SCARF: A Scalable Data Management Framework for Context-Aware Applications in Smart Environments

Eun-Jeong Shin *

University of California, Irvine
ejshin1@uci.edu

Sharad Mehrotra

University of California, Irvine
sharad@ics.uci.edu

Dhrubajyoti Ghosh *

University of California, Irvine
dhrubajg@uci.edu

Nalini Venkatasubramanian

University of California, Irvine
nalini@ics.uci.edu

ABSTRACT

This paper proposes a context data management framework for efficient context data collection and evaluation to support context-aware applications. The proposed framework named *SCARF* leverages two architectures: a server-based centralized architecture and an edge-based architecture to address the problem of scalability of context collection and evaluation. *SCARF* collects the context data efficiently from resource constrained mobile/edge and multiple in-situ sensors (not resource constrained) deployed in smart environments. In particular, *SCARF* takes a set of contextual queries from applications and optimizes the rate at which the context data needs to be collected and/or transmitted from the sensor to the server. Furthermore, *SCARF* decides whether the code for evaluating context should be executed on the server or on the edge. Execution on the edge will result in reduced communication between the server and the edge but can result in low performance due to resource constraints of the edge. On the other hand, evaluation of context on the server, will require *SCARF* to transmit sensor data from the sensor to the server but can result in high performance due to high resource availability of the server. *SCARF* is an adaptive framework which generates an optimal context acquisition plan and a context evaluation plan such that the overall latency is minimized.

KEYWORDS

Context-aware applications, Mobile Computing, Smart Environments, Scalable Data Management

ACM Reference Format:

Eun-Jeong Shin *, Dhrubajyoti Ghosh *, Sharad Mehrotra, and Nalini Venkatasubramanian. 2019. SCARF: A Scalable Data Management Framework for Context-Aware Applications in Smart Environments. In *16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, November 12–14, 2019, Houston, TX, USA. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3360774.3360826>

* Both the authors contributed equally.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MobiQuitous '19, November 12–14, 2019, Houston, United States

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-7283-1/19/11...\$15.00

<https://doi.org/10.1145/3360774.3360826>

1 INTRODUCTION

We are living in the era of smart environments where we are surrounded by billions of devices collecting contextual information from us. The rapid growth of sensors has revolutionized smart environments such as smart homes, smart shopping malls, and intelligent office buildings. With ever increasing sensor deployments, understanding “context” and subsequently making sense of environment, situation, sensors, along with providing real time responses have become fundamental research challenges [1].

Context-aware applications play an important role in smart systems, starting from smart hospital [2–5], smart surveillance [6, 7], smart community [8, 9], smart notification [10–13], smart shopping [14, 15], to smart building [16, 17] applications. Let us consider the following examples of context-aware applications in smart environments. In a smart shopping mall, when a user steps in, the system recognizes the user based on the cameras installed in the mall. The system retrieves previously purchased items by the user and creates a recommendation list of items for the user to purchase. Suppose a user with a health problem, such as allergies, enters into a smart building. The system detects the current level of pollen in the air at the user’s location and sends a warning to the user that the pollen level inside that location is very high. Based on this warning, the user takes out or buys an asthma pump and/or avoid that particular location inside the building.

Context acquired from user devices is often very low-level for the end applications to interpret. Such information must be enriched/tagged appropriately before it can be effectively used by the applications. Advancement of data processing techniques, such as machine learning and crowd-powered techniques, have led to a variety of sophisticated analysis algorithms being developed in past few years specifically targeting Internet of Things (IoT) systems. However, these techniques can not be applied to the collected context data on the resource constrained mobile devices due to high resource requirements of these algorithms.

Systems that support context-aware applications face with several interesting challenges. Some of these challenges are as follows:

- **Scalability.** Scalability is an important issue while supporting large number of context-aware applications. Several architectures have been proposed in the past to address the scalability issues such as processing speed-up, reduced communication bandwidth, and increased reliability [18]. Among different programming paradigms of context-aware systems, service-oriented and agent-oriented [19] programming model have addressed the scalability issues up to some extent but they do not consider the scalability

of context collection and evaluation from a large number of heterogeneous sources.

- **Latency.** The context-aware applications need to provide real time/near real time responses to the end users. In this scenario, latency becomes a significant issue while detecting/extracting contextual features from collected context data [20–22].
- **Energy Consumption.** Most of the devices in smart environments are battery powered. Energy consumption becomes a critical issue as context-aware applications consume large amount of energy from user's devices. Several authors [23–26] have pointed out this issue for context-aware applications.
- **Security.** Some of the major security concerns of context-aware applications include context data access by malicious users, gaining access to user's mobile sensors by attacker and inferring sensitive user information from the output of these applications. Systems that support context-aware applications need to provide security guarantees against these types of attackers.
- **Privacy.** Context-aware applications can disclose behavior and social phenomena such as a user's location, their friends, associates, the timing of their interactions which can lead to serious privacy concerns [27].

In this paper, we focus on the scalability challenge: in particular, achieving minimum latency and ensuring a bounded amount of energy consumption. We propose a context collection/management framework entitled SCARF which resides between the sensors and the applications to support the needs of the context-aware applications. In particular, SCARF takes a set of contextual queries from applications and optimizes the rate at which the context data needs to be collected and/or transmitted to the server in order to minimize the latency. Furthermore, SCARF decides whether the code for extracting contextual features should be executed on the server or on the edge. If the code is executed on the edge, it results in reduced communication between the server and the edge. However, it can result in low performance due to low resource availability of the edge. On the other hand, if it is executed on the server, the context data needs to be transmitted from the sensor to the server. However, it can result in better performance due to high resource availability of the server.

The proposed framework SCARF is an adaptive framework which changes the acquisition plan and contextual feature extraction plan based on several factors. It considers the dynamic state of sensors (e.g., sensor acquisition cost can change based on the battery level, network connectivity, optimizations performed by the underlying OS etc.), cost of network transfer at different points of time, list of contextual features that are concurrently being evaluated by the applications, to determine an optimized plan. We use SCARF to develop a novel context-aware messaging application called *Hedwig* that allows users to send and receive contextualized messages.

In summary, our contribution in this paper is as follows:

- (1) We introduce a framework named SCARF, an efficient and scalable framework for collecting and managing context data from a variety of mobile and in-situ sensors for context-aware applications.
- (2) We have developed a novel context-aware messaging application named *Hedwig* where the delivery of the message is dictated by the conditions on complex contexts set by the senders and receivers.
- (3) We provide a survey of existing context-aware applications and identify the data management, energy management, and security issues of the context-aware applications.
- (4) We have experimentally evaluated SCARF using different types of real world contexts mimicking the needs of context-aware applications in smart environments.

The rest of the paper is organized as follows. Section 2 describes how we model context in our framework SCARF. Section 3 describes the architecture of SCARF. Section 4 provides experimental evaluations of context in scalable manner, while Section 5 reviews the related work. Section 6 concludes the paper with plans for future work.

2 MODELING CONTEXT IN SCARF

We consider two types of context-aware applications: *user-centric* and *building-centric* [1] context-aware applications. User-centric and building-centric context are defined as follows:

User-Centric Context. We define a user-centric context as a context which consists of the properties of users inside a building. We denote a user-centric context using the notation of U_i .

In the following, we provide examples of user-centric context that can be specified in SCARF:

- $\langle user, start_time, end_time \rangle$
- $\langle user, location, start_time, end_time \rangle$
- $\langle user, activity, start_time, end_time \rangle$
- $\langle user, location, occupancy, start_time, end_time \rangle$
- $\langle user, location, activity, start_time, end_time \rangle$

Building-Centric Context. A building-centric context is defined as any context which consists of a number of building properties such as temperature, occupancy, air volume, and water pressure. We denote a building-centric context using the notation of B_i . Example building-centric contexts can be as follows: $\langle \text{Temperature of room A is higher than } 85^\circ \text{ F} \rangle$, $\langle \text{Occupancy of rooms higher than } 100 \rangle$, and $\langle \text{List of rooms after 7 p.m. with occupancy higher than } 5 \rangle$. Such contexts can be specified by the building administrators to monitor the building resources such as rooms and sensors at different points of time of a day. Examples of building-centric contexts that can be specified in SCARF:

- $\langle sensor, threshold, start_time, end_time \rangle$
- $\langle sensor, sensor_health, rate, start_time, end_time \rangle$
- $\langle sensor, sensor_anomaly, value, start_time, end_time \rangle$

Context Evaluation. Both building-centric and user-centric context can be derived from the data captured by the sensors that are part of the infrastructure. The sensors, in general, can be in-situ (such as temperature sensors and occupancy sensors) or mobile sensors such as mobile devices, cell phones, and smart watches that are carried by the user. Data from the sensors is collected based on the need of applications and such data is used to create contextual information relevant to the context-aware application. For instance, a context-aware message in *Hedwig* may specify that a user should

Notation	Definition
C_i	Context
s_i	An in-situ sensor
m_i	A mobile sensor
$\mathcal{E}_i$	Energy consumption cost of collecting one observation from m_i
$\mathcal{S}_i$	Data storage cost of storing one observation of m_i
$\mathcal{Y}_i$	Data transmission cost of transmitting observations of m_i
v_i^a	Acquisition frequency of sensor m_i
v_i^t	Transmission frequency of sensor m_i
$\mathcal{A}(m_i, v_i^a, v_i^t)$	Total acquisition cost of sensor m_i with acquisition frequency of v_i^a and the transmission frequency of v_i^t
f_i	Evaluation function
$w_s(f_i)$	Running time of the function f_i in the server
$w_m(f_i)$	Running time of the function f_i in the mobile
b_i	A bit vector signifying whether the function should be evaluated in the server or on the mobile

Table 1: Frequently used notations.

be delivered a message when the user enters his/her workspace. Sensor data, e.g., the GPS device on the mobile phone carried by the user and/or data about connectivity of the user's device to the in-situ WiFi access points can be used to determine the above context (viz., a user enters his/her office). Note that often context required by the end application may not be directly captured by the sensor. Instead, such a context is evaluated using (a sequence of) possibly expensive functions on top of data from one or more sensors. To support context-aware applications, SCARF continuously computes the contextual features from raw sensor data at a given periodicity between the timestamps when the application needing the specific context is active.

Cost of Context Acquisition. We assume that the cost of context acquisition from a mobile sensor is significantly higher than an in-situ sensor. We therefore focus on the data acquisition cost from the mobile sensors in the following part of the paper. The cost of context acquisition from a mobile sensor m_i is dependent on the following factors: energy consumption in capturing data for a given sensor associated with the mobile device, storage cost of the collected data and the energy consumption cost of transmitting the data to the server.

We denote the energy consumption cost of one probe from a mobile sensor by the notation of $\mathcal{E}$. For example, the cost of a sensor m_i when it is probed once is denoted by $\mathcal{E}_i$. We associate an **acquisition frequency** (denoted by v_i^a) with each of the mobile sensors. This frequency signifies the number of times the sensor is probed per second. For example, if the acquisition frequency of a GPS sensor is set to five then the GPS sensor is probed five times per second. Similarly, we denote the storage cost by the notation of $\mathcal{S}$ and the data transmission cost by the notation of $\mathcal{Y}$. In the previous example, the storage cost of m_i is denoted by $\mathcal{S}_i$ and the data transmission cost is denoted by $\mathcal{Y}_i$. Both $\mathcal{S}_i$ and $\mathcal{Y}_i$ are measured in terms of energy consumption cost (battery level drop) of the mobile device.

We associate a **transmission frequency** with each sensor and it is denoted by the notation of v_i^t . This is defined as the number of times context data is transmitted from the mobile sensor to the server per second. For example, if $v_i^t = 2$, then the sensor data is transmitted from the mobile sensor to the server twice per second. This implies that the context data is collected for a duration of $1/2$ seconds = 0.5 seconds and then transmitted to the server. On the other hand, if $v_i^t = 0.2$ then sensor data is collected for five seconds and transmitted to the server at the end of each five seconds time interval.

Definition 2.1. Context Acquisition Cost. Suppose a mobile sensor m_i is collecting context data with an acquisition frequency of v_i^a and a transmission frequency of v_i^t between time t_0 and t_1 . Then the total cost of context acquisition is as follows.

$$\mathcal{A}(m_i, v_i^a, v_i^t) = (t_1 - t_0)v_i^a \mathcal{E}_i + \mathcal{S}_i \frac{v_i^a}{v_i^t} + (t_1 - t_0)v_i^t \mathcal{Y}_i \left(\frac{v_i^a}{v_i^t} \right) \quad (1)$$

where $\mathcal{E}_i$ is the energy consumption cost of one probe from m_i , $\mathcal{S}_i$ is the energy consumption cost of storing one record of m_i , and $\mathcal{Y}_i(n)$ is the energy consumption cost of transmitting n records from the mobile sensor to the server.

Evaluation Function. Input to an evaluation function is a list of context records (e.g. GPS coordinates) between time t_0 and t_1 . Output of the function is a boolean value signifying whether the context condition (e.g. a user is in the building) is true or not. For example, a sample evaluation function for GPS data can be as follows: given a list of GPS locations of a user and a target GPS location, a function which determines the geodesic distance between the list and the target location and outputs *true* if the distance is below a certain threshold. An evaluation function is denoted by the notation of f_i .

Cost of Context Evaluation. The cost of an evaluation function f_i is defined as the average execution time of the function on a single tuple of the context data. Each evaluation function has two costs: *server-running cost* and *mobile-running cost*. Server running cost, denoted by $w_s(f_i)$, refers to the execution time of the function on a single tuple on the server. Mobile running cost, denoted by $w_m(f_i)$ refers to the execution time on the mobile device.

Latency. We define the latency as the delay in detecting an event by the system from the time of occurrence of the event. Let us consider an example context $\langle \text{Location}(P_1, R_1) = \text{true} \rangle$. Suppose user P_1 reached location R_1 at 10:00:00 a.m. and SCARF detected the event at 10:02:00 a.m., then the latency will be two minutes.

Definition 2.2. Plan. Let $\mathcal{M} = \{m_1, m_2, \dots, m_n\}$ be a set of mobile sensors from which context data needs to be collected and evaluated. A plan consists of the following items.

- List of acquisition frequency values: $v^a = \langle v_1^a, v_2^a, \dots, v_n^a \rangle$.
- List of transmission frequency values¹: $v^t = \langle v_1^t, v_2^t, \dots, v_n^t \rangle$.

¹We will henceforth assume that all v_i^t values are equal to each other and we denote it by v_0^t . The reason behind this choice is that we only evaluate the context data at the end of each transmission cycle. If we transmit context data earlier and not evaluate it until the end of a transmission cycle, it will always increase the latency.

- List of function evaluation costs on the server: $w_s = \langle w_s(f_1), w_s(f_2), \dots, w_s(f_n) \rangle$.
- List of function evaluation costs on the mobile: $w_m = \langle w_m(f_1), w_m(f_2), \dots, w_m(f_n) \rangle$.
- A bit vector signifying whether the function is evaluated on the server side or on the mobile side. $B = \langle b_1, b_2, \dots, b_n \rangle$. If $b_1 = 1$ then the evaluation function is evaluated on the server side where as if $b_1 = 0$ then it is evaluated on the mobile side.

We denote a plan by the notation of $P(v^a, v^t, B)$.

Definition 2.3. Latency of a Plan. Given a plan defined as above, we define the latency of a plan $P(v^a, v^t, B)$ as follows:

$$\text{Latency}(P) = \sum_{i=1}^n \frac{v_i^a}{v_0^t} \cdot (b_i \cdot w_s(f_i) + (1 - b_i) \cdot w_m(f_i)) \quad (2)$$

In the following we provide an example of a plan and the latency of the plan.

Example 2.4. Let m_1, m_2, m_3 be three sensors from which context data needs to be collected and evaluated. Let us consider a plan where the acquisition frequency values are 2/sec, 3/sec and 4/sec respectively. Let the values of the transmission frequencies are as follows: $v_1^t = v_2^t = v_3^t = 1/30$. This implies that at the end of each 30 seconds time interval, collected context data is transmitted to the server. At the end of each thirty seconds time interval, the number of records of m_1 that needs to be evaluated are $30 \cdot 2 = 60$. Similarly, the number of records of m_2 and m_3 are 90 and 120 respectively.

Suppose the cost of functions f_1, f_2, f_3 when they are executed on the server are 0.01 sec, 0.02 sec and 0.03 seconds respectively. The cost of the functions f_1, f_2, f_3 on the mobile are 0.04, 0.015 and 0.05 respectively. Suppose the context data from m_1 and m_3 needs to be evaluated on the server and the data from m_2 needs to be evaluated on mobile, then the latency of the plan is calculated as follows: $60 \cdot 0.01 + 90 \cdot 0.015 + 120 \cdot 0.03 = 5.5$ seconds.

Problem Statement. Let $\mathcal{M} = \{m_1, m_2, \dots, m_n\}$ be a set of mobile sensors from which context data needs to be collected and evaluated. Let $\mathcal{E}^{max}$ be the maximum cost (in terms of energy consumption) that can be incurred for the context data acquisition and evaluation from $\mathcal{M}$. Our objective is to generate a plan which minimizes the latency given that the total cost of context data acquisition, storage and transmission from all the sensors is bounded by $\mathcal{E}_i^{max}$. Our optimization problem is formally defined as follows:

$$\begin{aligned} & \underset{v^a, v^t, B}{\text{minimize}} && \text{Latency}(P(v^a, v^t, B)) \\ & \text{subject to} && \sum_{i=1}^n (t_1 - t_0) v_i^a \mathcal{E}_i + \mathcal{S}_i \frac{v_i^a}{v_0^t} + (t_1 - t_0) v^t \mathcal{Y}_i \left(\frac{v_i^a}{v_0^t} \right) \leq \mathcal{E}^{max}, \\ & && v_i^a \geq v^{min} \end{aligned} \quad (3)$$

where v^{min} is the minimum frequency value which ensures that there is no missing event due to the acquisition of data being far apart on the time-line. We assume that if a context condition is true at a particular time instant, then it will remain true for at least a certain amount of time interval called $event^{min}$. If $event^{min}$ is two seconds, then we set the value of v^{min} to be 0.5. This ensures that there exists at least one data point in every two seconds. Our

goal is to choose the frequency values v^a, v^t and the bit vector B in such a way that minimizes the latency.

3 DESCRIPTION OF SCARF

3.1 Overview

SCARF divides the entire execution time into multiple smaller time intervals called epochs. In each epoch, we generate a plan as described in Definition 2.2 by solving the optimization problem shown in Equation 3. This plan is generated at the server. Depending on the outcome of the optimal solution (i.e., the bit vector B), the evaluation is either performed at the server or on the mobile. Each epoch consists of two phases: a plan generation phase and a plan execution phase. We assume that the cost of the sensors (i.e., $\mathcal{E}_i$) and the context evaluation costs (i.e., w_s and w_m) does not change within an epoch. Algorithm 1 presents a high-level overview of how SCARF works.

The input to SCARF is a list of mobile sensors, a list of context evaluation functions, a start time and end time between which the context has to be evaluated. The variables *totalExecutionTime* and *curEpochTime* maintain the overall execution time and the execution time within an epoch respectively. We maintain two hashmaps for storing the cost values of the mobile sensors w_s and w_m . The energy consumption cost $\mathcal{E}$ is updated at the end of each epoch. In line 8, we generate the plan using the updated cost values w_s and w_m . Plan is executed until the epoch time is exhausted.

Algorithm 1 Adaptive Approach.

```

1: procedure ADAPTIVE APPROACH( $\mathcal{M}, \{f_1, f_2, \dots, f_n\}, t_0, t_1$ )
2:    $totalExecutionTime \leftarrow 0$ 
3:    $contextResult \leftarrow False$ 
4:   INITIALIZE-DATA-STRUCTURE( $w_s, w_m$ ) // Initialize the data structures to store the running costs of functions on server and mobile
5:   while  $totalExecutionTime < (t_1 - t_0)$  do
6:      $curEpochTime \leftarrow 0$ 
7:      $\mathcal{E} \leftarrow \text{GET-UPDATED-COST}(\mathcal{M}, f_1, f_2, \dots, f_n)$ 
8:      $\langle P, t_p \rangle \leftarrow \text{GENERATE-PLAN}(\mathcal{E}, w_s, w_m, \Delta)$  //  $t_p$ : plan-generation time
9:     while  $curEpochTime < (epochTime - t_p)$  do
10:       $t_1 \leftarrow time()$ 
11:       $contextResult \leftarrow \text{EXECUTE-PLAN}(P)$ 
12:      if  $contextResult \leftarrow True$  then
13:        return True
14:      end if
15:       $t_2 \leftarrow time()$ 
16:       $curEpochTime \leftarrow curEpochTime + (t_2 - t_1)$ 
17:    end while
18:  end while
19:  return False
20: end procedure

```

In SCARF, we have implemented a hybrid architecture that leverages server-based centralized system with an edge-based distributed component. In the following, we describe the life-cycle of context data acquisition, context data storage and context data evaluation in the different components of SCARF.

As a first step, given a condition on context from an application, SCARF checks the sensor data from the in-situ sensors. If a particular context can be evaluated just by using the data from the in-situ

Figure 1: Diagram of the Server-based Architecture

sensors, then SCARF does not have to generate any plan for that particular context. If a particular context can not be evaluated by the data from in-situ sensors, then SCARF generates a plan for the mobile sensors according to Equation 3. Depending on the output of the bit vector B of the optimization problem, SCARF uses the following strategies. In a certain epoch, if $b_i = 1$ then SCARF collects the sensor data at the mobile device for a certain duration of time and then transmit it to the server. Server stores the data in a local database and runs the context evaluation function on that data. If $b_i = 0$, then the mobile device collects the sensor data, stores it in a local database and also executes the evaluation function on the collected data. The process of context evaluation on the server side is shown in Figure 1 whereas the process of context evaluation on the mobile side is shown in Figure 2.

3.2 Plan Generation

At the beginning of each epoch, SCARF performs this step. In this step, SCARF generates an acquisition plan of the sensors, transmission plan and an evaluation plan whether the context will be evaluated in the server side or on the mobile side. It solves the optimization problem as described in Equation 3 and determines the above plans. Note that, the constraint of the equation has following coefficients: $\mathcal{E}_i$, $\mathcal{S}_i$ and $\mathcal{Y}_i$ which need to be updated in every epoch.

The first step of plan generation phase of a certain epoch k , is to update the cost values $\mathcal{E}_i$, $\mathcal{S}_i$ and $\mathcal{Y}_i$ respectively. We determine the average energy consumption costs incurred in the previous epoch (i.e., epoch $k-1$) and set it as the value of $\mathcal{E}_i$ in epoch k . For example, suppose in a certain epoch $k-1$, the total energy consumed by 100 probes of a certain mobile sensor is 0.02. Then the value of $\mathcal{E}_i$ in epoch k will be as follows: $\mathcal{E}_i = 0.02/100 = 0.0002$. Similarly, let the values of $\mathcal{S}_i$ and $\mathcal{Y}_i$ are 0.0001 and 0.004 respectively.

Once the cost values are updated we solve the optimization problem as shown in Equation 3. In the above example, if we add the values of the $\mathcal{E}_i$, $\mathcal{S}_i$, $\mathcal{Y}_i$ in Equation 3, we get the following optimization problem:

$$\begin{aligned}
 & \text{minimize}_{v^a, v^t, B} \quad \sum_{i=1}^n \frac{v_i^a}{v_0^t} \cdot (b_i \cdot w_s(f_i) + (1 - b_i) \cdot w_m(f_i)) \\
 & \text{subject to} \quad (t_1 - t_0)v_i^a \cdot 2 \cdot 10^{-4} + \frac{v_i^a}{v_0^t} \cdot 10^{-4} \\
 & \quad \quad \quad + (t_1 - t_0)v_0^t \cdot 4 \cdot 10^{-3} \leq \mathcal{E}_i^{max} \\
 & \quad \quad \quad v_i^a \in \mathbb{R}, v_0^t \in \mathbb{R}, b_i \in \{0, 1\} \\
 & \quad \quad \quad v_i^{max} \geq v_i^a \geq v_i^{min}, v_0^{max} \geq v_0^t \geq v_0^{min}
 \end{aligned} \tag{4}$$

The frequency value v_i^{max} denotes the maximum frequency of data collection that is supported by the operating system of the mobile device. The above optimization problem is a mixed integer non-linear programming problem where the values of v_i^a and v_0^t are real numbers and the value b_i is an integer which can take two values 0 or 1. Since the domains of v_i^a and v_0^t are bounded, we solve the above optimization problem numerically. We discretize the domain of v_i^a and v_0^t from the values of v_i^{min} to v_i^{max} using a width of 0.01. For each pairs, we check the constraint condition to find out if the pair is a feasible solution of the optimization problem. If the solution is feasible, then we calculate the objective function and store the value and the solution in a minimum priority queue. After iterating over the space of v_i^a and v_0^t , we retrieve the optimal solution from the priority queue.

3.3 Context Data Collection

Given a set of mobile sensors $m_1, m_2, \dots, m_n$ from which context data have to be collected, SCARF generates an acquisition plan (i.e., v_i^a) for each sensors. Note that the energy consumption cost of a sensor in a mobile device is dependent on several factors such as the operating system installed, underlying hardware, and type of the network to which the device is connected to.

In a certain epoch, if the context data needs to be evaluated on the server side, then SCARF uses a push-based mechanism to collect the data. The sensor data gets collected at the mobile device

at a frequency of v_i^a till the end of the epoch. At the end of the epoch, the mobile device pushes the collected data to the server side. Mobile device uses appropriate RESTful API to transmit the data to the server. The server stores the collected data in an internal database.

In an epoch, if the context data needs to be evaluated on the mobile side, then the data collection takes place on the mobile device itself. The sensor data is collected and stored at a frequency of v_i^a at the mobile device till the end of the epoch. Since the context data is evaluated on the mobile device itself, the collected data is not transmitted from the mobile device to the server in this epoch. The context data is stored in a SQLite database of the mobile device.

3.4 Context Data Management

Context data collected is stored in a database called *ContextDB*. In the following, we explain how this database is managed both on the server-side as well as on the mobile side. Context Reasoner (as explained in the next section) queries this database and evaluate the required context. Query performance of ContextDB affects the latency of the end applications.

In the server-side, we maintain a centralized database, which stores the context information collected from all the mobile devices. The schema of the database is shown as follows:

- OBSERVATION (id INT PRIMARY KEY, sensor_id INT, payload VARCHAR(255), observation_type_id INT, timestamp datetime)
- USER (user_id INT PRIMARY KEY, name VARCHAR(255), email VARCHAR(255), office VARCHAR(255), mobile_number VARCHAR(20), preferred_communication VARCHAR(255))
- SENSOR (sensor_id INT PRIMARY KEY, description VARCHAR(255), name VARCHAR(255), platform_id INT)
- PLATFORM (platform_id INT PRIMARY KEY, description VARCHAR(255), name VARCHAR(255))
- PLATFORM_SENSOR (platform_id INT, sensor_id INT)

In the above schema, the OBSERVATION table contains the collected context data from the mobile sensors. In this table, sensor_id contains the id of the mobile sensor, observation_type_id contains the type of the context data such as GPS, accelerometer, gyroscope, light, etc. Payload attribute contains the actual value of the sensor and it is stored as a JSON object. For example, for a GPS sensor, a sample payload will be as follows: {"Latitude": "33.6431", "Longitude": "-117.84"}. The timestamp attribute captures the time at which the observation was collected by the mobile sensor. Context reasoner queries this table based on the sensor_id, timestamp and observation_type_id attributes. We created indexes on these attributes of OBSERVATION table for better query performances by the reasoner.

In the mobile-side implementation, the schema of the ContextDB is as follows.

- GPS (id INTEGER PRIMARY KEY, latitude FLOAT, longitude FLOAT, accuracy FLOAT, provider VARCHAR(255), timestamp DATETIME)
- ACCELEROMETER (id INTEGER PRIMARY KEY, x_coordinate FLOAT, y_coordinate FLOAT, z_coordinate FLOAT, timestamp DATETIME)

- GYROSCOPE (id INTEGER PRIMARY KEY, x FLOAT, y FLOAT, z FLOAT, timestamp DATETIME)
- LIGHT (id INTEGER PRIMARY KEY, light FLOAT, timestamp DATETIME)

In an epoch, if a context is evaluated on the mobile side, then after the context evaluation step, we remove all the records from the tables. Since mobile devices have a limited storage capacity, this ensures that none of the tables become very large. Furthermore, we create indexes on the timestamp attribute of the above tables for faster retrieval of the records based on the timestamp attribute.

3.5 Context Data Evaluation

The process of context data evaluation consists of multiple steps. In the first step, it fetches the required context data from ContextDB. The context data collected only in the current epoch is fetched from the database. For example, if the length of an epoch is one minute, then the data collected in last one minute is fetched from the ContextDB. In the following, we define the specifications of a context reasoner, and then we explain how the context evaluation takes place on both the server side and on the edge-side.

Context Reasoner. The input to a context reasoner is the context condition and a list of context data of a particular type. For example, if the context is location, then it takes an input as a set of GPS records. The output of the context reasoner is a boolean value of *True* or *False*. If the context reasoner outputs true at any point, then it is not further evaluated in the future.

In the epochs, where the context data is evaluated on the server side, context reasoner fetches the data of the last epoch from the ContextDB and checks whether the context condition is satisfied or not. We deploy the reasoner at the same host where ContextDB is hosted to reduce the network I/O of fetching the data from the database to the server. Context Evaluator on the server is multi-threaded.

In an epoch, if the context data needs to be evaluated on the mobile side, the reasoner takes the following steps. It fetches the records from the local ContextDB and executes the reasoner on top of the fetched data. If the evaluation of the context becomes true, then it sends a *success* message to the server signifying that the context evaluation was successful and further context evaluations should be stopped.

Benefits and Limitations of the Server and Edge Based Approaches. In the server-based approach, we perform the context evaluation in the same server in which we maintain the Context DB. The benefits of the context evaluation step on the server side are as follows:

- SCARF can execute complex context reasoners such as machine learning based algorithms, complex rule-based algorithms on the central server. SCARF can use deep learning based algorithms which require special hardware such as GPGPUs to execute.
- The server-side component does not have any limitations on the storage size. The size of the Context DB can potentially be very large without causing any performance issue.
- ContextDB in the server-side can be replicated appropriately to support context-aware applications during node failures.

The advantages of the edge-based component are as follows:

Figure 2: Diagram of Edge-based Architecture

- Since ContextDB is maintained on the user’s mobile devices, sensitive context data (such as location and activity) is very less prone to attack by the attackers.
- The context evaluation step is performed by the mobile device itself, which implies that the server only gets to receive the output of the context evaluation step. Malicious users on the server will not be able to perform any inference attacks on the context data.

4 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate our framework using real world dataset. In the following, we first explain the experimental setups, then present experimental results.

Figure 3: Figure illustrating the context-aware message application.

Implementation and Deployment Specifications. We implemented both the components (server and edge) of SCARF in Java.

In the server-side implementation, we use the MySQL database for managing Context DB. In the edge-side implementation, we use SQLite database. Our server has the following specifications: Intel(R) Xeon(R) CPU E5-46400, 132 GB RAM, 80 TB of disk space and runs CentOS 7.1. Mobile devices have the following specifications: i) Motorola Moto X (2nd Generation): 2.5GHz Qualcomm Snapdragon 801 processor, 16 GB internal storage, 2GB RAM, running Android 6.0 and ii) Samsung Galaxy 5: 2.1 GHz quad-core Cortex-A15 and 1.5 GHz quad-core Cortex-A7 CPU, internal storage of 16GB, 2 GB LPDDR3 RAM running Android 5.0.

Sensors. We consider five different types of mobile sensors: accelerometer, location (gps and network), gyroscope, light, and proximity. These data is stored on Android SQLite database as described in section 3.4. For accelerometer sensor, we store the accelerometer values along the directions of x-, y-, and z-axis. For GPS, we store latitude, longitude, accuracy, and timestamp values. For proximity sensors, we store the proximity values measured in centimetres. For light sensors, we store the values of light levels measured in terms of luminous flux (i.e., lux). For gyroscope sensors, we store the rate of rotation values along the directions of x-, y-, and z-axis. For in-situ sensors, we have used the sensors deployed as part of a live IoT testbed named TIPPERS [28]. This testbed is collecting real time sensor data from 64 WiFi access point sensors, 40 video cameras, 50 beacons and 6000 HVAC sensors located in our department building since January 2016.

Context-Aware Messages Used. We use the following context-aware messages to evaluate our framework.

- 1) M_1 : **If** $Location(P_1, "R_1") = true$ **then** send a message.
- 2) M_2 : **If** $Activity(P_1, "still") = true$ **then** send a message.
- 3) M_3 : **If** $Location(P_1, "R_1") = true$ **AND** $Activity(P_1, "still") = true$ **then** send a message.
- 4) M_4 : **If** $Occupancy(R_1) = 0$ **AND** $Location(P_1, "DBH") = true$ **then** send a message.

In the above messages, first message contains a building-centric context and the second message contains a user-centric context. The third message consists of two user-centric contexts and the fourth message consists of a building-centric and a user-centric context. Although we have chosen the above four representative

Epoch Size	Latency (Sec.) of M_1	Cost of M_1	Latency (Sec.) of M_2	Cost of M_2	Latency (Sec.) of M_3	Cost of M_3	Latency (Sec.) of M_4	Cost of M_4
10 s	4.5134	6820.8	4.5019	3460.8	4.5057	6820.8	4.5230	6720
20 s	9.5099	5090.4	9.5008	3393.6	9.5055	5124	9.5141	6770.4
30 s	14.5099	3393.6	14.5003	3388	14.5048	4536	14.5107	6753.6
40 s	19.5107	3376.8	19.5003	3385.20	19.5043	4250.4	19.5103	5065.2
50 s	24.5099	3380.16	24.5001	2708.16	24.5044	4737.6	24.5096	5389.44
60 s	29.5034	2811.2	29.5001	2822.40	29.5040	3948	29.5093	5616.8
70 s	34.5068	2889.6	34.5002	2894.40	34.5044	4348.8	34.5093	4809.6
80 s	39.5064	2528.4	39.5001	2528.40	39.5044	4636.8	39.5088	5048.4
90 s	44.5051	2628.27	44.5001	2254.93	44.5048	4132.8	44.5091	4487.47
100 s	49.5022	2358.72	49.5001	2365.44	49.5048	4391.52	49.5096	4717.44
110 s	54.5033	2455.85	54.5001	2458.91	54.5021	3970.91	54.5090	4282.47
120 s	59.5033	2251.2	59.5001	2254	59.5044	3920	59.5091	4488.4

Table 2: Average Latency and Cost (in mAh) of M_1 , M_2 , M_3 and M_4 at different epoch size values.

Figure 4: Figure (a) illustrates latency of messages. Figure (b) illustrates cost of messages.

contexts, it can be specified for any building-centric contexts and user-centric contexts. Figure 3 shows the interface for creating these context-aware messages.

4.1 Experimental Result.

Experiment 1. Latency of SCARF at Different Event Times.

In our experiments we set the value of $event^{min}$ to be 120 seconds, i.e., when a particular context condition becomes true, it remains true for the duration of next 120 seconds. This implies that SCARF needs to set the transmission frequency to be at least once in 120 seconds in order to ensure that there is no missing event. Since latency is sensitive to the actual time of event occurrence, we measure the latency of SCARF for a given value of epoch size (i.e., $1/v_0^t$) as follows.

Let us consider that the epoch size for a certain message is 10 seconds, i.e., after every ten seconds, the context data is transmitted and evaluated. An event can take place at $T_{event} = 1$ second, 2 seconds, $\dots$, or at 10 seconds. We measure the latency of SCARF in each of these ten possibilities and derive the average latency. We report this value as the latency of SCARF at the epoch size of 10 seconds. Similarly, we calculate the average latency for the other

epoch sizes as shown in Table 2 and Figure 4. The acquisition frequency value for this experiment was set up at 10.

The results for all the messages are shown in Table 2 and Figure 4. From this table, we conclude that SCARF performs very well in terms of the latency of these messages across different event times. For M_1 , the average latency of SCARF varies from 4.5134 seconds to 59.5033 seconds depending on the time at which the context condition was true. Furthermore, we observe that the latency of the messages increase when the epoch size is high (i.e., transmission frequency is low) but at the same time the cost reduces. Depending on the value of $\mathcal{E}^{max}$, SCARF chooses appropriate transmission frequency.

Experiment 2. Latency Comparison with the Baseline Approaches. In this experiment, we measure the latency of SCARF with two baseline approaches. For baseline 1 approach, we perform the context acquisition, context storage, and context evaluation entirely on the mobile side. In baseline 2 approach, we perform all the three steps (i.e., context acquisition, context storage, and context evaluation) completely on the server side.

$\mathcal{E}^{max}$ (Battery Level Drop)	Baseline 1 (ms)	Baseline 2 (ms)	Our Approach (ms)
250 mAh	90.67	145.45	86.33
500 mAh	139.67	247.56	123.67
750 mAh	197.67	251.78	163.67
1000 mAh	277.33	260.89	226.33
1250 mAh	331.33	366.89	277

Table 3: Average Latency comparison of $\mathcal{M}_1$

Figure 5: Comparison of SCARF with baseline approaches.

We choose different values of $\mathcal{E}^{max}$ for each message and compare the latency (measured in milliseconds) of our approach with the baseline approaches. $\mathcal{E}^{max}$ is the maximum energy consumption cost set up for each messages. For all the messages, the latency of SCARF is lower than both *Baseline 1* and *Baseline 2*. The latency values of different approaches for $\mathcal{M}_1$ are shown in Table 3 and a comparison plot is drawn in Figure 5. The reason behind the performance gap of our approach and the baseline approaches is as follows: our approach chooses an optimal acquisition frequency, transmission frequency and bit vector B specifying if the context should be evaluated on the server or on the mobile side in order to minimize the latency.

Experiment 3. Energy Consumption Cost. In this experiment, we provide the details of the energy consumption cost from different types of sensors. The results are shown in Table 4. For each type of sensors, we measure the probe cost of one sensor probe and report it as the probe cost in Table 4. We measure the cost of transmitting one record of a particular sensor type from the mobile device to the server and report it as the transmission cost. We performed this step for 1000 iterations and measure the average transmission cost from the total cost. In the evaluation cost column, we provide the energy consumption cost of evaluating the context reasoner on a single record. From this experiment, we draw the following conclusions: the probe cost, transmission cost and evaluation cost vary depending on the type of the device, type of hardware, operating system and the type of network. Furthermore, this implies that the cost model of SCARF as shown in the constraint of Equation 3 is an accurate and practical model for measuring the cost of context acquisition, transmission, and evaluation.

5 RELATED WORK

Context-aware applications have been studied by several authors in the past [29–32]. Context data which were popularly used by these applications were location, activity, timestamps, schedule and a combination of them. In our opinion, following factors affect

Sensor	Device	Probe Cost	Transmission Cost	Evaluation Cost
GPS	Device 1	4.6	0.092	0.76
Accelerometer	Device 1	0.023	0.46	0.012
Gyroscope	Device 1	0.383	0.18	0.0046
GPS	Device 2	1.6	0.168	0.93
Accelerometer	Device 2	0.028	0.056	0.014
Gyroscope	Device 2	0.56	0.168	0.0056

Table 4: Details of Cost in *Device 1* and *Device 2* measured in mAh.

the scalability of any context-aware application: context data management system and energy efficiency of context data collection, transmission and evaluation.

Systems for Context-Aware Applications. A number of authors have identified the requirements, implementation, and major research challenges of systems that support context-aware applications [33–35]. According to the authors in [34], existing implementations can be categorized according to the functionalities provided by these systems such as the capability of collection, storage, query, prediction, and context representation. To the best of our knowledge, we could not find any work that addresses the problem of minimizing the latency of the context-aware applications while maintaining a bound on the energy consumption cost.

Energy Efficiency of Context-Aware Applications. Several authors [36–39] in the past highlighted the importance of energy efficiency of context-aware mobile applications. In [40], authors have proposed an efficient context data processing pipeline where an early feature detection stage can eliminate the usage of expensive context recognition. Although such an approach can minimize the overall energy consumption cost of mobile devices, but it can incur high latency depending on the complexity of the context evaluation function. In contrast, our hybrid architecture chooses to collect, transmit and evaluate context data appropriately either on the mobile side or on the server side to overcome the latency barrier of the end applications.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a scalable data management framework for context-aware applications in smart environment. Context-Aware applications are emerging to be very powerful applications in IoT domain as the number of infrastructure sensors and mobile sensors are growing. We have proposed SCARF that utilizes both centralized and edge-based approach of context evaluation. We have empirically shown that the acquisition cost can be appropriately tuned to minimize the overall latency of context evaluation.

We would like to continue our research towards improving the scalability of frameworks for context-aware applications. We are investigating how to evaluate more complex contexts such as contexts including large number of conjunctive and disjunctive conditions. We are also looking into the problem where the context evaluation is inaccurate in nature (i.e., the evaluation function can make mistakes while evaluating a context). One possibility would be to use multiple context evaluation functions to evaluate a single context and then decide whether the context condition is true or not.

Another direction that we plan to investigate is how to enhance privacy and security of the end users who are using context-aware

applications. Several works exist on how to perform activity detection, location detection in a privacy-preserving manner, but a generalized framework which is capable of evaluating large number of context conditions in a privacy-preserving manner is an interesting direction of future work.

ACKNOWLEDGEMENT

This material is based on research sponsored by DARPA under agreement number FA8750-16-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. This work is partially supported by National Science Foundation under Grant No. 1527536, 1545071 and 1544969.

REFERENCES

- [1] Omer Berat Sezer et al. Context-aware computing, learning, and big data in internet of things: a survey. *IEEE Internet of Things Journal*, 5(1):1–27, 2017.
- [2] JP Black et al. Pervasive computing in health care: Smart spaces and enterprise information systems. In *MobiSys 2004 Workshop on Context Awareness*, 2004.
- [3] Andreas Holzinger et al. From smart health to smart hospitals. In *Smart health*. Springer, 2015.
- [4] Antonio Coronato et al. Towards an implementation of smart hospital: A localization system for mobile users and devices. In *2008 International Conference on Pervasive Computing and Communications (PerCom)*.
- [5] Agusti Solanas et al. Smart health: a context-aware health paradigm within smart cities. *IEEE Communications Magazine*, 52(8), 2014.
- [6] Yunyoung Nam et al. Intelligent video surveillance system: 3-tier context-aware surveillance system with metadata. *Multimedia Tools and Applications*, 57(2), 2012.
- [7] Vio Onut et al. Smart surveillance system applications. In *Proceedings of the 2010 conference of the center for advanced studies on collaborative research*. IBM Corp., 2010.
- [8] Kyle Benson, Charles Fracchia, Guoxi Wang, Qiuxi Zhu, Serene Almomen, John Cohn, Luke DăăZarcy, Daniel Hoffman, Matthew Makai, Julien Stamatakis, et al. Scale: Safe community awareness and alerting leveraging the internet of things. *IEEE Communications Magazine*, 53(12):27–34, 2015.
- [9] Md Yusuf Sarwar Uddin, Alexander Nelson, Kyle Benson, Guoxi Wang, Qiuxi Zhu, Qing Han, Nailah Alhassoun, Prakash Chakravarthi, Julien Stamatakis, Daniel Hoffman, et al. The scale2 multi-network architecture for iot-based resilient communities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8. IEEE, 2016.
- [10] Hang Nguyen, Md Yusuf Sarwar Uddin, and Nalini Venkatasubramanian. Multi-stage adaptive load balancing for big active data publish subscribe systems. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, pages 43–54. ACM, 2019.
- [11] Md Yusuf Sarwar Uddin, Vinay Setty, Ye Zhao, Roman Vitenberg, and Nalini Venkatasubramanian. Richnote: Adaptive selection and delivery of rich media notifications to mobile users. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 159–168. IEEE, 2016.
- [12] Md. Yusuf Sarwar Uddin and Nalini Venkatasubramanian. Edge caching for enriched notifications delivery in big active data. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pages 696–705, 2018.
- [13] H. Jafarpour, B. Hore, S. Mehrotra, and N. Venkatasubramanian. Ccd: A distributed publish/subscribe framework for rich content formats. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):844–852, May 2012.
- [14] Salvatore Longo et al. Enriching shopping experiences with pervasive displays and smart things. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication, UbiComp '13 Adjunct*, New York, NY, USA, 2013. ACM.
- [15] Gonzalo Garcia-Perate et al. Ambient recommendations in the pop-up shop. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 773–776. ACM, 2013.
- [16] Rosslin John Robles and Tai-hoon Kim. Context aware tools for smart home development. *International Journal of Smart Home*, 4(1), 2010.
- [17] Son N Han et al. Semantic context-aware service composition for building automation system. *IEEE Transactions on industrial informatics*, 10(1), 2013.
- [18] Unai Alegre, Juan Carlos Augusto, and Tony Clark. Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, 117:55–83, 2016.
- [19] Victor R Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on knowledge and data engineering*, 11(1), 1999.
- [20] Philip Moore et al. Fog computing and low latency context-aware health monitoring in smart interconnected environments. In Leonard Barolli, Fatos Xhafa, Nadeem Javaid, Evjola Spaho, and Vladi Kolic, editors, *Advances in Internet, Data & Web Technologies*, pages 29–40. Cham, 2018. Springer International Publishing.
- [21] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Latency-aware application module management for fog computing environments. *ACM Transactions on Internet Technology (TOIT)*, 19(1):9, 2018.
- [22] Altti Ilari Maarala, Xiang Su, and Jukka Riekkii. Semantic reasoning for context-aware internet of things applications. *IEEE Internet of Things Journal*, 4(2):461–473, 2016.
- [23] Nishkam Ravi, James Scott, Lu Han, and Liviu Iftode. Context-aware battery management for mobile phones. In *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 224–233. IEEE, 2008.
- [24] Narseo Vallina-Rodriguez, Pan Hui, Jon Crowcroft, and Andrew Rice. Exhausting battery statistics: understanding the energy demands on mobile handsets. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, pages 9–14. ACM, 2010.
- [25] Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. Self-adaptive battery and context aware mobile application development. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 761–766. IEEE, 2014.
- [26] Nishkam Ravi, James Scott, and Liviu Iftode. Context-aware battery management for mobile phones: A feasibility study. In *Sixth Annual IEEE International Conference on Pervasive Computing and Communications, 2008. PerCom 2008, 2006*.
- [27] Simon Jones et al. efriend: an ethical framework for intelligent environments development. *Ethics and Information Technology*, 2015.
- [28] Sharad Mehrotra et al. TIPPERS: A privacy cognizant iot environment. In *IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*.
- [29] Yasuto Nakanishi et al. Context aware messaging service: A dynamical messaging delivery using location information and schedule information. *Personal Technologies*, 4(4):221–224, 2000.
- [30] Anind K Dey and Gregory D Abowd. Cybreminder: A context-aware system for supporting reminders. In *Handheld and Ubiquitous Computing*, pages 172–186. Springer, 2000.
- [31] A. Almagrabi et al. Towards context-aware messaging for emergency situations. In *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, pages 1116–1121, Oct 2012.
- [32] Stephen Knox et al. Strategy-proof thermal comfort voting in buildings. *Revue d'Intelligence Artificielle*, 22:pp 549–568, June 2008.
- [33] Paolo Bellavista et al. A survey of context data distribution for mobile ubiquitous systems. *ACM Comput. Surv.*, 44(4):24:1–24:45, September 2012.
- [34] P. Makris et al. A survey on context-aware mobile and wireless networking: On networking and computing environments' integration. *Communications Surveys Tutorials, IEEE*, 15(1):362–386, First 2013.
- [35] O. Yurur et al. Context-awareness for mobile sensing: A survey and future directions. *Communications Surveys Tutorials, IEEE*, PP(99):1–1, 2014.
- [36] Vladimir Jovanovikj et al. Security policy model for ubiquitous social systems. In *CONTEXT 2017*.
- [37] Nailah Saleh Alhassoun et al. Context-aware energy optimization for perpetual iot-based safe communities. *Sustainable Computing: Informatics and Systems*, 2019.
- [38] Ashraf Alkhresheh et al. Context-aware automatic access policy specification for iot environments. In *IWCMC. IEEE*, 2018.
- [39] Gaoqiang Zhuo and Huihui Yang. Privacy-preserving context-aware friend discovery based on mobile sensing. In *ICCE. IEEE*, 2018.
- [40] Seungwoo et al. Seemon: Scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. *Personal Technologies*, 4(4):267–280, 2008.