximum values. We represent this as a transformation of μ^ω to $\hat{\mu}^\omega$, where $\hat{\mu}^\omega = (\xi \downarrow, \xi \uparrow)$ is represented by two scaler encodings obtained from all minimum and maximum bounds as described below.

Example 3.1.

$$\mu^{sensT} = \begin{bmatrix} 0 & 100 \\ 0 & 1 \\ 2 & 2 \end{bmatrix}$$

The above is an example of a temperature sensor capable of sensing 0 to 100°C, with an accuracy of 0 to 1 degrees, and transmitting twice in a minute (or in a unit time).

We assign unique prime number ρ_i to each capability parameter $\mu_i^\omega \in \mu^\omega$. For most of the capability parameters the constraints are: $R(\mu_{i,min}^\omega) \leq Q(\mu_{i,min}^\omega)$ and $R(\mu_{i,max}^\omega) \geq Q(\mu_{i,max}^\omega)$, where R, Q denote capability of resource and request respectively.

We use the prime number property $p^v \, mod \; p^u = 0$ **iff** $v \geq u$ where $u, v \in \mathbb{Z}^+$ and p is a prime number. Similar prime number based approach has been used in [6] for IO redirection. Formulation of $\xi \downarrow, \xi \uparrow$ are shown in Equation 2. For a resource (R) to be feasible for request (Q), we need $R(\xi \uparrow)$ to be divisible by $Q(\xi \uparrow)$ and $Q(\xi \downarrow)$ to be divisible by $R(\xi \downarrow)$.

However all capability parameters cannot be modeled in the similar manner. In Example 3.1 capability parameter accuracy is used. Unlike sensitivity, a request requiring sensor accuracy of ±2 cannot be allocated to a resource having accuracy ±5, however the opposite is true. We use the term *restrictive* to denote such type of parameters. To model the restrictive parameters we keep another flag $s_i \in \{0, 1\}$ which is 1 for restrictive parameters. We model another pair of scalars that preserve only the restrictive parameters in $(\zeta \downarrow, \zeta \uparrow)$ as shown in Equation 3.

$$\xi \downarrow = \prod_{i=1}^{|\mu^\omega|} \rho_i^{\mu_{i,min}^\omega} \qquad \xi \uparrow = \prod_{i=1}^{|\mu^\omega|} \rho_i^{\mu_{i,max}^\omega} \tag{2}$$

$$\zeta \downarrow = \prod_{i=1}^{|\mu^\omega|} \rho_i^{s_i \mu_{i,min}^\omega} \qquad \zeta \uparrow = \prod_{i=1}^{|\mu^\omega|} \rho_i^{s_i \mu_{i,max}^\omega} \tag{3}$$

$$where \; \rho_i \in \mathbb{P} \; is \; the \; {}^i th \; prime \; and \; \mathbb{P} \; is \; the \; set \; of \; primes$$

We combine ξ and ζ in Equation 4 to formulate feasibility of a resource w.r.t. a request. As ζ contains prime powered product of the restrictive parameters only, we cancel out by dividing ξ by ζ. Then we swap the restrictive parameters of resource and requests as shown in Example 3.2.

$$\begin{aligned} Q(\xi \downarrow)\frac{R(\zeta \downarrow)}{Q(\zeta \downarrow)} \; mod \; R(\xi \downarrow)\frac{Q(\zeta \downarrow)}{R(\zeta \downarrow)} = 0 \\ R(\xi \uparrow)\frac{Q(\zeta \uparrow)}{R(\zeta \uparrow)} \; mod \; Q(\xi \uparrow)\frac{R(\zeta \uparrow)}{Q(\zeta \uparrow)} = 0 \end{aligned} \tag{4}$$

Example 3.2.

$$\begin{bmatrix} \alpha_r & \alpha'_r \\ \beta_r & \beta'_r \\ \gamma_r & \gamma'_r \end{bmatrix} \begin{bmatrix} \alpha_q & \alpha'_q \\ \beta_q & \beta'_q \\ \gamma_q & \gamma'_q \end{bmatrix} \begin{array}{c|cc|cc} & \xi \downarrow & \xi \uparrow & \zeta \downarrow & \zeta \downarrow \\ \hline R & \rho_1^{\alpha_r} \rho_2^{\beta_r} \rho_3^{\gamma_r} & \rho_1^{\alpha'_r} \rho_2^{\beta'_r} \rho_3^{\gamma'_r} & \rho_3^{\gamma_r} & \rho_3^{\gamma'_r} \\ Q & \rho_1^{\alpha_q} \rho_2^{\beta_q} \rho_3^{\gamma_q} & \rho_1^{\alpha'_q} \rho_2^{\beta'_q} \rho_3^{\gamma'_q} & \rho_3^{\gamma_q} & \rho_3^{\gamma'_q} \end{array}$$

Given a resource r and a request q having capabilities denoted as shown in the first two matrices from left, the right-hand side table is the ξ and ζ as calculated through Equations 2 and 3 assuming the last parameter γ is restrictive. When we apply Equation 4 we actually swap the last parameter as shown below.

$$Q(\xi \downarrow)\frac{R(\zeta \downarrow)}{Q(\zeta \downarrow)} = \rho_1^{\alpha_q} \rho_2^{\beta_q} \rho_3^{\gamma_r} \; , R(\xi \downarrow)\frac{Q(\zeta \downarrow)}{R(\zeta \downarrow)} = \rho_1^{\alpha_r} \rho_2^{\beta_r} \rho_3^{\gamma_q}$$

$$R(\xi \uparrow)\frac{Q(\zeta \uparrow)}{R(\zeta \uparrow)} = \rho_1^{\alpha'_r} \rho_2^{\beta'_r} \rho_3^{\gamma'_q} \; , Q(\xi \uparrow)\frac{R(\zeta \uparrow)}{Q(\zeta \uparrow)} = \rho_1^{\alpha'_q} \rho_2^{\beta'_q} \rho_3^{\gamma'_r}$$

Therefore Equation 4 is satisfied only if the following inequalities hold, because p^v is divisible by p^u if and only if $v \geq u$ where $u, v \in \mathcal{Z}^+$ and $p \in \mathbb{P}$ is a prime number.

$$\alpha_r \leq \alpha_q, \quad \beta_r \leq \beta_q, \quad \gamma_r \geq \gamma_q, \quad \alpha'_r \geq \alpha'_q, \quad \beta'_r \geq \beta'_q, \quad \gamma'_r \leq \gamma'_q$$

However, instead of checking all the above-mentioned inequalities, we perform a single divisibility check as shown in Equation 4. Also ξ and ζ for resources can be computed at the time of resource registration using Equations 2 and 3. So we do not need to recompute this every time when we need to allocate resources for a request.

Thus the proposed capability model may include any property that can be expressed using a pair of non-negative integers and *restrictiveness* flag.

Appropriateness

Our objective is to select different consecutive time slots of most appropriate resources that cover the requested time frame while minimizing cost and maximizing performance. In order to do that we need to select resources which are feasible to satisfy the request. So we create a binary *Feasibility Matrix (F)*. However all feasible resources are not equally favorable for a request. Resources which deviate less from the specifications in the requests are more preferred than the others. So we create a *Distance Matrix (W)*. The construction of these two matrices are shown below. We obtain appropriateness matrix (A) based on F and W as shown in Equation 7.

Feasibility Matrix(F): (binary matrix): Capability and contextual parameter matrices are scalarized when the resources join the infrastructure or update their details (which do not happen often). Based on the divisibility of scalarized min and max values we define the feasibility matrix F as described in the Equation 5. If resource j is feasible for request i, $F_{i,j}$ is set to one.

$$F = M_{|R| \times |Q|}$$

$$F_{i,j} = \begin{cases} 1 & if \; Q(\omega) = R(\omega) \\ & \wedge \; Q(\lambda) \; = R(\lambda) \\ & \wedge \; eq \; 4 \; holds \\ 0 & otherwise \end{cases} \tag{5}$$

Distance Matrix(W): (metric space): On the other hand, one request may be feasible for multiple resources. However, each of them may not be appropriate; e.g. to satisfy a request for temperature sensor with sensitivity of 50°C to 100°C we should try selecting resources with ranges closest to the bounds specified in the request's capability matrix. Thus, if there are two sensors, one with a sensing range of 0°C to 100°C and the other with sensitivity of 40°C to 100°C, the latter should be more preferred. Without a metric of appropriateness, sensing services will depend only on a few sensing resources having large range of capability parameters, and rest of the resources will starve.

Capabilities of resources and requests can be expressed in an euclidean space. We can use euclidean distance as a distance function to calculate the favourability of a resource for a request. However, this would require the distance to be calculated on every new request. Although computation of euclidean distance is trivial, if the number of requests is high and many resources with various capabilities are available, computational cost for calculating distances may not be negligible. Therefore, we first compute a scalar representing capability matrix. While comparing a resource's capability with that of a request, we compare these scalars instead. We maintain a distance matrix as shown in Equation 6

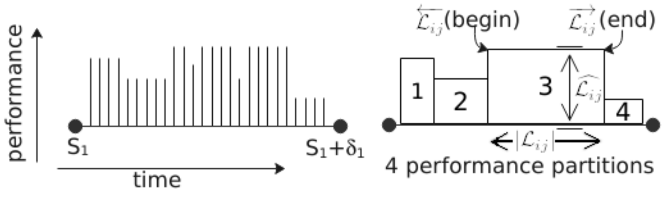
$$\begin{aligned} W &= M_{|R|\times|Q|} \\ W_{i,j} &= \begin{cases} \text{inf} & if\ F_{i,j} = 0 \\ \frac{R(\xi\uparrow)Q(\xi\downarrow)}{R(\xi\downarrow)Q(\xi\uparrow)} \left(\frac{R(\zeta\downarrow)Q(\zeta\uparrow)}{R(\zeta\uparrow)Q(\zeta\downarrow)}\right)^2 & otherwise \end{cases} \end{aligned} \tag{6}$$

Based on these two matrices we compute A_{ik} as appropriateness of resource i to satisfy request k as shown in Equation 7.

$$A_{ik} = \frac{F_{ik}}{W_{ik}} \tag{7}$$

Performance Partition

A continuous time slot (t_s, t_e) is specified for a request $q \in Q$. However a single resource may not be available during the entire time slot. The resource manager is supposed to multiplex multiple resources with availability slots which are subsets of the requested time span to power the virtual sensor. On the other hand varying performance of a sensor may be observed during an availability slot. Depending on communication methodologies, environmental hazards, signal-to-noise ratio, usage pattern etc. a sensor may perform differently at different times impacting its transmission. This leads to increase or decrease in the number of sensed data at different times. During the process of resource selection, we try to favor a sensor when it has better performance (e.g. transmits more sensed data) than others. Once a resource is leased multiple times, we can predict a time varying performance expectation by observing its past performances. The observed performance of a sensing resource is used to find a set of non-uniform time intervals, each with different expected throughput as shown in Figure 3. A partitioning process partitions the observed performance which is shown on the left to non-uniform performance partitions as shown on the right. These performance partitions are the input to the scheduling process. There have been many works on traffic prediction [7] [8] [9] [10]. However, traffic prediction and the partitioning process are not within the scope of this work. In this work throughput is considered as its performance. But, any time varying property can be incorporated within the scheduling process through partitioning. After partitioning, we get a vector $\mathcal{L}_i$ for each sensor S_i. Each



performance
time
S_1
$S_1+\delta_1$
$\overleftarrow{\mathcal{L}_{ij}}$(begin)
$\overrightarrow{\mathcal{L}_{ij}}$(end)
1
2
3
4
$\widehat{\mathcal{L}_{ij}}$
$|\mathcal{L}_{ij}|$
4 performance partitions

Fig. 3: Performance Partitions

such partition in $\mathcal{L}_i$ is a set of start time ($\overleftarrow{\mathcal{L}_{ij}}$), end time ($\overrightarrow{\mathcal{L}_{ij}}$), and expected approximate throughput ($\widehat{\mathcal{L}_{ij}}$). We use the symbol $|\mathcal{L}_{ij}| = \overrightarrow{\mathcal{L}_{ij}} - \overleftarrow{\mathcal{L}_{ij}}$ to denote the length of a partition. To satisfy the entire request we need a set of partitions that cover the requested time frame. These partitions may overlap each other, however, if there is a gap between two partitions there will be no service available during that time period. We schedule partitions having high appropriateness and better performance. However, switching from one partition to another may cause resource migration if the source and destination partitions are from two different resources. Consecutive partitions of the same sensor is a preferred option, however if another sensor provides much better appropriateness and performance, we perform a migration from one sensor to the other. Frequent switching over may result in service delays. Also a hop (switch over) to a longer partition is less costly than a hop to a shorter partition. In order to minimize switch over we use a *parachor curve* as a logistic function to determine the hop cost. ϵ is the maximum switch over cost.

$$H_{ij\to pq} = \begin{cases} \begin{cases} 0 & p = i \\ \epsilon\, e^{-f|\mathcal{L}_{p,q}|} & \text{otherwise} \end{cases} & b_{pq} \le e_{ij} \\ \infty & \text{otherwise} \end{cases} \quad \forall\ k \tag{8}$$

Hop cost as modeled in Equation 8 is described as follows. $H_{ij\to pq}$ is the cost of reserving partition q of resource p after reserving partition j on resource i, where $|\mathcal{L}_{p,q}|$ is size of the destination partition and f determines steepness of the curve. For our experiments we have used 0.02 as the value of f.

As we are scheduling performance partitions, we need to compute favourability of the partitions in terms of their appropriateness and performance. P_{ijk} shown in Equation 9 measures how favorable is the partition j of resource i for request k. We want to select a set of partitions such that the total favorability is maximized and total hop cost is minimized.

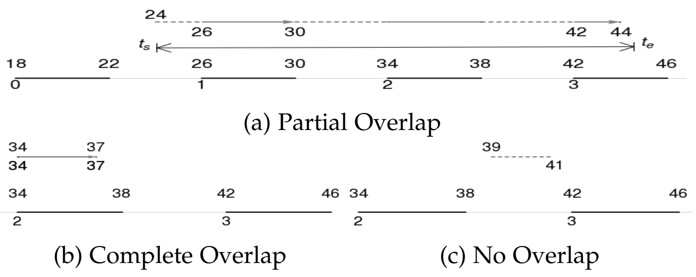
$$P_{ijk} = A_{ik}\widehat{\mathcal{L}_{ij}}|\mathcal{L}_{i,j}| \quad \forall\ i,j,k \tag{9}$$

Competition

However if the same partition is selected for too many requests we will have more number of consumers dependent on less number of resources. Small devices like sensors are more prone to errors. Also if some resource is never allocated for any request in spite of being feasible it will cause resource starvation. This resource starvation may discourage the resource provider to participate in the sensor cloud infrastructure. Thus we need to balance the load in such a way that number of dependent consumers is inversely proportional to its number of competitors. A performance partition having no competitors implies the resource is available in such a time span when no other resource is available. In such cases that partition has to be shared with all requests as number of competitors is 0. Otherwise we have a group of resources available in the same time span creating a competitors group.

Competition of i^{th} resource's p^{th} performance partition on m^{th} availability instance w.r.t. the k^{th} request is defined by $C_i^k(m,p)$ as shown in Equation 10. A resource may not compete with all resources in the inventory, i.e. all resources in the inventory may not be feasible for request k. Thus competition of resource i is determined against a subset of resources $R^k \subset R \setminus \{R_i\}$. To check feasibility we use the feasibility matrix (F) defined in Equation 5. $R_l(T)$ denotes the availability slot $T = \{s, \kappa, \delta\}$ of l^{th} resource $\in R^k$. Competition is expressed in terms of number of availability slots of other resources that cover the SPAN of p^{th} performance partition on m^{th} availability instance of resource i. SPAN is calculated as a pair of start and end time of that partition on the given instance as shown in Equation 11. $\mathcal{O}(T, ts)$ shown in Equation 12 is a binary function that returns 1 if the given time span (ts) is completely overlapped by some instance of availability slot T.

$$\begin{aligned} C_i^k(m,p) &= \sum_{R_l \in R^k} \mathcal{O}(R_l(T), rs) \\ rs &= \text{SPAN}(R_i, m, p) \\ R^k &= \{R_t \in R \mid F_{t,k} = 1\} \end{aligned} \tag{10}$$



(a) Partial Overlap

(b) Complete Overlap

(c) No Overlap

Fig. 4: Time spans overlapping with T

$$\text{SPAN}(R_i, m, p) = \{s_i + m\kappa_i \ , \ s_i + m\kappa_i + \delta_i\} \tag{11}$$

$$\mathcal{O}(T, ts) = \begin{cases} 1 & OverlapsF(T, ts).c = 1 \\ 0 & \text{else} \end{cases} \tag{12}$$

An example of such complete overlap is shown in Figure 4b between time span $\{t_s : 34, t_e : 37\}$ and availability slot $T = \{s : 18, \delta : 4, \kappa : 8\}$. Two other time spans $\{t_s : 24, t_e : 44\}$ and $\{t_s : 39, t_e : 41\}$ have been shown in Figure 4a and 4c. These time spans may be of some other resource partition that is feasible to serve the same request. As we can see in Figure 4 one or more instances of a resource availability slot may be used to cover a given time span. We use a, b to denote the index of first and last resource instance of an availability slot that the given time span can use. In the partial overlap scenario shown in Figure 4a $a = 1, b = 3$. Similarly in Figure 4b $a = b = 2$ as service starts and ends in the second service instance of the resource's availability slot. In the No overlap scenario no instances of the resource can be used to satisfy the given time span. We obtain coverage of a resource availability slot w.r.t. a given time span through OVERLAPSF as shown in Algorithm 1.

Algorithm 1 OVERLAPSF($T = \{s, \kappa, \delta\}$, $ts = \{t_s, t_e\}$)

```
1: $\lfloor \_ \rfloor^+ := \frac{1}{2}(\_ + |\_|)$        ▷ operator $\mathbb{Z} \to \mathbb{Z}^+ \iff \_ > 0$
2: $a_s \leftarrow \lfloor \frac{t_s - (s+\delta)}{k} \rfloor + 1$;      $a_e \leftarrow \lfloor \frac{t_e - s}{k} \rfloor$
3: if $a_s \neq a_e$ then return {-1,-1,0} end if
4: $b \leftarrow \lfloor \frac{t_e - s}{k} \rfloor$; $a \leftarrow a_s$
5: $\theta \leftarrow$ MIN($\lfloor (s + ak) - t_s \rfloor^+, t_e - t_s$)
6: $\phi \leftarrow$ MIN($\lfloor t_e - (s + bk + \delta) \rfloor^+, t_e - t_s$)
7: $c \leftarrow \frac{\theta + \phi + (n-m)(k-d)}{t_e - t_s}$
8: return $\{a, b, c\}$
```

OVERLAPSF as shown in Algorithm 1 takes availability pattern $T = \{s, \kappa, \delta\}$ of a resource and a time slot (t_s, t_e). If there is any overlapping, the function finds the first and last availability instances of the resource. As a, b are the first and last availability instances, the recurrence relations in Equations 13 holds. We compute euclidean division w.r.t. t_s and t_e in order to find a_s, a_e respectively. Unless $t_e < s$ or the time span $\{t_s, t_e\}$ lies in the gap (e.g. Figure 4c) of two instances, $a_s = a_e$ should hold. So if that condition does not hold that time span never overlaps with the input T, and we return 0 coverage and invalid (-1) instance indexes.

$$s + (a-1)k + \delta < t_s < s + ak + \delta \tag{13a}$$
$$s + ak < s + bk < t_e < s + (b+1)k \tag{13b}$$

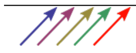
θ, ϕ are the terminal left over of the span that are either beyond a^{th} or b^{th} instance. To discard the negative values we define an operator that yields 0 if the input is < 0, otherwise returns the input. Then we use that to compute the coverage as c.

The OVERLAPSF function is generic and none of its arguments are necesssarily tied up with a resource or request. Given any periodic pattern and time interval it can compute $\{a, b, c\}$.

TABLE 2: Resources

ID	AVAILABILITY	PERFORMANCE
1	$\{s : 10, \delta : 60, \kappa : 80\}$	$\begin{bmatrix} 0 & 15 & 15 & 100 \\ 15 & 20 & 5 & 5 \\ 20 & 40 & 20 & 120 \\ 40 & 60 & 20 & 80 \end{bmatrix}$
2	$\{s : 20, \delta : 50, \kappa : 100\}$	$\begin{bmatrix} 0 & 15 & 15 & 50 \\ 15 & 20 & 5 & 200 \\ 20 & 50 & 30 & 150 \end{bmatrix}$
3	$\{s : 65, \delta : 30, \kappa : 110\}$	$\begin{bmatrix} 0 & 15 & 15 & 50 \\ 15 & 30 & 15 & 60 \end{bmatrix}$

TABLE 3: Legends used in Figures

Symbol	Definition
	Performance partitions, differently colored to differentiate from the neighbors, heights correspond to performance of that partition.
★, ♦	the beginning and ending partitions of a resource w.r.t. a request are marked with star and rhombus
	Arrows of any color denote possible hop. Red arrows denote the selected hops

In this section we use ts as the span of a resource partition. However in latter sections we use ts of a request time span.

A competitor's group is formed with a resource partition and its competitor partitions which belong to different service instances of different resources. Partitions having no competition leads to creation of a singleton group with only one element. Also having no competition ($C = 1$) implies it has to be shared with all requests as there are no other resources that serve in the same time span. So competition value is inversely proportional to sharing coefficient. Thus sharing coefficient SHARE(i, m, p) of p^{th} performance partition on m^{th} availability instance of resource i is expressed as sum of inverse of competition of all partitions in the group as shown in Equation 14.

$$\text{SHARE}(i, m, p) = \left\lceil \sum_k \frac{1}{C_i^k(m, p)} \right\rceil \tag{14}$$

Sharing coefficient determines a limit that can be imposed on resources while scheduling. Imposing this limit will result in a balanced schedule, whereas ignoring it will lead to a scenario where more number of requests are satisfied with less number of resources thus compacting the number of resources in use. Although we can achieve compaction by ignoring this constraint, this may result in resource starvation. We discuss the effect of both of these balancing and compaction strategies in next Section 5.1.

4 PROPOSED APPROACHES TO SCHEDULING

We can model our problem like a graph traversal problem where each partition is represented as a vertex and edges represent migration (hop) from one partition to another. Our objective is to compute a continuous path that starts from a partition overlapping with t_s till we reach a partition overlapping with t_e of the request. However while selecting a vertex we need to favor more *appropriate* partitions, but also obey the *competition* constraints. The problem can be visualized in Figure 5. The consecutive partitions are colored differently using different colors, so that they can be distinguished from each other. Heights of the partitions correspond to their appropriateness with respect to a request. Edges represent hop from one partition to another partition. All possible edges are shown in Figure 5. Each partition of resource i is labeled with $X_{i,j}$ where j is the vertex id of that partition. A vertex id is obtained from the instance id and partition id. Figure 5 represents the problem

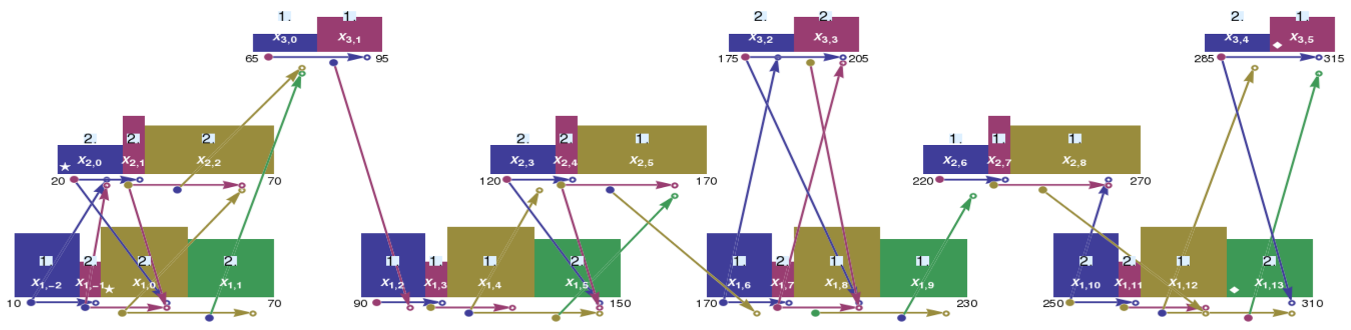
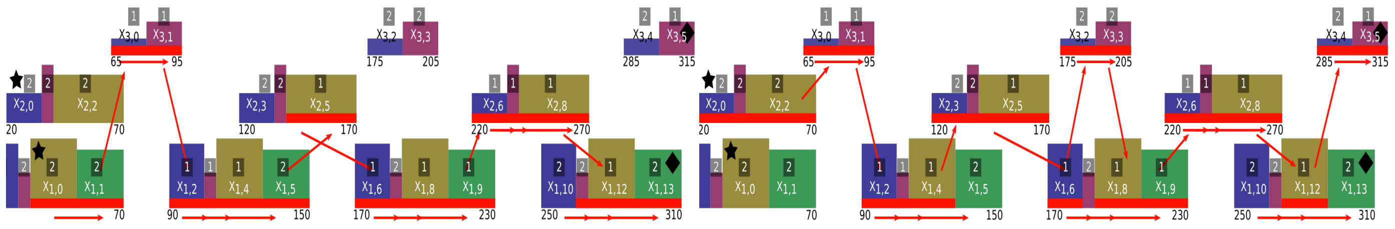


Fig. 5: Single request layer of partition scheduling problem (all possible edges), Legends shown in Table 3



(a) Solution of request layer 1

(b) Solution of request layer 2

Fig. 6: Solution of scheduling partitions (with inappropriate edge position), Legends shown in Table 3

w.r.t. 3 resources and one request only. We have one such layer representing one request. With multiple such layers we label our vertices with $X_{i,j,k}$ where k corresponds to the request. While visualizing our model, we use a set of resources that only differ by availability slot and performance for simplicity. These resources are described in Table 2. Although the resources are of same type and context, appropriateness of partitions differ due to varying performances. Further we consider two requests of same type and context requesting for same time span $\{t_s : 30, t_e : 300\}$. Start and end of partitions are marked with star and rhombus. It may be noticed that resource 3 does not have a starting partition (star) and resource 2 does not have an ending partition (rhombus). This is because no instance of these resources overlap with t_s : 30 and t_e : 300 respectively. Competition (an integer) of each partition is highlighted inside the squares. In Figure 5, some partitions $(X_{3,0}, X_{3,1})$ have no competitors because no other resource availability slots serve in the same time span. So such partitions have to be shared among all requests. On the other hand, other partitions have competition higher than one. This implies that requests can be distributed in such a way that resource starvation is minimized as discussed in Equation 14.

4.1 ILP Formulation

As discussed earlier, Figure 5 visualizes one request layer. To solve the problem for multiple requests we need to have multiple such request layers. Decision variable X_{ijk} shown in Equation 15 represents whether i^{th} resource's j^{th} vertex has been used by the service request k. An edge $Y^k_{ij\to pq}$ shown in Equation 16 indicates whether there exists a hop from i^{th} resource's j^{th} vertex to p^{th} resource's q^{th} vertex. Possible edges in a request layer are shown in Figure 5.

$$X_{ijk} = \begin{cases} 1 & \text{vertex i,j,k is used} \\ 0 & \textit{otherwise} \end{cases} \tag{15}$$

$$Y^k_{ij\to pq} = \begin{cases} 1 & \text{hop from vertex i,j,k to vertex p,q,k} \\ 0 & \textit{otherwise} \end{cases} \tag{16}$$

Based on these decision variables we need to model our constraints such that we get a continuous path from the source vertex to the destination vertex. For each request layer we have a start vertex denoted by $X_{0,0,k}$ connected with all vertices with star. To complete the path we have another vertex $X_{-1,-1,k}$ in each layer that each ending vertex (marked with rhombus in Figure 5) connects to. As we need connecting path in each layer these two request vertices are always used as shown in Equation 17.

$$X_{0,0,k} = 1 \;\; \forall k \qquad X_{-1,-1,k} = 1 \;\; \forall k \tag{17}$$

To make our reservation path continuous we can use continuity constraints which are very similar to ILP formulation of single source single destination shortest path problem [11]. Equation 18 constraints that in-degree of all resource vertices will be same as its out-degree. However for the request vertices shown in Equation 17, there is no incoming edge for vertex $X_{0,0,k}$ and no outgoing edge for vertex $X_{-1,-1,k}$. As these two sets of vertices are to be used in all layers, out degree of request start vertex must be 1, and the same applies for in degree of request end vertex of each layer as shown in Equation 18.

$$\begin{aligned} \sum_{p,q} Y^k_{ij\to pq} &= \sum_{p,q} Y^k_{pq\to ij} \; \forall i,j \\ \sum_{i,j} Y^k_{0,0\to ij} &= 1 \; \forall k \\ \sum_{p,q} Y^k_{pq\to -1,-1} &= 1 \; \forall k \end{aligned} \tag{18}$$

If an edge originated from vertex (i, j, k) is used it implies that vertex (i, j, k) is used as shown in Equation 19.

$$X_{ijk} = \sum_{p,q} Y^k_{ij\to pq} \quad \forall p,q \tag{19}$$

Competition constraint is applied on competing vertices as shown in Equation 20 where j is the vertex id of i^{th} resource's m^{th} instance's p^{th} partition. It may be noted that all other constraints operate at different layers whereas this constraint is applied on the competeting vertices accross the layers. We discuss more on this later in the next few sections.

$$\sum_k X_{ijk} \leq \text{SHARE}(i, m, p) \tag{20}$$

Our objective is to maximize gain by selecting more appropriate vertices while minimizing number of hops as shown in Equation 21. P_{pqk} and $H_{ij\to pq}$ are the favourability of vertex (p,q) and hopcost of migrating from vertex $(i,j) \to (p,q)$ w.r.t. request k as shown in Equation 8 and Equation 9 respectively.

$$Max \sum Y^k_{ij\to pq}(P_{pqk} - H_{ij\to pq}) \tag{21}$$

We solve the ILP with the above mentioned equations. A visualization of the solution is presented on Figure 6. Selected partitions are highlighted with red rectangles. Edges are highlighted with red arrows. As we have two requests, 2 layers are visualized in our solution. First layer shown in Figure 6a is the solution for the first request and Figure 6b displays the solution for the second one. Due to the effect of Equation 20, vertices $X_{1,0}$ and $X_{1,1}$ have been used in the first layer, however in the second layer, vertices $X_{2,0}, X_{2,1}$ and $X_{2,2}$ are used to minimize dependency in that time frame. On the other hand vertices $X_{3,0}, X_{3,1}, X_{1,2}, X_{1,3}$ have competition value of 1. As these vertices have no competitors, all request layers use these vertices.

However entry and exit positions of the edges are arbitrarily drawn, as the equations we have presented so far are not sufficient to determine that. Also this leads to a miscalculation of our objective function. We assume that if a vertex is used it will be used for its entire duration. Thus we multiply appropriateness of a partition with its duration $|\mathcal{L}_{ij}|$ in Equation 9. But if we check any hop shown in Figure 6a (e.g. $X_{1,9} \to X_{2,6}$), if edge $Y_{1,9,2,6,1}$ originates from the end of $X_{1,9}$, vertex $X_{2,6}$ is used partially. Similarly if edge $Y_{1,9,2,6,1}$ reaches $X_{2,6}$ at the beginning of that vertex, then vertex $X_{1,9}$ is partially used. There will be many possibilities based on the entry and exit positions of the edges. Unless we address this issue the solution is actually incomplete. So in the next few sections we try to reformulate our problem differently so that we can also determine suitable entry and exit positions for edges.

4.2 MINLP Formulation

Although we get a set of vertices and a set of edges selected for each request layer, we cannot compute the entry and exit position of the edges. Whenever a vertex is visited by an edge our equations assume that the vertex has to be visited completely till the partition ends. However the time span a request stays on a vertex depends on its entry position of in-edge and exit position of out-edge. In the subsequent sections we try to provide a more precise problem formulation.

Let us have two variables $a_{pqk}, b_{pqk} \in \mathcal{R}^+$ to determine the destination position of in-edge and source position of out-edge for a vertex (p,q,k). If in-edge of vertex (p,q,k) incidents on the beginning of the partition, value of $a_{pqk} = 0$. Similarly if the out-edge of vertex (p,q,k) originates from the end of that partition we have $b_{pqk} = 1$. So these two variables are bound within $[0,1]$ as shown in Equation 22. Though this way we can effectively express entry and exit positions of the edges, we are not dealing with only integers. The problem is now transformed to a mixed integer problem as the variable a and b are real numbers.

$$0 \le a_{pqk}, b_{pqk} \le 1 \tag{22}$$

Based on these two variables we can compute $\Delta_{pqk} = b_{pqk} - a_{pqk}$ as the amount of stay in vertex (p,q,k). Δ_{pqk} is also bound within $[0,1]$. $\Delta_{pqk} = 1$ for a vertex (p,q,k) implies that the vertex have been fully visited. Our objective function also changes to incorporate these variables as shown in Equation 23. As we can observe the objective function turns into an MINLP (Mixed Integer Non Linear Problem) which is comparatively more difficult to solve.

$$\begin{aligned} &Max \sum Y^k_{ij\to pq}(\Delta_{pqk} P_{pqk} - H_{ij\to pq}) \\ \Rightarrow &Max \sum Y^k_{ij\to pq}((b_{pqk} - a_{pqk}) P_{pqk} - H_{ij\to pq}) \end{aligned} \tag{23}$$

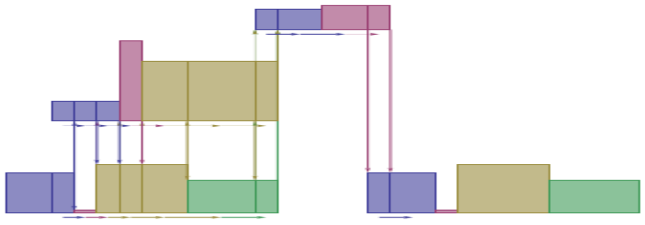


Fig. 7: Slicing Partitions of 3 resources,Legends in Table 3

For continuity of the hops we need the exit position from one vertex to be at the same time as the entry position of that edge in the next vertex as shown in Equation 24.

$$\begin{aligned} &Y^k_{ij\to pq}((b_{ijk}\mathcal{L}_{ij} + A_{ij}) - (a_{pqk}\mathcal{L}_{ij} + A_{pq})) = 0 \\ &A_{ij} = \text{SPAN}(R_i, m, p).begin \\ &\text{m,p are instance and partition of vertex j of } R_i \end{aligned} \tag{24}$$

Due to the inherent complexities of MINLP the solution of the above problem is time consuming. Therefore, we process the inputs differently and remodel the problem with another ILP formulation. The newly formulated problem can be solved in reasonable amount of time which is discussed in the next section.

Remodeling with pre-processing of vertices

In the previous section, formulation of a mixed integer programming problem is considered. However, if we can enforce a hop to start at the end of a partition and end at the beginning of another partition, all visited partitions will be fully utilized. But ending position of a partition is rarely aligned with the beginning position of the next partition as observed in Figures 5 and 6. So we slice the partitions in such a way that beginning position of a slice of the reachable partitions is always aligned with the ending position of a source slice as shown in Figure 7.

With this modification, instead of taking partitions as vertices, the slices are considered as vertices. A hop is performed from the ending position of a slice and leading to the starting position of the destination vertex. So a slice is either fully visited or not visited at all resulting to no partial visits.

Now we need to translate our decision variables shown in Equations 15, 16 and 17 in terms of slices. Same applies for the constraints and objective function shown before in Equations 18, 19, 20 and 21. The algorithms for slicing the partitions are shown below. CREATERESOURCESLICES shown in Algorithm 2 slices resource partitions while using SLICERESOURCE shown in Algorithm 3. Each such slice is represented as a vertex in the graph. We create a directed edge between two slices iff the ending position of the first one is same as the starting position of the second one as shown in Figure 7. Now we explain the algorithms shown below.

CREATERESOURCESLICES considers all resources(R) and resource requests(Q) and returns a matrix of $|R| \times |Q|$ dimension. Each cell in that matrix is a vector that corresponds to resource slices for an instance of the resource. As time span of requests may vary, different resource instances may be appropriate for different requests. Using OVERLAPSF we get a pair of instances starting from a and ending on b. We iterate through $a \cdots b$ and generate the resource slices of each instances by using SLICERESOURCE.

SLICERESOURCE takes a resource R_r and the instance n subjected to be sliced depending on the start and end positions of other resources R. Each element $N_i \in N$ corresponds to a pair $\{a_i, b_i\}$ denoting the begin and end instances of resource R_i that are computed in CREATERESOURCESLICES. Beginning and ending points of all partitions $\mathcal{L}_{ij} \in \mathcal{L}_i$ ($\mathcal{L}_i$ is the set of all performance partitions of resource R_i) of all $R_i \in R$ are

Algorithm 2 CREATERESOURCESLICES(R,Q)

```
1: $V \leftarrow \phi$                                  ▷ Matrix $|R| \times |Q|$
2: for $i \in |R|$ do
3:     for $k \in |Q|$ do
4:         $N \leftarrow \phi$                          ▷ Vector $|R|$
5:         for $l \in |R|$ do
6:             $rgn \leftarrow$ OVERLAPSF($R_l(T), Q_k(t)$)
7:             $N_l \leftarrow \{rgn.a, rgn.b\}$
8:         end for
9:         for $m \in N_{i1} \cdots N_{i2}$ do
10:            $S \leftarrow$ SLICERESOURCE($R, N, i, m$)
11:            INSERT($V_{ik}, m, S$)
12:        end for
13:    end for
14: end for
15: return $V$
```

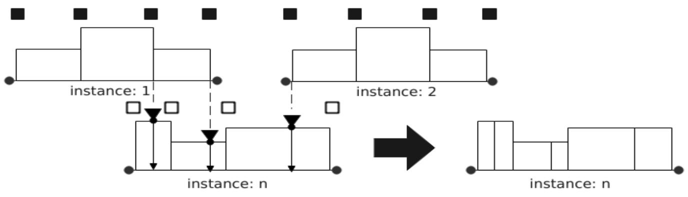


Fig. 8: Slicing a Resource Instance

extracted in P. These points are shown in Figure 8 (black and white squares). $\overleftarrow{\mathcal{L}_{ij}}$ and $\overrightarrow{\mathcal{L}_{ij}}$ are the begin and end points of j^{th} partition of resource R_i. Begin and end time of all partitions p of resource R_r are stored in set E w.r.t. instance n shown by the white squares in example Figure 8. As we are interested in slicing resource R_r w.r.t. to other resources in R, we keep only those points which lie within the time span of R_r. We complement vector E from P' to exclude the begin and ending points of partitions of R_r as shown by the black triangles in Figure 8. The remaining values stored in Ep are the begin and end points of other resources' partitions that lie within a partition of n^{th} instance of R_r. Then we split the partitions in Ep points and reshape the performance matrix in *Eph*.

We pre-process the partitions into slices using the above mentioned algorithms. All possible directed edges are constructed between the reachable slices. Now we proceed towards reformulating our ILP using these slices as vertices. Our reformulation will not be much different from the previous one shown using Equations 15...21. However we use $\widehat{X}$ and $\widehat{Y}$ instead of using using X and Y in our decision variable names. $\widehat{X}$ and $\widehat{Y}$ shown in Equations 25 and 26 now represent vertices and edges. Decision variable $\widehat{X}_{ijk}$ denotes whether j^{th} vertex(slice) of resource R_i is selected for request k. $\widehat{X}_{0,0,k}$ and $\widehat{X}_{-1,-1,k}$ are the start and terminal request vertices which are to be selected in order to satisfy request k. OUTDEGREE of the start request vertex and INDEGREE of the terminal request vertex are exactly 1 as shown in Equation 26. For any intermediate resource slice, its OUT-DEGREE and IN-DEGREE are to be same to retain continuity as shown in Equation 27. Similar to the previous ILP formulation, if there exists an edge out going from a vertex it is marked as used as shown in Equation 28.

$$\widehat{X}_{ijk} = \begin{cases} 1 & S_{ijk} \text{ is used} \\ 0 & otherwise \end{cases} \qquad \widehat{X}_{0,0,k} = 1 \ \forall k \qquad \widehat{X}_{-1,-1,k} = 1 \ \forall k \tag{25}$$

Algorithm 3 SLICERESOURCE(R,N,r,n)

```
1: $P \leftarrow \phi$                          ▷ begin, end stamps of all partitions
2: for $i \in 1 \dots |R|$ do
3:     for $j \in 1 \dots |\mathcal{L}_i|$ do      ▷ $\mathcal{L}_i$ is the partitions of $R_i$
4:         for $t \in a_i \dots b_i$ do          ▷ instances $N_i = \{a_i, b_i\}$
5:             $P = P \cup \{s_i + tk_i + \overleftarrow{\mathcal{L}_{ij}}, s_i + tk_i + \overrightarrow{\mathcal{L}_{ij}}\}$
6:         end for
7:     end for
8: end for
9: $E \leftarrow \phi$
10: for $\rho \in 1 \dots |\mathcal{L}_r|$ do
11:     $E = E \cup$ SPAN($R_r, n, \rho$)
12: end for
13: $P' \leftarrow \{p \in P : p \in [\text{MIN}(E), \text{MAX}(E)]\}$
14: $Ep \leftarrow P' \setminus E$
15: $C \leftarrow \phi$
16: for $h \in 1 \dots |\mathcal{L}_r|$ do
17:     $Eph \leftarrow \{p \in Ep : \overleftarrow{L_{rh}} < p < \overrightarrow{L_{rh}}\}$
18:     for $p \in Eph$ do
19:         $C \leftarrow C \cup$ SPLITF($\mathcal{L}_{rh}, p$)
20:     end for
21: end for
22: return $C$
```

$$\widehat{Y}^k_{ij\to pq} = \begin{cases} 1 & \text{hop } S_{ijk} \longrightarrow S_{pqk} \\ 0 & otherwise \end{cases} \qquad \sum_{i,j} \widehat{Y}^k_{0,0\to ij} = 1 \ \forall k \qquad \sum_{p,q} \widehat{Y}^k_{pq\to -1,-1} = 1 \ \forall k \tag{26}$$

$$\sum_{p,q} \widehat{Y}^k_{ij\to pq} = \sum_{p,q} \widehat{Y}^k_{pq\to ij} \quad \forall i,j \qquad \text{where} \quad S_{ijk} \in \text{IN}(S_{pqk}) \quad \forall i,j \qquad S_{pqk} \in \text{OUT}(S_{ijk}) \quad \forall p,q \tag{27}$$

$$\widehat{X}_{ijk} = \sum_{p,q} \widehat{Y}^k_{ij\to pq} \quad \forall p,q \qquad \text{where} \quad S_{pkq} \in \text{OUT}(S_{ijk}) \quad \forall p,q \tag{28}$$

We group the slices belonging to the same partition, and label each partition with a tuple (i, m, p) where the label consists of the resource id (i), instance (m) and partition (p). Slices in each group $U_{imp} \in U$ have the same sharing coefficient determined by SHARE(i, m, p) shown in Equation 14. As the minimum value of $C^k_i(m, p)$ is 1, maximum value of SHARE(i, m, p) is $|Q|$. Previously in Equation 20 we were constraining usage of the same partition for multiple requests to be

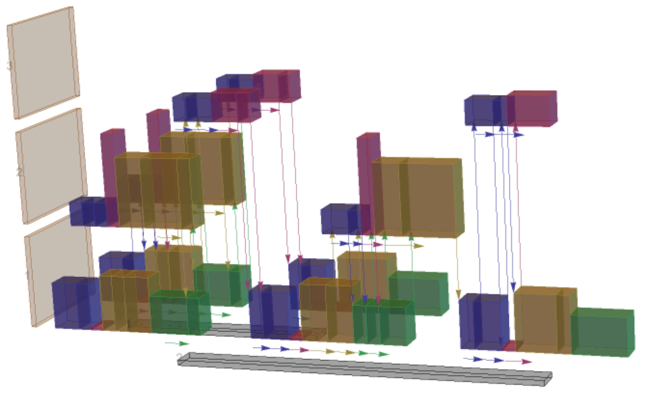


Fig. 9: Problem Visualization with Slicing (Legends in Table 3)

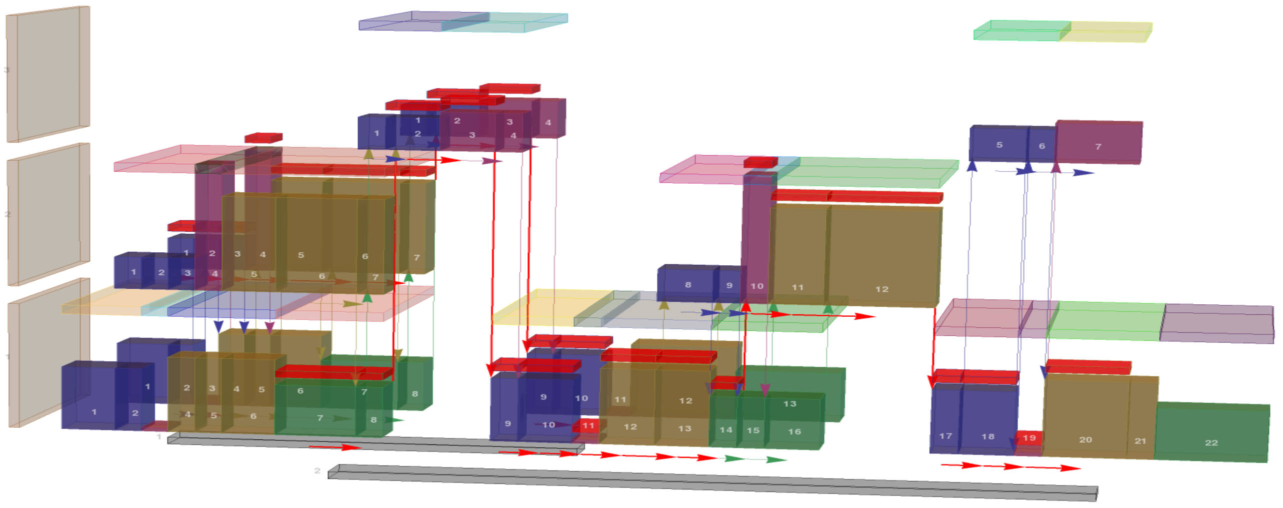


Fig. 10: Solution Visualization with Slicing (ϵ = 200), Legends in Table 3

$\leq$ SHARE(i, m, p). Now we are applying the constraint on usage of all slices belonging to the same partition. As maximum value of SHARE(i, m, p) is $|Q|$, maximum value of $|\mathcal{L}_{ip}|$SHARE(i, m, p) is $|\mathcal{L}_{ip}||Q|$, where $|\mathcal{L}_{ip}|$ is the length (time duration) of partition p of resource R_i. The partition $\mathcal{L}_{ip}$ is spit into multiple slices S_{ijk}, sum of size $|S_{ijk}|$ of all slices will be same as $|\mathcal{L}_{ip}|$ w.r.t. request $Q_k \in Q$. So now we constrain the total visited length of all slices in Equation 29.

$$\sum_k |\mathcal{S}_{ijk}|\widehat{X}_{ijk} \leq |\mathcal{L}_{ip}|\text{SHARE}(i, m, p) \quad \forall S_{ijk} \in U_{imp}$$
$$\text{where SHARE}(i, m, p) = \left\lceil \sum_k \frac{1}{C_i^k(m, p)} \right\rceil \tag{29}$$

In all other constraints from Equation 25 to 28 we are actually enforcing a shortest path for each request, which can be solved in polynomial time. However absence of this competition constraint of Equation 29 will lead to compaction in scheduling as mentioned before in Section 2. So in Section 5.1 we simulate the scheduling with or without this constraint to compare the two approaches compation and balancing.

Our objective shown in Equation 30 is to maximize total performance gained while minimizing cost of switch over from one resource to the other. S^g_{pqk} refers to the performance gain of slice (i, j, k). $H_{ij\to pq}$ is the hop cost for switching from slice S_{ij} to S_{pq}.

$$Max \sum (S^g_{pqk} - H_{ij\to pq})\widehat{Y}^k_{ij\to pq} \tag{30}$$

A set of example resources are shown in Table 2 previously. We now take two requests for the same set of capabilities, however having different time slots $\{t_s : 20, t_e : 100\}$ and $\{t_s : 60, t_e : 200\}$. The problem can be visualized in Figure 9 where x axis corresponds to resource slices, y (height) axis corresponds to resources and z (depth) axis corresponds to the requests. Slices (shown as cuboids) belonging to the same performance partition have the same color. Heights of these slices are proportional to the weight of performance partition. Each of the two gray tracks at the bottom braces the resource slices feasible for that request. Possible hops are shown in arrows originating from the source slice, heading towards the target slice. Arrows are colored with the same color as its source slice. Edges originate from the end position of the source slice and reach its target slice on its starting position. However if the start and end slice belong to the same resource end position of source resource and start position of the target resource are same. So for visualization we have drawn the edges originating from the middle of the source and reaching the target on its middle.

Now we solve the problem with the above mentioned Equations 25, 26, 27, 28, 29 and 30 with ϵ = 200. The solution has been visualized in Figure 10. Slices are labeled with their vertex id which is shown on the cuboids. Selected slices are highlighted with a thin red block on their head. Selected edges are marked with red arrows. Competition groups spanning multiple request layers are shown by thin layers floating on the top of the slices belonging to the same partitions. We can see in the visualization that slices $S_{1,7,1}, S_{1,7,2}, S_{1,8,1}, S_{1,8,2}$ (shown in green) belong to the same partition which is competing with slices $S_{2,6,1}, S_{2,6,2}, S_{2,7,1}, S_{2,7,2}$ (shown in yellow). Due to the constraint shown in Equation 29 slices $S_{1,7,1}, S_{1,8,1}$ are selected for request 1. But the same slices are not selected for request 2. Instead slices $S_{2,6,2}, S_{2,7,2}$ are selected to serve for request 2. On the other hand as the slices $S_{3,2,1}, S_{3,2,2}, S_{3,3,1}, S_{3,3,2}$ have no competitors they have been used for all requests.

Now as the partitions are quantized into non-uniform slices we may use some slices of a partition and then switch to a different resource and again come back to some slice of the previous resource's partition. Though such events are rare (depending on the hop cost), there will be more than one entry and exit points for a partitions. However using these informations we may also create a time table of allocation/de-allocation/migration tasks that we need to perform to fulfill the advanced reservation. A task is defined by a tuple of $\{time(t), source \to target\}$, where source and target are defined by a resource id. Each row in the time table is indexed by a time stamp (t). A sequence of such tasks are to be performed for each request $Q_k \in Q$. Source and target pair denotes the transition from source R_u to target R_v. Transitions like $\phi \to R_v$, $R_u \to \phi$ denotes the initial allocation and final deallocation while serving the request Q_k. Task schedules deduced from the solution shown in Figure 10 are shown in Table 4. Tables 4a and 4b shows the resource migration for request 1 and 2 respectively. Number of migrations depends on the hop cost (ϵ). If we use a lower hop cost on the same set of resources and requests we will have more number of migrations. Table 5 shows the tasks that we need to perform is ϵ = 150. Tasks that we need to perform for requests 1 and 2 are shown in Tables 5a and 5b.

4.3 Fault Tolerant Approach

In previous section we have discussed about two approaches to solve our problem one through shortest path which compacts

TABLE 4: Tasks (ϵ = 200)

(a) Request1

t	Task
20	$\phi \to 2$
70	$2 \to 3$
90	$3 \to 1$
105	$1 \to \phi$

(b) Request2

t	Task
50	$\phi \to 1$
70	$1 \to 3$
90	$3 \to 1$
135	$1 \to 2$
170	$2 \to 1$
205	$1 \to \phi$

TABLE 5: Tasks (ϵ = 150)

(a) Request1

t	Task
20	$\phi \to 2$
30	$2 \to 1$
35	$1 \to 2$
70	$2 \to 3$
90	$3 \to 1$
105	$1 \to \phi$

(b) Request2

t	Task
50	$\phi \to 1$
70	$1 \to 3$
90	$3 \to 1$
135	$1 \to 2$
170	$2 \to 1$
205	$1 \to \phi$

requests into fewer number of resources and another one balances by imposing capacity constraints. The second approach minimizes the overall consumer impact if one resource interrupts in service. However as sensors are resource constrained devices there is still a chance of a subset of request being disturbed if some resource performs poorly. In this section we move towards a hybrid approach by using both of these approaches and with some minor modification we provide a solution which balances while making the overall service fault tolerant.

To achieve that we select a set of secondary resources along with the primary resource selected with the above mentioned balanced method. To provide p fault tolerence (i.e. System remains unaffected if $\leq p$ resources are down at a given time) we transform the requests Q in input space to Q' by duplicating itself p times as shown below ($||$ is the concatenation operator).

$$Q' = \Big\|^{p}(Q) = \underbrace{Q \,\|\, Q \,\|\, \cdots \,\|\, Q}_{p \text{ times}} \quad \text{where } \{a\} \,\|\, \{b\} = \{a, b\}$$

So every Q_k and $Q_{k+|Q|}$ is actually the same request. The continuity constraints on Equation 27 are applied on all resource vertices. However the capacity constraints shown on Equation 29 is applied on vertices $X_{i,j,k}$ only where $1 \leq k \leq |Q|$. However for any resource slice if its competition $C_{ijk} \leq p$ then requested p tolerance cannot be provided for that slice. Also we don't want the primary and secondary resource to be the same we impose Equation 31.

$$\sum_{i=0}^{p-1} X_{i,j,k+i|Q|} \leq 1 \ \forall (i,j,k) \tag{31}$$

After we solve this transformed problem we get a pair of schedules for each request Q_k. One for Q_k which is solved while imposing the capacity constraints and the other for $Q_{k+|Q|}$ which is actually the second request. So whenever the primary resource serving request Q_k doesn't serve as promised we may switch to the secondary resource which is allocated for $Q_{k+|Q|}$ thus feasible to serve Q_k for that time. Thus we achieve fault tolerance with degree k with a minor modification to the ILP. In Section 5 we contrast these three approaches.

5 Experimental Results

We have implemented our algorithms in Mathematica [12]. To provide the inputs to the solver and for representation we have used Qt framework. We have also created a C++ library, Mathematica++ [13] to bridge between C++ and Mathematica over WSTP [12] connection.

For our simulation, resources having varying availability patterns and non uniform performance partitions are used. These resources are requested for different time spans. The experiments are conducted using different scheduling strategies to contrast the effectiveness of these strategies on the basis of (i) service performance (ii) resource migration (iii) resource sharing and (iv) resource utilization. Number of requests and resource capabilities required by the requests are varied in different experiments to test consistency of the results as the system is scaled up. We have also conducted experiments to relate hop cost (ϵ) introduced in this paper with resource migration. For all requests we consider a time span of 360 minutes to serve all requests. We call it operational time. To quantify resource sharing and resource utilization we use the three metrics $u(R_i)$, $c(R_i)$ and $s(R_i)$ as described below.

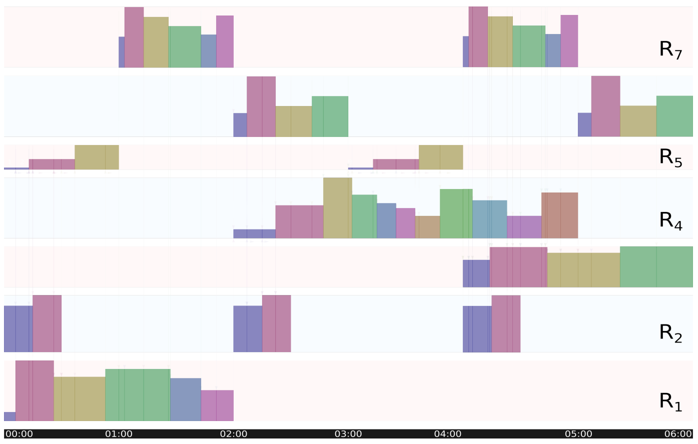


Fig. 11: Availability of resource in Table 6

$\mathbf{u(R_i)}$ is measured as ratio of total working time a resource R_i has served and operational time. e.g. a resource is scheduled for total 1 hour out of 360 minutes and thereafter by one request for 45 minutes has $u(R_i) = 60/360 = 1.167$

$\mathbf{c(R_i)}$ is total service time of resource R_i consumed by multiple requests. e.g. a resource is scheduled for total 1 hour and consumed (shared) by 2 requests for 15 minutes $s(R_i) = 2 * (15/360) + 1 * (45/360) + 0 * (300/360) = 0.2083$

$\mathbf{s(R_i)}$ is measured as $c(R_i)/u(R_i)$ which indicates sharing of that resource.

Increase in $s(R_i)$ implies fewer resources serving large number of requests, thus making multiple requests dependent on fewer number of resources. Increase and decrease in $u(R_i)$ implies resource utilization and starvation respectively.

5.1 Comparison of The Proposed Approaches

Table 6 shows the availability pattern (s, δ, κ) of each resource. Each of these resources start service on 1st January 2018 (UTC+05:30) on the time given in the start column in hh:mm format. We calculate time stamps in minutes (minutes passed since unix epoch). The non-uniform performance partitions of all resources are shown in Table 9. LCM of recurrences of the availability of resources in Table 6 is 6 hours (360 minutes). Figure 11 shows availability of the 7 resources for 6 hours $(00:00 \to 06:00)$. Height of each slice corresponds to the expected throughput of each performance partition as shown in the fourth column of Table 9. Slices belonging to the same partition are filled with the same color. We have used three strategies, (i) compaction, where competition constraints in Equation 29 are omitted, (ii) balanced, where competition constraints are used, and (iii) random, where a randomly chosen feasible schedule is used. We have conducted the following experiments in this paper. In the first two experiments we contrast the schedules obtained by different scheduling strategies and compare the quality of results obtained, on the basis of the parameters mentioned previously in Section 5. In the first experiment these 7 periodically available resources are homogeneous (only in terms of type and capabilities) and are scheduled for 2 requests. In the next experiment resources are heterogeneous (having varying capabilities) and are scheduled to 10 requests. Objective of the second experiment is to validate whether the comparative analysis obtained by the first

TABLE 6: Resource Availability Pattern

	$start(s)$	$duration(\delta)$	$recurrence(\kappa)$
1	00:00 (25245750)	02:00 (120)	06:00 (360)
2	00:00 (25245750)	00:30 (30)	02:00 (120)
3	04:00 (15777990)	02:00 (120)	12:00 (720)
4	02:00 (25245870)	03:00 (180)	12:00 (720)
5	00:00 (25245750)	01:00 (60)	03:00 (180)
6	02:00 (25245870)	01:00 (60)	03:00 (180)
7	01:00 (25245810)	01:00 (60)	03:00 (180)

TABLE 7: Resource Requests

	start (t_s)	end (t_e)
1	00:00 (25245750)	04:00 (25245990)
2	01:00 (25245810)	05:00 (25246050)
3	00:00 (25245750)	01:00 (25245810)
4	00:00 (25245750)	05:00 (25246050)
5	02:00 (25245870)	06:00 (25246110)
6	01:00 (25245810)	03:00 (25245930)
7	00:00 (25245750)	03:00 (25245930)
8	00:00 (25245750)	06:00 (25246110)
9	01:00 (25245810)	06:00 (25246110)
10	05:00 (25246050)	06:00 (25246110)

experiment is consistent with heterogeneity while increasing the number of requests. Maximum hop cost (ϵ in Equation 8) is set to 200 for both experiments. Then we relate hop cost and number of hops by gradually increasing hop cost in the same experiment.

Resources with Uniform Capabilities

The requests described in Table 8 are used in the first experiment. In this experiment we assume that the capabilities of all resources are same, all resources are feasible for all requests. But the resources show varying performances thereby contributing to varying appropriateness. Fault tolerance has been set to 1, i.e. maximum 1 backup resource slice will be selected for each selected primary resource slice. Figure 12a and 12b show the schedule obtained through compact, balanced and random approaches. The resource slices selected in compact, balanced and random approaches are marked with thick blue, red and green overlines respectively. The blue, red and green paths show the allocation and migration with compact, balanced, and random approaches. The secondary (first backup) resources which are supposed to be used if primary resources fail are marked with a circle inside the resource slice. The expected performance varying with time for both of these requests are shown in the Figures 13a and 13b. Number of migrations for different approaches are marked with arrows. The horizontal lines depicts the average performance observed in each approaches. It is observed that number of migrations in random approach ($Q_1 : 26, Q_2 : 32$) is too high as compared to both compact ($Q_1 : 5, Q_2 : 6$) and balanced approaches ($Q_1 : 6, Q_2 : 7$). Average performance of random approach is also the lowest. So the random approach can only be used if all the resources perform uniformly over time and hop cost is 0, which is rare in real life scenario. Two requests overlap for 120 minutes (02:00 → 04:00). During this period compact approach produces nearly the same schedule for both requests (blue line), however with balanced approach (red dashed line) it differs, as observed in Figure 12a and 12b. Hence, if R_4 fails between 02:00 and 04:00 then both requests will be affected when scheduled with the compaction approach. But with balanced approach this impact will be less as different resources are allocated for different requests. Number of requests served by each resources are shown in Figures 14a to 14g. We can see that R_2 serves 2 requests for a short period with compact approach (blue), whereas with balanced approach it serves 1 request, thus decreasing the impact of failure as shown in Figure 14b. Total length of time R_2 is used has also increased in balanced approach, thereby decreasing resource starvation. Similar effects can be observed for resource 3, 4, and 6 shown in Figures 14c, 14d and 14f respectively.

$s(R_i)$ and $u(R_i)$ of the 7 resources are shown in Figures 15a and 15b with colored marks. The average $s(R_i)$ and $u(R_i)$ of all resources are shown as colored horizontal lines. It is observed that although compact approach achieves marginally increased amount of sharing ($s(R_i)$) balanced approach results significant amount of increase in resource utilization ($u(R_i)$).

TABLE 8: Resource Requests

	$start(t_s)$	$end(t_e)$
1	00:00 (25245750)	04:00 (25245990)
2	02:00 (25245870)	06:00 (25246110)

TABLE 9: Performance Partitions

	$\overleftarrow{\mathcal{L}_{ij}}$	$\overrightarrow{\mathcal{L}_{ij}}$	$\lvert\mathcal{L}_{ij}\rvert$	$\widehat{\mathcal{L}_{i,j}}$
$\mathcal{L}_{11}$	0	6	6	47
$\mathcal{L}_{12}$	6	26	20	120
$\mathcal{L}_{13}$	26	53	27	97
$\mathcal{L}_{14}$	53	87	34	108
$\mathcal{L}_{15}$	87	103	16	95
$\mathcal{L}_{16}$	103	120	17	78
$\mathcal{L}_{22}$	0	15	15	100
$\mathcal{L}_{23}$	15	30	15	115
$\mathcal{L}_{31}$	0	14	14	73
$\mathcal{L}_{32}$	14	44	30	91
$\mathcal{L}_{33}$	44	82	38	83
$\mathcal{L}_{34}$	82	120	38	92
$\mathcal{L}_{41}$	0	22	22	47
$\mathcal{L}_{42}$	22	48	26	81
$\mathcal{L}_{43}$	48	62	14	120
$\mathcal{L}_{44}$	62	75	13	96
$\mathcal{L}_{45}$	75	85	10	84
$\mathcal{L}_{46}$	85	96	11	77
$\mathcal{L}_{47}$	96	109	13	66
$\mathcal{L}_{48}$	109	125	16	104
$\mathcal{L}_{49}$	125	143	18	88
$\mathcal{L}_{410}$	143	161	18	66
$\mathcal{L}_{411}$	161	180	19	99
$\mathcal{L}_{51}$	0	14	14	37
$\mathcal{L}_{52}$	14	37	23	49
$\mathcal{L}_{53}$	37	60	23	69
$\mathcal{L}_{61}$	0	7	7	68
$\mathcal{L}_{62}$	7	22	15	120
$\mathcal{L}_{63}$	22	41	19	78
$\mathcal{L}_{64}$	41	60	19	92
$\mathcal{L}_{71}$	0	3	3	78
$\mathcal{L}_{72}$	3	13	10	120
$\mathcal{L}_{73}$	13	27	14	106
$\mathcal{L}_{74}$	27	43	16	93
$\mathcal{L}_{75}$	43	51	8	81
$\mathcal{L}_{76}$	51	60	9	108

TABLE 10: capabilities

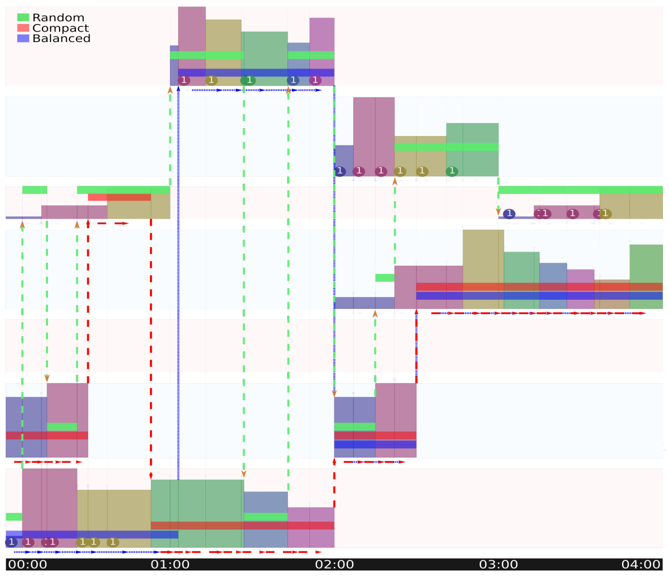
R	sensitivity	accuracy
1	0 → 100	±2
2	0 → 80	±5
3	0 → 50	±2
4	0 → 50	±2
5	0 → 100	±2
6	0 → 50	±5
7	0 → 100	±5
Q	sensitivity	accuracy
1	0 → 50	±5
2	0 → 50	±2
3	0 → 80	±5
4	0 → 50	±2
5	0 → 80	±5
6	0 → 50	±5
7	0 → 50	±5
8	0 → 50	±2
9	0 → 50	±5
10	0 → 50	±5

TABLE 11: Feasibility (■ feasible, □ infeasible)

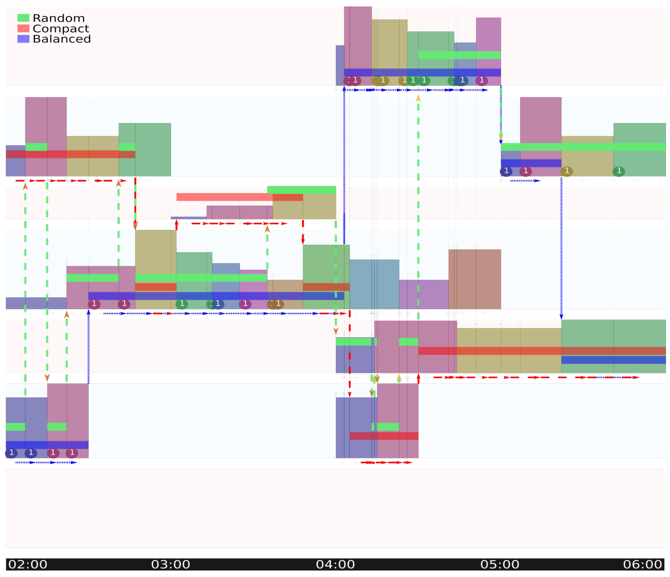
q \ r	1	2	3	4	5	6	7
1	■	■	□	□	■	□	■
2	■	□	■	■	■	□	□
3	■	■	□	□	■	□	■
4	■	□	■	■	■	□	□
5	□	■	■	■	■	■	■
6	■	■	□	■	□	■	■
7	■	■	□	■	■	■	■
8	■	■	■	■	■	■	■
9	■	■	■	■	■	■	■
10	□	□	■	□	□	■	□

Resources with Non-Uniform Capabilities

Next we experiment with larger set of requests as shown in Table 7 and simulate a scenario when all resources are not feasible for all requests. We assume all the resources are of same type and requests are also for same type of resource. But unlike the previous experiment the capabilities the resources vary contributing to a more heterogeneous inventory. Availability pattern of the resources are same as the previous experiment. We have used two capability parameters, *sensitivity* and *accuracy* out of which accuracy is a restrictive parameter. The availability values of resources, and requests are shown in Table 10. In Table 10 the first 7 rows are the capabilities of resources and next 10 rows are capability requirements of requests. The feasibility matrix is shown in Table 11. Each cell (k, i) in this table denotes whether i^{th} resource is feasible for k^{th} request. A black square denotes that the resource is feasible for the request and a white square denotes the resource is infeasible.

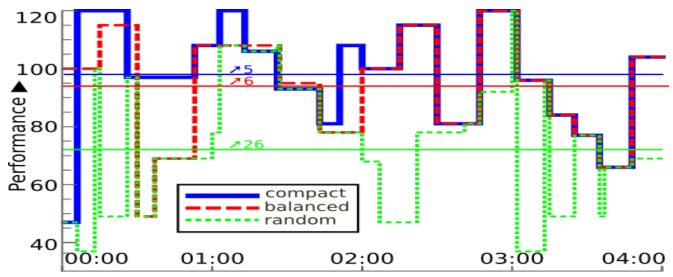


(a) Schedule for Request 1

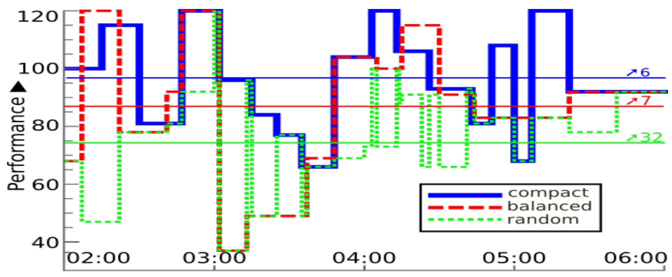


(b) Schedule for Request 2

Fig. 12: Schedule for requests in Table 8 Using Compact and Balanced Strategies, Legends in Table 3



(a) Schedule for Request 1



(b) Schedule for Request 2

Fig. 13: Performance for requests in Table 8 Using Compact, Balanced and random Strategies

In this experiment we observe that the average performance achieved using compact approach is marginally better than the balanced approach for all 7 resources as observed in the previous experiment.† Average hops for these 10 requests for compact and balanced approaches are 4.1 and 4.3 respectively. $s(R_i)$ and $u(R_i)$ of the 7 resources in this experiment are shown in Figures 15c and 15d with colored marks. We observe a similar trend in sharing as well as utilization in this scaled up experiment as well. Similar to the previous experiment the compact approach achieves increased sharing ($s(R_i)$) whereas the balanced approach results to significant amount of increase in resource utilization ($u(R_i)$). So our results suggest that the balanced approach provides better resource utilization while marginally affecting the performance, resource migration and sharing, when compared with the compact approach.

Relation between hop cost and number of migration

We have previously modeled resource migration with a metric hopcost as a function of ϵ in Equation 8. ϵ is used as maximum cost for resource migration. So far we have simulated with $\epsilon = 200$. Now we increase ϵ and check how that affects the number of migrations using the same set of resources and requests as used previously. Figure 16 shows number of migrations each request will encounter individually for a given ϵ value. Total‡ number of migrations (including migrations of all resources) is shown in Figure 17. We can observe in Figure 16 that although increasing ϵ may causes overall decrease in resource migration, it may result in increase of number of migration for some requests. For example, when we increase ϵ from 300 to 400, number of migrations for Q_6 increases although total number of migrations performed by all requests are actually decreased from 53 to 45. We can also observe that as we keep increasing the value of ϵ, decrease in total number of migrations slows down and moves towards a saturation point.

†. Please see Figures 18, 19 in Appendix A for schedule and performance for each request respectively and Figure 20 for dependency and utilization of resources in this experiment.

‡. initial allocation and request completion are included as migration from nothing and to nothing respectively.

6 Related Works

Sensors in varying application domains, like health care [14], smart cities [15], environment monitoring [16] are integrated with applications hosted on cloud. However, in most cases, the sensors are dedicated for particular applications. In this context, Lim et al mentioned that "The tight coupling between a network and application limits the usability of sensor data" [17]. They proposed an SCI to overcome such limitations. Other researchers have also come up with proposals for SCI. Surveys on sensor-cloud architecture and applications are presented in [18] [19]. Bose et al propose an architecture of SCI to support sensor-aware applications [2] for heterogeneous devices using virtualisation of sensors that lays the foundation of this paper.

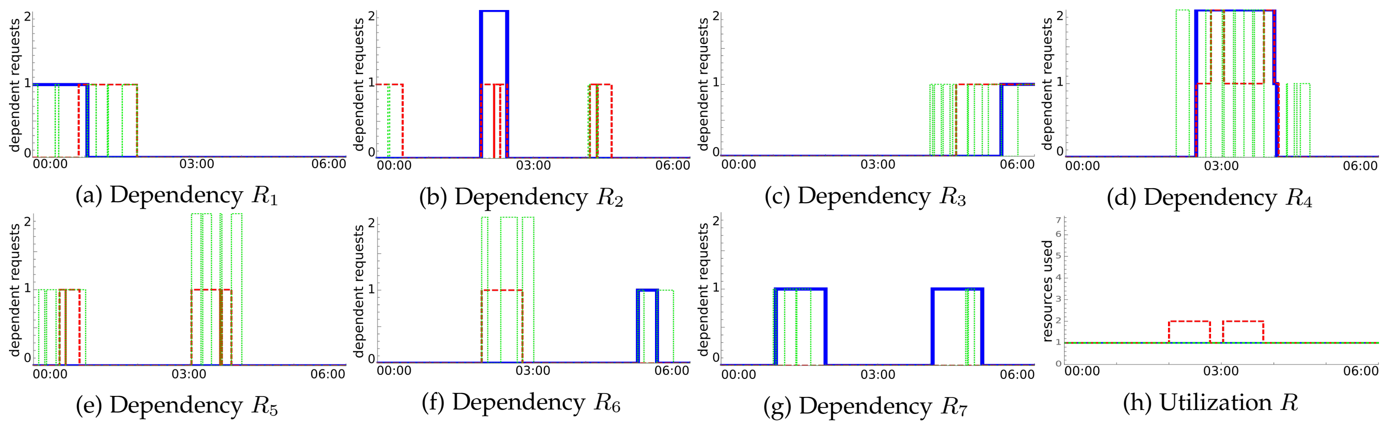

(a) Dependency R_1 (b) Dependency R_2 (c) Dependency R_3 (d) Dependency R_4 (e) Dependency R_5 (f) Dependency R_6 (g) Dependency R_7 (h) Utilization R

Fig. 14: Resource Dependency and utilization w.r.t. time for requests in Table 8 (— compact — balanced — random)

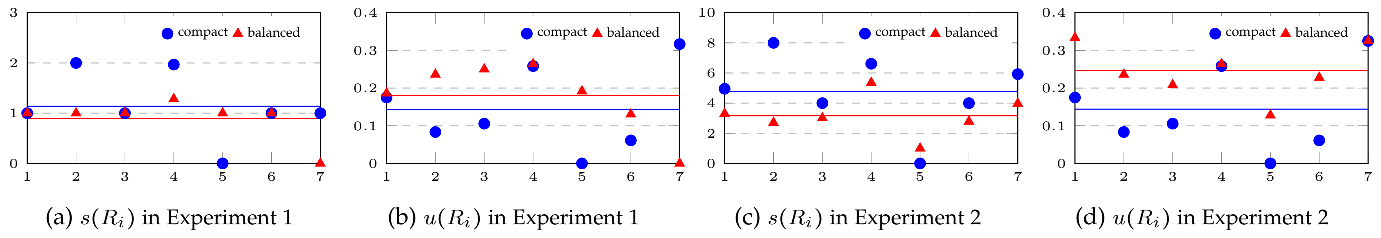

(a) $s(R_i)$ in Experiment 1 (b) $u(R_i)$ in Experiment 1 (c) $s(R_i)$ in Experiment 2 (d) $u(R_i)$ in Experiment 2

Fig. 15: Resource sharing and utilization in Experiment 1 and 2

Rachkidi points out [20] that there can be two different approaches to support a 'pay as you go' business model in IoT (Internet of Things) frameworks — one is a data centric approach [21], and the other is a device centric [2] approach. Data centric approaches are used in [22] where sensing resources are exposed as data sources through APIs. On the other hand, in [2] remotely located sensors are exposed as virtual devices to the Virtual Machines (VM) in cloud's computing infrastructure. It has been shown that use of virtual sensors allow loose coupling between applications and sensing resources, thereby enabling sharing the sensors on demand basis.

As pointed out earlier, when sensors are available in a market-place for consumption by different applications, scheduling of sensors become a complex problem. Scheduling of transmission of data from sensing devices has been investigated in contemporary literature. Wireless sensor networks have been used in industrial process monitoring applications using WirelessHART [23] network. In their system, real time transmission scheduling for sensors to actuators for a set of periodic data flows has been addressed in [24]. Here, a heuristic based approach is proposed towards the deadline sensitive transmission scheduling problem. TDMA based scheduling schemes are proposed in [25] to schedule sensing tasks in WSN. As the sensing devices are energy constrained resources, in [26] availability of resources are temporally scheduled to maximize the spatial coverage and lifetime of a set of homogeneous sensors. In [8] [27] spatio-temporal phenomenon modeling is used to predict the phenomenon on unobserved locations by observing a small subset of locations. However such algorithms do not deal with the complexities encountered while scheduling a set of heterogeneous sensors which are available periodically. The problem of finding optimum offsets to minimize periodic spikes when multiple sensors transmit simultaneously is addressed in [28]. In [29], periodic-task based high-performance scheduling models are proposed with a cloud-supported caching mechanism for multisensor gateway. Load balancing schemes for LEO satellite networks for IoT communications have been proposed in [30]. Though

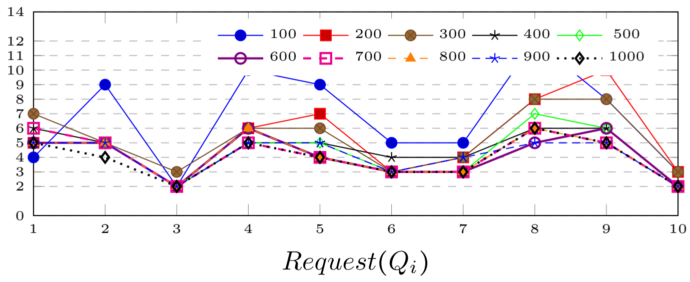


Fig. 16: Number of migration for different ϵ in (Equation 8)

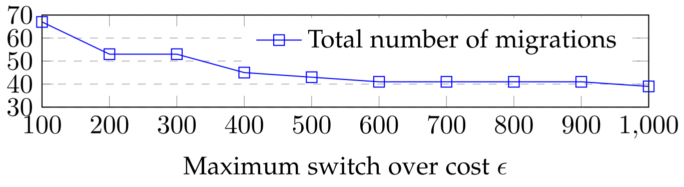


Fig. 17: Total migrations as we change ϵ of Equation 8

these papers deal with periodic properties of resources, they do not handle scheduling between several applications in an open system scenario. With periodic availability, it is required to schedule the service time of multiple resources appropriately. Ignoring periodic nature while scheduling these resources will lead to sensing holes and even infeasible schedules. So we model periodic availability as recurrence pattern and incorporate that into the optimization problem.

In [31] [32] algorithms have been proposed for selecting a subset from a pool of available homogeneous sensors. Scheduling computational tasks on edge devices are proposed in [33] [34]. Geographic location is treated as an important property of resources and requests in [35] [36]. Sensing requests are modeled as circular and rectangular geo-fences respectively in [35] and [36]. In [35] a Gossip based algorithm is proposed that discovers feasible resources through communicating in the sensor network. In contrast to the above research works, this paper assumes that the heterogeneous sensing resources are periodically available and sensing performance is time varying.

7 Conclusion

This paper is focused on scheduling sensing resources in sensor cloud infrastructure. Primarily different approaches have been proposed that consider the availability, appropriateness and performance of the sensors while mapping them to the application requests. An evaluation has also been made among these approaches. This work can be extended to many other variants of scheduling problems, including generic job scheduling with heterogeneous resources having repetitive appearance pattern and predictive performance w.r.t. time. Additional constraints may be imposed on the problem. So far we did not consider resource buckets. However consumer requests may constrain requested resources to be allocated from the same bucket or from different buckets. Our proposed algorithm can be extended to solve this problem.

Acknowledgments

This work is supported by Information Technology Research Academy (ITRA), Government of India under, ITRA-Mobile grant ITRA/15(59)/Mobile/RemoteHealth/01.

References

[1] S. Bose *et al.*, “A Framework for Heterogeneous Resource Allocation in Sensor-Cloud Environment,” *Wirel. Pers. Commun.*, apr 2019.

[2] S. Bose and N. Mukherjee, “SensIaas: A Sensor-Cloud Infrastructure with Sensor Virtualization,” in *3rd Int. Conf. Cyber Secur. Cloud Comput.* IEEE, 2016.

[3] A. Mandal *et al.*, “Impact of Mobility on Community Sensing with Environment Sensors,” in *13th Int. Conf. Signal-Image Technol. Internet-Based Syst.* IEEE, 2017.

[4] S. C. Shah and S. Korea, “NETWORK AWARE RESOURCE SCHEDULING IN SENSOR CLOUD,” *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 2, 2017.

[5] B. Faltings *et al.*, “Incentive Mechanisms for Community Sensing,” *IEEE Trans. Comput.*, vol. 63, no. 1, 2014.

[6] S. Bose and N. Mukherjee, “PrIOR: A prime number based I/O redirection algorithm for sensor-cloud infrastructure,” in *2016 Int. Conf. High Perform. Comput. Simul.* Innsbruck: IEEE, 2016.

[7] A. Samba *et al.*, “Instantaneous throughput prediction in cellular networks: Which information is needed?” in *IFIP/IEEE Symp. Integr. Netw. Serv. Manag.* IEEE, 2017.

[8] A. Krause *et al.*, “Toward Community Sensing,” in *Int. Conf. Inf. Process. Sens. Networks.* IEEE, 2008.

[9] Q. He *et al.*, “On the predictability of large transfer TCP throughput,” *Comput. Networks*, vol. 51, no. 14, 2007.

[10] J. Yao *et al.*, “An empirical study of bandwidth predictability in mobile computing,” *Proc. third ACM Int. Work. Wirel. Netw. testbeds, Exp. Eval. Charact. - WiNTECH ’08*, 2008.

[11] R. K. Ahuja *et al.*, *Network flows : theory, algorithms, and applications*. Prentice Hall, 1993.

[12] I. Wolfram Research, “Mathematica, Version 11.3.”

[13] S. Bose, “https://gitlab.com/neel.basu/mathematicapp/.”

[14] N. Mukherjee *et al.*, “Virtual sensors in remote healthcare delivery: Some case studies,” in *BIOSTEC 2016*, 2016.

[15] N. Mitton *et al.*, “Combining Cloud and sensors in a smart city environment,” *EURASIP J. Wirel. Commun. Netw.*, vol. 2012, no. 1, 2012.

[16] S. Bose *et al.*, “Environment Monitoring in Smart Cities Using Virtual Sensors,” in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2016.

[17] Y. Lim and J. Park, “Sensor Resource Sharing Approaches in Sensor-Cloud Infrastructure,” *Int. J. Distrib. Sens. Networks*, vol. 10, no. 4, 2014.

[18] A. Alamri *et al.*, “A Survey on Sensor-Cloud: Architecture, Applications, and Approaches,” *Int. J. Distrib. Sens. Networks*, vol. 9, no. 2, 2013.

[19] A. Botta *et al.*, “Integration of Cloud computing and Internet of Things: A survey,” *Futur. Gener. Comput. Syst.*, vol. 56, 2016.

[20] E. Rachkidi, “Modelling and placement optimization of compound services in a converged infrastructure of cloud computing and internet of things,” *Netw. Internet Archit. [cs.NI]. Univ. Paris-Saclay; Univ. d’Evry-Val-d’Essonne*, 2017.

[21] J. Soldatos *et al.*, “OpenIoT: Open Source Internet-of-Things in the Cloud.” Springer, Cham, 2015.

[22] S. Alam *et al.*, “SenaaS: An event-driven sensor virtualization approach for internet of things cloud,” *IEEE Int. Conf. Networked Embed. Syst. Enterp. Appl. NESEA*, 2010.

[23] D. Chen *et al.*, “Why WirelessHART,” in *WirelessHART™*. Boston, MA: Springer US, 2010.

[24] A. Saifullah *et al.*, “Real-time scheduling for WirelessHART networks,” *Proc. - Real-Time Syst. Symp.*, 2010.

[25] R. Dalvi and S. K. Madria, “Energy Efficient Scheduling of Fine-Granularity Tasks in a Sensor Cloud.” Springer, Cham, 2015.

[26] C. Han *et al.*, “An Energy Efficiency Node Scheduling Model for Spatial-Temporal Coverage Optimization in 3D Directional Sensor Networks,” *IEEE Access*, vol. 4, 2016.

[27] A. Krause *et al.*, “Simultaneous optimization of sensor placements and balanced schedules,” *IEEE Trans. Automat. Contr.*, vol. 56, no. 10, 2011.

[28] S. Oh and J. W. Jang, “A scheme to smooth aggregated traffic from sensors with periodic reports,” *Sensors (Switzerland)*, vol. 2017, no. 3, 2017.

[29] Y. Lyu *et al.*, “High-performance scheduling model for multisensor gateway of cloud sensor system-based smart-living,” *Inf. Fusion*, vol. 21, 2015.

[30] Z. Liu *et al.*, “HGL : A hybrid global-local load balancing routing scheme for the Internet of Things through satellite networks,” *Int. J. Distrib. Sens. Networks*, vol. 13, no. 10, 2017.

[31] S. Joshi and S. Boyd, “Sensor Selection via Convex Optimization,” *IEEE Trans. Signal Process.*, vol. 57, no. 2, 2009.

[32] R. J. Caverly, “Optimal linear combination of sensors and actuators to enforce an interior conic open-loop system,” *Int. J. Robust Nonlinear Control*, vol. 29, no. 17, 2019.

[33] S. R. Sarangi *et al.*, “Energy efficient scheduling in IoT networks,” in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing - SAC ’18*. ACM Press, 2018.

[34] G. Li *et al.*, “Methods of Resource Scheduling Based on Optimized Fuzzy Clustering in Fog Computing,” *Sensors*, vol. 19, no. 9, 2019.

[35] S. Abdelwahab *et al.*, “Cloud of Things for Sensing-as-a-Service: Architecture, Algorithms, and Use Case,” *IEEE Internet Things J.*, vol. 4662, no. c, 2016.

[36] S. Misra *et al.*, “On Theoretical Modeling of Sensor Cloud: A Paradigm Shift From Wireless Sensor Network,” *IEEE Systems Journal*, vol. 11, no. 2, 2017.

Sunanda Bose is working as a Senior Research Fellow under RUSA 2.0 in Dept. of CSE, Jadavpur University, India. His current research interest is Resource Management in Sensor Cloud Infrastructure. He has worked in the project “Remote Health: A Framework for Healthcare Services using Mobile and Sensor-Cloud Technologies” funded by Ministry of Electronics and Information Technology, Govt. of India.

Nandini Mukherjee received Ph.D. degree in computer science from the University of Manchester, UK. Currently she is a professor in Dept. of Computer Science and Engineering in Jadavpur University, India. She served as the Director of School of Mobile Computing and Communication, Jadavpur University. Her current research interests are in the areas of High Performance Computing and Wireless Sensor Networks.