Open Access

Open Journal of Internet of Things (OJIOT)
Volume 3, Issue 1, 2017

www.ronpub.com/ojiot
ISSN 2364-7108

Latency Optimization in Large-Scale Cloud-Sensor Systems

Adhithya Balasubramanian [A], Sumi Helal [B], Yi Xu [C]

[A] University of Florida, CISE department, Gainesville, FL 32611, USA, adhithyab@ufl.edu
[B] Lancaster University, School of Computing and Communication, Bailrigg, Lancaster LA1 4YW, UK, s.helal@lancaster.ac.uk
[C] Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA, yix@google.com

Abstract

With the advent of the Internet of Things and smart city applications, massive cyber-physical interactions between the applications hosted in the cloud and a huge number of external physical sensors and devices is an inevitable situation. This raises two main challenges: cloud cost affordability as the smart city grows (referred to as economical cloud scalability) and the energy-efficient operation of sensor hardware. We have developed Cloud-Edge-Beneath (CEB), a multi-tier architecture for large-scale IoT deployments, embodying distributed optimizations, which address these two major challenges. In this article, we summarize our prior work on CEB to set context for presenting a third major challenge for cloud sensor-systems, which is latency. Prolonged latency can potentially arise in servicing requests from cloud applications, especially given our primary focus on optimizing energy and cloud scalability. Latency, however, is an important factor to optimize for real-time and cyber-physical applications with limited tolerance to delays. Also, improving the responsiveness of any IoT application is bound to improve the user experience and hence the acceptability and adoption of smart city solutions by the city citizens. In this article, we aim to give a formal definition and formulation for the latency optimization problem under CEB. We propose a Prioritized Application Fragment Caching Algorithm (PAFCA) to selectively cache application fragments from the cloud to lower layers of CEB, as a key measure to optimize latency. The algorithm itself is an extension of one of the existing optimization algorithms of CEB (AFCA-1). As will be shown, PAFCA takes into account the expectations of cloud applications on real-timeliness of responses. Through experiments, we measure and validate the effect of PAFCA on latency and cloud scalability. We also introduce and discuss the trade-off between latency and sensor energy in this given context.

Type of Paper and Keywords

Regular research paper: *cloud-sensor systems, latency optimization, application caching, cloud scalability*

1 Introduction

As the smart city concept proliferates into a massive scale, smart city applications are bound to be pushed to the cloud where they can be hosted and executed. This is not only due to the cloud's economies of scale but also because numerous stakeholders will demand access to sensor data and the services (applications), which is difficult to achieve without a neutral and a common platform like the cloud. But connecting

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2017)* in conjunction with the VLDB 2017 Conference in Munich, Germany. The proceedings of VLIoT@VLDB 2017 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

hundreds of millions of sensors and devices directly to the cloud is bound to result in massive traffic and unbounded use of cloud resources. Utilizing edge computing and connecting devices to the cloud through edge computers have been shown to be a promising approach to manage growth in the smart city and to achieve economical scalability of the cloud, by slowing down its elasticity rate as more devices and applications are deployed [22]. We have developed the cloud-edge-beneath (CEB) architecture to address the economical scalability challenge and to minimize the energy used by the sensors and devices.

CEB embodied a distributed optimization framework deployed at all three layers of the architecture. Four optimization algorithms in CEB have been designed to minimize movements of application requests down to the sensors, and movements of data updates from the sensors up to the applications. Also, sensor sampling is minimized and a new guiding principle which we call sentience efficiency is applied. In a nutshell, sentience efficiency refers to the extent that data sampled from sensors and moved up or made available to applications is actually necessary to the proper execution of the application. In other words, a sentient efficient cloud-sensor system is one that cleverly avoids any unnecessary sensing. To implement our optimization algorithms which we will briefly summarize in this paper, we introduced a bi-directional waterfall optimization framework that coordinates the interplay among the four algorithms, and that correlates application characteristic dynamics with data change dynamics, again to minimize all movements. While our prior work focused on energy savings in the beneath layer and the cloud economical scalability in the cloud layer, it left latency and real-timeliness unaddressed.

In this paper, we extend our CEB optimization approach to address latency, and to continue to optimize cloud scalability and sensor energy use under deadline constraints. We formally capture time and deadlines into the architecture and introduce a new, fifth algorithm that aims to bridge the gap between our prior optimization goals and its potential latency side effect.

2 STRUCTURE OF ARTICLE

This paper is organized as follows. Section 3 gives an overview of the CEB architecture, states the challenges that are addressed, and briefly discusses the distributed optimization framework of CEB. Section 4 presents the important related works. Section 5 formally defines and formalizes latency in CEB and analyzes the potential causes for high latency inherent in the architecture. It also discusses the different possible latency optimization problems and states the ones addressed in this paper. Section 6 proposes a solution to the chosen latency optimization problem, discusses the trade-off between latency and sensor energy and presents the proposed algorithm. Section 7 evaluates the effectiveness of the proposed algorithm and shows the trade-off between cloud scalability and latency through simulation experiments. Conclusion and future work are presented in section 8.

3 BACKGROUND

In this section, we briefly describe our prior work on Cloud-Edge-Beneath (CEB) architecture whose understanding is required for our proposed work in this article to be clearly explained. CEB is a three-tier architecture and a framework for deploying and managing cloud-sensor systems whose applications are programmed, hosted and run on the cloud [22]. The architecture is intended to enable an ecosystem for developing and deploying smart city applications in the cloud. Figure 1 shows an abstracted high level view of the architecture. The beneath layer refers to the sensors/devices and their sensor platforms which are low power computing and communication platforms, connecting related sensors (e.g., belonging to same geographical area, organization or an authority) to a corresponding edge. Edge layer groups related sensors and connects them to the cloud. Deploying and powering up devices under CEB makes the devices automatically externalized and represented in the cloud as software services. This “externalization” concept introduced by CEB has panned out to other emerging architectures such as the ARM mbed [1] in which device cloud services can be generated as RESTful services.

Automatic externalization immediately enables developers to program and deploy smart city applications in a practical fashion that decouples physical device deployment from application development. CEB is built on top of Atlas [7], which is an implementation of the Service Oriented Device Architecture (SODA) model [5]. For every beneath device connected to an Atlas node, a corresponding basic service is automatically created on the edge. And for every basic edge service, there exists a corresponding replica basic service created also automatically on the cloud, and managed by an instance of the Atlas Cloud Middleware (ACM) at the cloud layer. ACM acts as a cloud gateway to the edge. It hosts all cloud sensor service bundles passed from the edge and provisions them (and makes them accessible) to developers with permissions as services ready to be subscribed to, by other cloud services and applications.

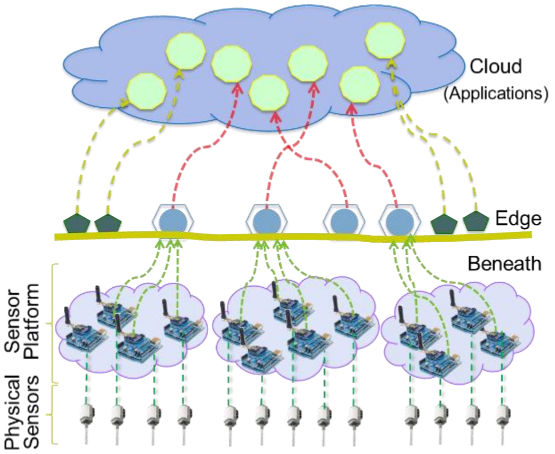

Figure 1: Overview of CEB Architecture

Sensor based services and applications are developed, deployed and run in a container called Cloud Application Runtime (CAR) at the cloud layer. Both the edge and the cloud layers use the Open Services Gateway Initiative - OSGi [13] as their basis to provide service registration, discovery, activation and configuration. The current implementation of CEB is based on an event driven application model called E-SODA [22], where the applications do not only utilize raw sensor data, but are also able to tune to specific events ranging from simple to complex events. We describe E-SODA briefly in section 3.2.

3.1 Challenges Addressed

The two key challenges that have been addressed by CEB are the cloud scalability challenge and the sensor energy challenge. We first discuss cloud scalability. Extensive interactions between sensors and cloud services could pose a challenge on the scalability of the cloud. In a smart city scenario, with millions of sensors requiring 'cloud attention' for every duty cycle, there would be billions of interactions every day which requires tremendous processing power, memory and huge incoming/outgoing cloud traffic, leading to a heavy draw on the costly cloud elasticity. Given the existing use based price models, the cloud would become too expensive as the economies of scale per sensor will not stand. This will be a major show stopper given that smart cities' main motivation is bridging supply and demand in face of increased urbanization and decline of resources and budget to spend per capita. So it would not make any sense to spend unbounded amount of money on cloud services monthly while trying to meet ends and bridge gaps! CEB has been architected to slow down cloud elasticity in face of growing demands of applications or expanding instrumentation of the smart city. CEB utilize optimizing algorithms to lean back on and exploit power-unconstrained edge servers and even beneath nodes to tackle some of the work, which effectively contains elasticity and enhances cloud economic scalability.

The second key challenge is maintaining the energy constraints of the sensor devices. Most of the sensors are generally battery powered, and this makes them vulnerable to power drainage. In a smart city scenario, a sensor may be queried by hundreds of applications, each requiring continuous evaluation of events based on sensor readings. This could lead to continuous sampling of the sensors. Without optimizations, the energy of the sensors might deplete rapidly, rendering them unreliable and unavailable. We have addressed this challenge by several optimization algorithms coordinated via a bi-directional waterfall optimization framework [20], which essentially renders a distributed optimization occurring at each layer of CEB. The same framework is used as the context in which our proposed latency optimization algorithm is coordinated, and hence, we briefly explain this framework first in the next section.

3.2 Bi-Directional Waterfall Optimization

In CEB, cloud applications request data from the physical layer which reaches the cloud through the edge layer. In our Bi-directional waterfall optimization framework [20], in addition to data being cached up from beneath to the edge and ultimately to the cloud, "application fragments" are also cached down from the cloud to the edge and even beneath, opening up a number of optimization opportunities. Application caching requires that the application model is inherently divisible (that is, an app can be divided into communicating parts easily). Pub/Sub, Event-driven and functional programming based application models are inherently divisible. CEB utilizes an event-driven model known as E-SODA which enables the concept of application fragment caching.

Under E-SODA, a complex event (cloud application) could be diced into a number of smaller complex events each represented by an Event Representation Tree (ERT), which can be cached at the lower layers. Caching an application fragment means caching a subtree of events from an ERT. A cached event is evaluated at the layer it is cached to and its event value is pushed back to its upper layers only when the value changes from its previous state. This is called 'selective push'. For any event cached to a lower layer, a 'shadow event' is created to act as a proxy of the cached event at the upper layers. This shadow event receives the selective push messages from the layer below it. The edge is of strategic importance here, because it has a view of both the data and the application domains and could potentially analyze how the data and applications interplay. This enables powerful optimizations which take the sensor data and the cloud applications as inputs to the optimization equation, thereby addressing the two key challenges mentioned above. The four optimization algorithms implemented at the three layers of CEB are briefly described below and their interplay explained.

Cloud-to-Edge Application Fragment Caching Algorithm (AFCA-1) [21] – cloud scalability: AFCA-1 selects application fragments (ERTs) from the cloud to cache at the edge layer to address the cloud scalability challenge and at the same time, staying within the limitations of the resources in edge servers (memory and processing power). Edge servers are not elastic as cloud.

Shortcut Evaluation and Branch Permutation Algorithm (BPA) [19] – saving sensor energy: In processing the application fragments cached at the edge layer, shortcut evaluation can be utilized when a subset of sensor data is sufficient to derive the occurrence of an event, saving the sensor power due to the skipped sensor samplings. For example, consider an event *A*, represented as an ERT, which is evaluated as *A = B AND* C, where *B and* C are two other ERTs (children or subtrees of *A*) and the value of *A* is calculated by performing a Boolean *AND* operation on its children *B* and *C.* If the value of *B* is known to be 0, then there is no necessity to evaluate and calculate *C's* value to determine the value of *A.* Branch permutation algorithm does exactly this and prunes ERTs if the occurrence of those events could be figured out without looking at the events that are not yet explored. Had the same equation been *A = C AND B*, and *B* is more likely to be 0, it would be better if we evaluated *B* first which might shortcut *C*'s evaluation. Thus, the order in which children are evaluated is the key to enable more shortcuts and hence improve sensor energy savings. BPA permutes the branches of the ERT affecting the order of sensor sampling and sub-event evaluation to enhance the chances of shortcuts happening.

Application-Aware Adaptive Sampling Algorithm (AAAS) [19] – saving sensor energy: Atomic events are the events which are directly associated with the sensors and immediately evaluated from the sensor readings (the most primitive application fragment). These events are at the leaves of any ERT. Atomic events could be cached at the beneath layer to save more sensor energy. For every atomic event cached at the beneath, AAAS algorithm uses ARMA (Auto-Regressive Moving Average) model [18] to predict sensor data and skip subsequent samplings, if the predicted value is close to the sampled value. However, there should be a limit on the number of samplings skipped and AAAS uses a modified version of the algorithm proposed in [4], bringing in some characteristics from the cloud applications to fix the maximum skip limit and better optimize sensor energy.

Edge-to-Beneath Application Fragment Caching Algorithm (AFCA-2) [19] – saving sensor energy: AFCA-2 selects the atomic events to cache at the beneath layer to achieve more optimized energy efficiency of the sensor nodes. Atomic events when cached at the beneath layer, would miss out on the energy savings happening because of shortcuts during the evaluation of ERTs at the edge. AFCA-2 calculates the overall benefit that could be obtained by caching an atomic event to the beneath but potentially missing out on savings because of shortcuts. If the calculated benefit is greater than a certain threshold, AFCA-2 caches the atomic event at the beneath and does not cache it otherwise.

4 RELATED WORKS

Several research projects focused on minimizing network latency in cloud based systems. One of the most significant work is the cloudlet approach by Satyanarayanan et al [16], in which the cloud is brought closer to the applications/mobile devices (one hop away) by introducing a new architectural element called cloudlet in the three-tier hierarchy: mobile device – cloudlet – cloud. In contrast to the cloudlet approach, in our cloud-sensor systems approach, the applications are on the cloud and the edge is used to bring the physical world (sensors and devices) closer to the cloud. In CEB, edge is used to either bring the physical world closer to the cloud or cache application fragments down closer to the physical world.

A power and latency aware optimum cloudlet selection strategy was proposed by Mukherjee et al [12] for multi-cloudlet environment. A computation model combining the characteristics of fog computing [3] [17] and in-cooperating Complex Event Processing (CEP) [14] at the edge of the network, to achieve low latency and real-time responses that cloud applications demand, was proposed by Madumal et al [9]. A mathematical model of fog computing which assesses the applicability of fog computing in Internet of Things to meet the demands of latency-sensitive applications running at network-edge, was proposed by Sarkar et al [15]. It also showed that as the number of latency sensitive applications increase, fog computing outperforms cloud computing. A service oriented network architecture named Application Assist Network (AAN), with an adaptive network cache algorithm achieving lower response times than the traditional caching algorithms was proposed by Matoba et al [10]. To effectively manage data latency, [8] provided a framework for a responsive data architecture (RDA), based on patterns seen in real IoT projects that leverage the public cloud.

AirBox – a performant and scalable edge service platform that can execute functionality onloaded on behalf of remote, cloud-based services, in order to address the bandwidth use and latency requirements of device-cloud interactions was proposed by Bhardwaj et al [2]. Unlike AirBox, where edge can directly handle application requests, in CEB all the application requests are addressed by the cloud. Zhang et al [23] introduced Mobile Edge Computing (MEC) into 5G architecture and evaluated the network end-to-end latency. Low-latency services with a requirement not smaller than 17ms are supported by MEC. Drolia et al [6] discussed about leveraging spatio-temporal context at the edge to dynamically create caches in edge servers and across mobile devices to decrease latency for vision-based applications. In this article, we model latency inherent in evaluating events in an event-driven application model and propose a greedy optimization algorithm within CEB to minimize latency.

5 LATENCY FORMULATION AND OPTIMIZATION

The time elapsed from the moment an application places an event evaluation request to the cloud application runtime (CAR) until the moment it gets back a response from the CAR, is defined as the latency involved with the request. As we mentioned before, the current implementation of CEB with bi-directional waterfall optimization framework has not considered latency optimization (minimization) as one of its key challenges. This section formulates and analyzes the latency under CEB as defined above and explains the potential reasons for high latency in the existing implementation of CEB. The section also introduces the two ways to think about latency optimization, and the type of latency optimization problem that this work aims to address.

Since selective push is employed when an event is cached at the edge, the event's value is pushed into the cloud from edge, only when it changes from its previous value. This means that any point in time, the cloud layer has the most updated value of the cached event. Thus, when an application requests the value of a cached event, it can directly be given the value from the shadow event service in the ACM. No request needs to be passed to the edge layer or the beneath and no sampling needs to be done to service the request.

When an atomic event is not cached by AFCA-1 at the edge, it is not cached at the beneath as well. This is because, AFCA-2 chooses atomic events to cache at the beneath, only from the set of events that are already present in the edge. Every event evaluation request of a non-cached atomic event at the cloud layer, results in a sampling of the sensor corresponding to that atomic event.

5.1 Formulation

Let $T_{ec}(E)$ denote the time taken to evaluate an event E already cached in the edge, whose cloud event service is denoted as $ES(E)$.

$$T_{ec}(E) = T_{CAR}^{ACM} + cpu\_t(ES(E)_{shadow}) + T_{ACM}^{CAR} \quad (1)$$

Here, $cpu\_t(ES(E)_{shadow})$ represents the time taken to fetch the event's value from the shadow event service corresponding to the cloud event service $ES(E)$. T_x^y denotes the time taken to send a request or response from $'x'$ to 'y' (communication time).

The time taken to evaluate an atomic event not cached at the edge, denoted by $T_{ea}(E)$, is given by the following equation:

$$T_{ea}(E) = (T_{CAR}^{ACM} + T_{ACM}^{Edge} + T_{Edge}^{Beneath}) \times 2 + T_{sample}(E) \quad (2)$$

Here, $T_{sample}(E)$ denotes the time taken to sample the sensor corresponding to the atomic event E. The time taken to evaluate any non-atomic event is given by the maximum time taken to evaluate each of the children of that event, assuming parallel evaluations of child events. Let $SC(E)$ denote the set of all child events of event E. The time taken to evaluate an event E in general, denoted by $T_e(E)$, is given by the following recurrence:

$$T_e(E) = \begin{cases} T_{ec}(E), & if\ E\ is\ cached \\ T_{ea}(E), & if\ E\ is\ atomic, not\ cached \\ cpu\_t(ES(E)) + \max\big(T_e(E_i \in SC(E))\big), & otherwise \end{cases} \quad (3)$$

The overall latency for any event evaluation request is formulated as follows:

$$T(E) = T_{App}^{CAR} + T_e(E) + T_{CAR}^{App} \quad (4)$$

Here, T_{App}^{CAR} is the time taken for the evaluation request to reach CAR from a cloud application and T_{CAR}^{App} is the time taken to send back the response from CAR to the cloud application after event evaluation which takes $T_e(E)$ time.

5.2 Analysis of Latency

In order to minimize latency, $T_e(E)$ has to be minimized. Observed latency is least when E is cached at the edge. In this case, the most recently updated value of the event from the cloud shadow event service is directly given to the application. Also, it is the maximum when no atomic event in the ERT of E is cached at the edge by AFCA-1, and the evaluation of each of them results in a sampling of the corresponding sensors. It could be seen that the latency directly depends on whether AFCA-1 caches an event in the edge or not. Hence, it must be made sure that AFCA-1 caches events that are of great interest to the application at the moment. Caching an event in the edge also gives the event a chance to be a part of all the optimizations at the edge and the beneath, which might also improve the energy savings of the sensors corresponding to the cached event.

5.2.1 Potential Reason for High Latency

AFCA-1 right now cares only about the cloud dimension with a constraint on the edge resources and does not take into account the interest of cloud applications on a particular event and the expectations of the applications on how fast event evaluations need to be done. When events are considered for caching at the edge by AFCA-1 without any prioritization (based on applications' interests and expectations) among them, there's a possibility that an event which is of more interest to the application or whose evaluation should not take more time, not getting cached at the edge, as the edge resources might already be exhausted with other cached application fragments, each of which might not be of use to any application at that moment. Similar to how the order in which the branches are evaluated in an ERT is important in shortcut evaluation, the order in which events are considered by AFCA-1 for caching at the edge is important, so that the lower layers are totally application aware. Also, without prioritization among events in AFCA-1, AAAS is only partially application aware because it is based on the range of values of a particular sensor on which cloud applications work. However, AAAS is not aware if some application is actually interested in that particular sensor's value in the first place, at that point in time. Hence, AFCA-1 needs to include a prioritization model for events, and try to cache events to the edge in the order given by the model.

5.3 Optimization Problems

Latency optimization can be looked in two ways. First, there could be a single constraint on the system saying any event evaluation request made by any cloud application should not suffer a latency more than a certain threshold, say K time units. The optimization problem would be to minimize the threshold K. Second, every event could be associated with a real-time constraint on the latency, which implies the expectation of applications on how fast the event's evaluation is to be done and the optimization problem would be to satisfy the maximum possible number of such constraints. This work aims to address the second type of latency optimization problem.

6 EVENT PRIORITIZATION MODEL

As suggested in the previous section, an event prioritization model needs to be included in AFCA-1 to optimize latency in the system. This section discusses in detail one such model we propose. The three key parameters involved in the model are (see Listing 1):

Listing 1: Three key parameters of event prioritization model

(1) Event Evaluation Rate, $R_{er}(E)$
(2) Latency Threshold, $L_{thres}(E)$
(3) Event Value Change Rate, $R_{vc}(E)$

$R_{er}(E)$ implies how fast evaluation requests for an event E comes from the cloud applications. This is a direct measure of how excited the applications are about $E. L_{thres}(E)$ is assigned by the applications to each event. This parameter is a measure of the expectation of applications on how fast the evaluation of E needs to be completed. $R_{vc}(E)$ measures how fast the value of an event changes and how often selective pushes to the cloud could happen if E is cached at the edge.

Based on $L_{thres}(E)$, the set of all events can be split into two broad categories namely critical and non-critical events. If $T(E)$, as defined by equation *(4)*, of an event E is greater than $L_{thres}(E)$ at any point in time, the latency constraint on the event can be satisfied only by caching E, or some or all of the sub-events of E to the edge, thereby reducing the latency. Such events are called critical events. Events for which $L_{thres}(E)$ is greater than $T(E)$ already satisfy the latency constraint. Such events are called non-critical events. Also, based on $R_{er}(E)$, events could be broadly categorized into "Hot events" and "Cold events" depending on if $R_{er}(E)$ is greater or lesser than a constant R_{er}^{thres}, respectively. R_{er}^{thres} is a common constant across any event belonging to any cloud application. Based on the combination of both these factors, events are classified into the following four categories (see Listing 2).

Listing 2: Four event categories

```
(1) Critical and Hot
(2) Critical and Cold
(3) Hot and Non-critical
(4) Cold and Non-critical
```

Caching category 1 and 2 events to the edge would result in achieving the latency optimization goal and caching category 3 events would result in a better user experience.

6.1 Application Fragments Redundancy Problem

Every cloud application is a combination of events. There is a possibility that multiple applications work on some common subset of events. However, each application has its own copy of every event it works on and this leads to redundant events in the CAR. This can be visualized as two ERTs from two different cloud applications, each having its own copy of a common subtree. In the current CEB implementation, this problem is not addressed and there is a possibility that two exactly same events from different ERTs of different applications redundantly getting cached at the edge, which is not an efficient way of utilizing edge resources. Hence, there needs to be a single unique view of every event in the cloud which is what AFCA-1 should be looking at.

Redundant application fragment/event is also problematic because the notion of criticality/latency threshold of the same redundant event might differ for different applications. Thus, for the same event E, one application might assign a latency threshold of say 10ms and another application might assign a latency threshold of 1s. Once there is a single unique view of every event, it becomes easy to define latency constraints on the events. For instance, the minimum of all the latency thresholds assigned for an event by all the cloud applications accessing it, would be the latency threshold of that event. Similarly, the event evaluation request rate of any event would be the sum of all the event evaluation request rates on that event by all the applications accessing it.

6.2 Energy Latency Trade-off

Caching an event to the edge reduces the latency involved in its evaluation. This could also potentially save sensor energy because of the lower layer optimizations acting on the cached event. However, there is a scenario where caching an event to the edge might deplete sensor energy faster than the scenario when the event is not cached at the edge. If $R_{vc}(E)$ is much higher than $R_{er}(E)$, and if E is cached at the edge, the rate at which sampling of sensors associated with E is done and the rate at which selective pushes happen to the cloud from the edge might be much higher than $R_{er}(E)$. However, had E not been cached at the edge, the rate of sampling of sensors associated with E would be equal to $R_{er}(E)$, which might be much lesser than the rate of sampling (and hence sensor energy depletion) in the former scenario. More formally, let $C^1_{egy}(E)$, as given by *(5),* denote the overall energy spent per unit time in evaluating E, when E is cached at the edge by AFCA-1.

In Equation (5), α_4 is the energy spent in communication (selective push) from edge to cloud, $P_{ns}(e_a)$ is the probability of shortcut not happening at event e_a, $\beta(e_a)$ is the actual energy spent in the sampling, done for evaluating atomic event e_a, and α_2 is the energy spent in communication of sampled value from beneath to the edge.

$$C^1_{egy}(E) = (R_{vc}(E) \times \alpha_4 + \sum_{\substack{\forall\, atomic\ event \\ e_a \in E}} P_{ns}(e_a) \times R_{vc}(e_a) \times (\beta(e_a) + \alpha_2) \qquad (5)$$

If E is not cached at the edge by AFCA-1, the cost is given by

$$C^2_{egy}(E) = R_{er}(E) \times \sum_{\substack{\forall\, atomic\ event \\ e_a \in E}} \beta(e_a) + (\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) \qquad (6)$$

Here, α_1 is the energy spent in communication from edge to beneath and α_3 is the energy spent in communication from cloud to edge.

The benefit of caching E to edge is given by

$$B_{ec}(E) = C^2_{egy}(E) - C^1_{egy}(E) \qquad (7)$$

If $B_{ec}(E)$ is greater than 0, then caching E to the edge saves sensor energy and if $B_{ec}(E)$ is less than 0, not caching E to the edge would result in better sensor energy saving. Thus, we could see that sensor energy could be saved by not caching certain events to the edge. However, not caching E to edge increases the latency involved in the evaluation of E. Thus, there exists a clear trade-off between energy savings and latency. This trade-off should also be taken into account when caching events from the cloud to edge.

6.3 Model

The key factors to consider while prioritizing events are their criticality (critical or non-critical), applications' interest (Hot or Cold), the scalability benefit obtained on caching them to the edge, given by $Benefit(E)$ [21], and the energy benefit obtained by caching them to the edge, given by $B_{ec}(E)$. The steps involved in the prioritization are as follows.

1. Start with an empty event list EL.
2. Add all the critical events (category 1 and 2) sorted in decreasing order of $Benefit(E)$ to EL.
3. Add all the Hot and Non-critical events (category 3) with $B_{ec}(E) > 0$, sorted in decreasing order of $Benefit(E)$ to EL.
4. Add all the remaining events (category 4) with $B_{ec}(E) > 0$, sorted in decreasing order of $Benefit(E)$ to EL.
5. Consider events for caching to the edge in the order given by EL.

Category 2 events are given more priority than category 3 events by this model to maximize the number of events whose latency constraints are satisfied, which is the goal of our optimization. The improved AFCA-1, which is called PAFCA to better optimize latency, is given below.

Prioritized Application Fragment Caching Algorithm

```
1.  Initialize $EL, CL, RL, TL$ to ϕ
2.  for every event $E$ in CAR
3.    if $L_{thres}(E) - T(E) \geq 0$
4.      add $E$ to $CL$
5.  end for
6.  sort events of $CL$ in decreasing order of $Benefit(E)$
7.  add events of $CL$ to $EL$ in the sorted order
8.  for every event $E$ not in $EL$
9.    if $R_{er}(E) > R_{er}^{thres}$ and $B_{ec}(E) > 0$
10.     add $E$ to $RL$
11. end for
12. sort events of $RL$ in decreasing order of $Benefit(E)$
13. add events of $RL$ to $EL$ in the sorted order
14. for every event $E$ not in $EL$
15.   if $B_{ec}(E) > 0$
16.     add $E$ to $TL$
17. end for
18. sort events of $TL$ in decreasing order of $Benefit(E)$
19. add events of $TL$ to $EL$ in the sorted order
20. for every event $E$ in $EL$, construct its ERT $T$
21.   Partition $T$ into areas (based on edges)
22.   for each area $A$ in $T$
23.     $selectedEvents = EventSelectV2(A, E)$
24.     send $selectedEvents$ to edge, wait for the response;
25.     receive the events approved to cache from edge;
26.   end for
27. end for
```

The $EventSelectV2()$ method at line 23 of the algorithm is an extension of $EventSelect()$ [21] method of AFCA-1. $EventSelect()$ selects events to obtain maximal total scalability $Benefit$ [21] of selected events and also makes sure that at most one event from each branch of the ERT is chosen. $EventSelectV2()$ filters out some of the events from those selected by $EventSelect()$. For a critical event E_c, only those events in E_c's subtree that have an evaluation time greater than $L_{thres}(E_c)$ need to be cached in order to satisfy the latency constraint of E_c. On the other hand, for a non-critical event E_{nc}, any event in E_{nc}'s subtree with $B_{ec} < 0$ should not be cached as that would result in greater sensor energy loss. The algorithm implementing the above-mentioned steps is given below.

EventSelectV2(Area *A*, Event *E*) Algorithm

```
$initialSelect = EventSelect(A)$
if $E$ is critical
  for every event $E_i$ in $initialSelect$
    if $L_{thres}(E) - T(E_i) \leq 0$
      add $E_i$ to $finalSelect$
  end for
else
  for every event $E_i$ in $initialSelect$
    if $B_{ec}(E_i) > 0$
      add $E_i$ to $finalSelect$
  end for
return $finalSelect$
```

7 Experimental Evaluation

The goals of our experiments were to validate how PAFCA proves to be an efficient solution to the stated latency optimization problem, and to show how reactive and dynamic the algorithm is, when events are accessed at varying rates by the cloud applications. Real-timeliness of responses is very important for critical and real-time tasks, and the responsiveness of any IoT application is the key for improving user experience and possibly user satisfaction. Extending the algorithm to optimize latency, we also made sure that we did not lose much on the scalability benefit obtained with AFCA-1 on place, at least for a practical proportion of critical events in the event set. The trade-off between improving cloud scalability (major goal of AFCA-1) and latency optimization (major goal of PAFCA) is also presented in this section.

Our experiments were based on a simulated set of about 99,400 events and 100 edges, with their properties set to match the real-world events and edge computers well, respectively. AFCA-1 and PAFCA were simultaneously executed on the simulated event set and several metrics like the number of critical events cached, latency in evaluating "Hot Events", the overall scalability benefit obtained by both algorithms were extracted and compared, after every execution cycle of both algorithms, to validate the effectiveness of PAFCA.

The simulated event set consisted of atomic events (no child) and events with 1 to 4 children, with their ERTs having up to 15 levels. Experiments were done on multiple such sets, each having a different number of atomic events, and different number of composite events at multiple levels. For any event E cached at the edge, the time taken to retrieve its value from its corresponding shadow service at ACM was modeled with a discrete probability function. The probability of the memory region being unavailable during retrieval of a cached event's value was set to be 0.05. The other case was the memory region being available during retrieval and the probability of that was set as 0.95. The evaluation time for atomic events, $T_{ea}(E)$, as given by Equation (2) was modeled as a normal distribution around a mean (300ms) with a standard deviation of 20ms.

The time taken to evaluate composite events was calculated using the recursive formulation given by Equation (3). The expected evaluation times of events $L_{thres}(E)$, that are fixed by the cloud applications were modeled as a uniform distribution within a fixed range of time values. The number of CPU cycles required to evaluate an atomic event E was also modeled using a normal distribution. Also, the number of CPU cycles required to evaluate a composite event E was calculated as the sum of the number of CPU cycles required to evaluate all of E's children recursively because of our assumption that evaluation of child events happen in parallel. In order to model the event value change rate $R_{vc}(E)$, we used the PLCouple1 dataset collected from PlaceLab [11] to learn how frequently the sensor values change and used that information to model $R_{vc}(E)$ for all the atomic events. For every composite event E, $R_{vc}(E)$ was calculated as the maximum event value change rate of all the children of E.

With the events simulated and their properties modeled as mentioned above, we ran an instance of AFCA-1, an instance of PAFCA and an instance of an event evaluation request simulator on three different threads. The request simulator was designed to pick events at random and simulate event evaluation requests from cloud applications, thereby changing $R_{er}(E)$, for every picked event. As $R_{er}(E)$ becomes high for certain events, we could see that PAFCA detects this and tries caching those events to the edge to improve the responsiveness of the application.

7.1 Experiment 1

The goal of this experiment was to compare the amount of critical events whose latency constraints as set by the cloud applications, were satisfied. $L_{thres}(E)$ for every event E, was modeled as a uniform distribution with a *lower limit* close to the minimum evaluation time of all atomic events. The *upper limit* of the distribution was assigned a much higher value than the maximum evaluation time of all events, and was decreased at regular intervals, until it became equal to the lower limit. For each {*lower limit, upper limit}* pair, $L_{thres}(E)$ was generated for all the events and AFCA-1 and PAFCA were simultaneously executed. As the *upper limit* of the distribution decreased, the number of critical events increased and PAFCA prioritized the critical events while caching events to edge. This resulted in a decrease in the evaluation times of the edge cached events such that their latency constraints were satisfied.

The result of performing this experiment with edge computers being Dell Latitude E6520 is shown in Figure 2. We could see that the amount of satisfied critical events with PAFCA is always greater than that with AFCA-1. When the number of critical events was 10% of the total events, PAFCA satisfied about 4.5% more critical events than AFCA-1, and when the amount was 20% PAFCA satisfied about 9% more critical events than AFCA-1. Figure 3 shows the results of the same experiment but done with a different edge computer (Raspberry Pi). The memory capacity of the edge was set to be 256MB and the processor speed was 700MHz. As the edge resources are very limited, the number of satisfied critical events is significantly lower.

Figure 4 shows the results of the experiment done with the *upper limit* of the distribution mentioned above, set to a value lower than the minimum of $T_{ea}(E)$ of all the atomic events. The *lower limit* was set to 0. All the events in the event set were critical with this setting. As we could see, when all the events were set with $L_{thres}(E) = 0$, no latency constraint could be satisfied by both AFCA-1 and PAFCA as it is impossible to achieve a response time of 0ms even with caching. As the *upper limit* on $L_{thres}(E)$ increases, PAFCA could satisfy up to 97.37% of critical events while AFCA-1 satisfied up to 52.03% of critical events.

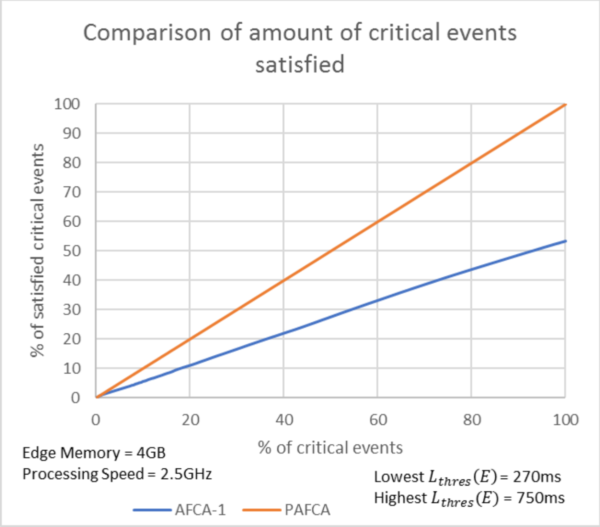

Figure 2: Comparison of number of critical events satisfied with Dell Latitude E6520 as Edge node

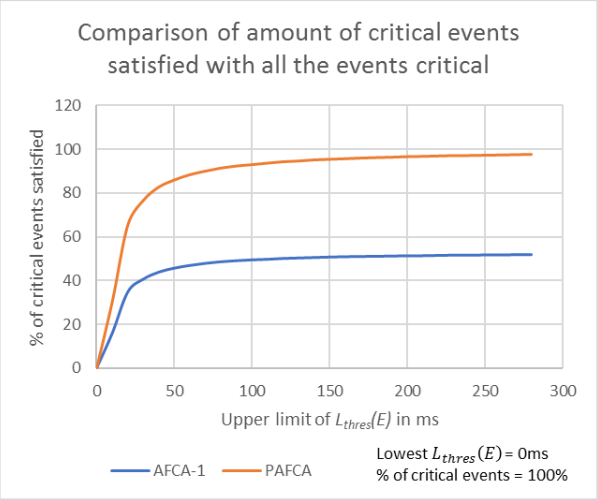

Figure 4: Comparison of amount of critical events satisfied with all events critical

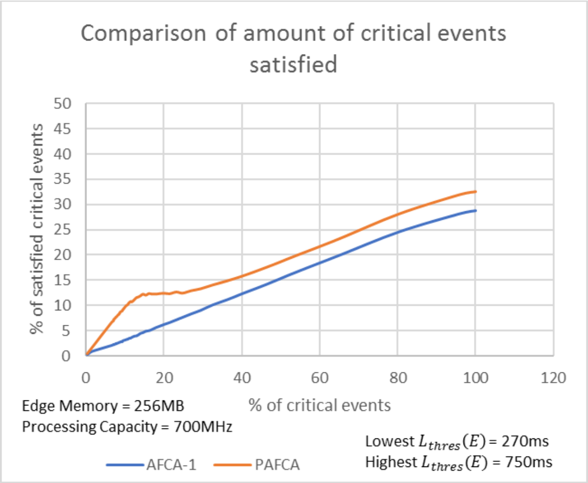

Figure 3: Comparison of amount of critical events satisfied with Raspberry Pi as Edge node

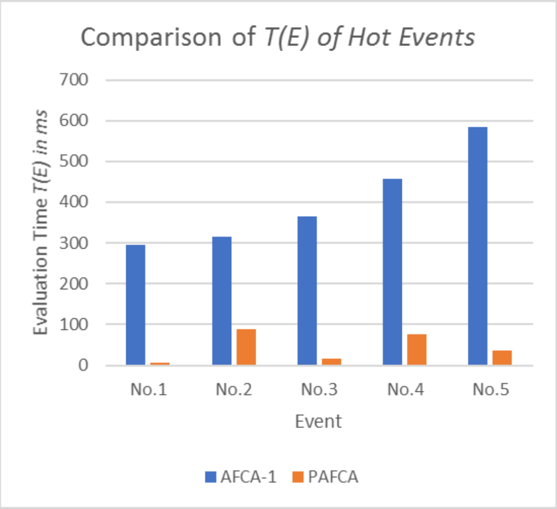

Figure 5: Comparison of $T(E)$ of Hot Events which is key to the responsiveness of cloud applications

7.2 Experiment 2

The goal of this experiment was to compare the evaluation times of “hot events” under AFCA-1 and PAFCA and understand how PAFCA dynamically caches events that are of most interest to cloud applications at the moment, in an attempt to improve the responsiveness of the cloud applications. The event evaluation request simulator was implemented in such a way to figure out events that are not cached by both AFCA-1 and PAFCA and simulate application requests on those events, which would increase $R_{er}(E)$ of those events. When $R_{er}(E) > R_{er}^{thres}$, we could see that PAFCA would cache it while AFCA-1 would be unaware of that. Figure 5 shows the results of the experiment. The lowest possible latency in our experiment was 7ms, which is the time needed to fetch an event’s value from its corresponding shadow service at the cloud and give it to the cloud application.

7.3 Experiment 3

The goal of this experiment was to compare the scalability benefit obtained by AFCA-1 and PAFCA in caching events to edge. The setup for this experiment was the same as the one for the Experiment 1. The results of the experiment are shown in Figure 6.

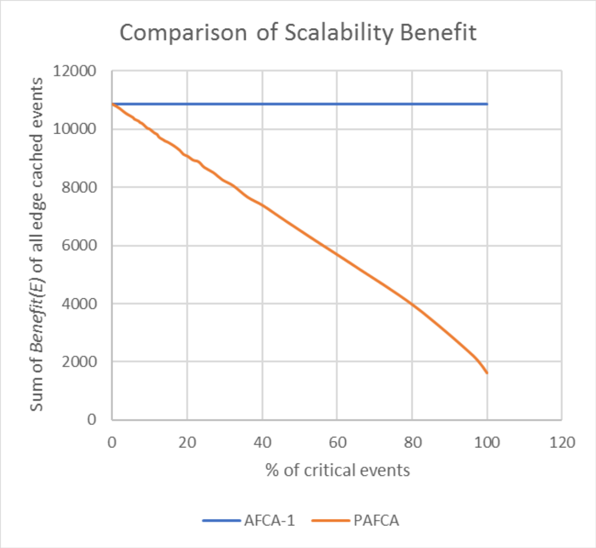

Figure 6: Comparison of scalability benefit with Dell Latitude E6520 as edge node

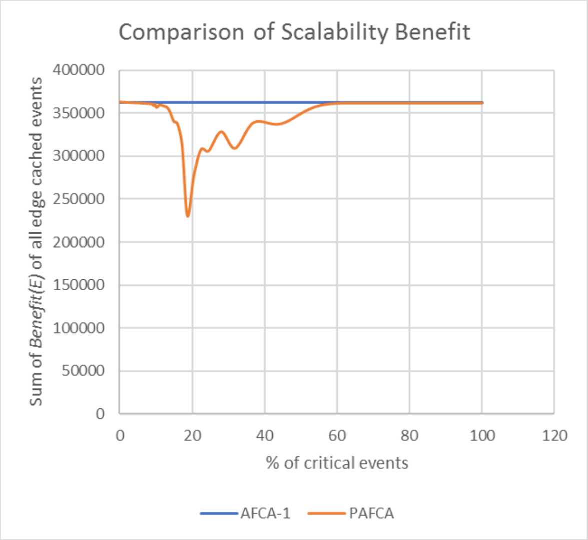

Figure 7: Comparison of scalability benefit with Raspberry Pi as edge node

It could be seen that regardless of the number of critical events, the sum of scalability benefit $Benefit(E)$ of all events cached at edge servers is the same for AFCA-1 and is the optimum. However, because PAFCA follows a greedy strategy with event prioritization, critical events that have negative scalability benefit also gets cached at the edge decreasing the overall sum. We could see that if the percentage of critical events is up to 5% of the total events (about 5000 events in our case), the scalability benefit obtained by PAFCA is up to 3.5% less than that obtained by AFCA-1. If the percentage of critical events is 10%, scalability benefit given by PAFCA is about 8% less than that obtained with AFCA-1. When the number of critical events increases to 20%, the scalability benefit decreases by 16%.

The reason for this drop is the greedy strategy of PAFCA which tries to satisfy critical events first even if that would decrease the overall scalability benefit. However, from a practical standpoint, assuming that the number of critical events would be less than 20% of the total events looks fair. Figure 7 shows the results of the same experiment with the edge computer being Raspberry Pi with the memory capacity of 256MB and processing speed of 700MHz. It could be seen that as the edge computers become saturated and has no more space or processing power to allow further caching, critical events with negative scalability benefit do not get cached at the edge and the sum of scalability benefit given by PAFCA becomes as good as AFCA-1 when the number of critical events is more than 50% of the total number of events.

8 Conclusion And Future Work

Latency is a major challenge for any cloud-sensor system or Internet of Things application. Our work gave a formal definition and formulation of latency inherent in event evaluation for any event-driven application model, and proposed an extension of the AFCA-1 optimization algorithm of our CEB architecture to include latency in its optimization goals and equations. The results of our simulation experiments prove the effectiveness of PAFCA in handling Critical and Hot events (as defined in this paper). Our ongoing and future work is focused on improving the scalability benefit obtained with PAFCA, even with a high number of critical events. We are also designing mobile sensor and device support and optimization in CEB, in which a device may change the edge it belongs to dynamically.

References

[1] ARM mbed, “Stack Architecture,” https://docs.mbed.com/docs/arm-ipv66lowpan-stack/en/latest/02_N_arch/, accessed: 16th November 2016.

[2] K. Bhardwaj, M. W. Shih, P. Agarwal, A. Gavrilovska, T. Kim and K. Schwan, "Fast, Scalable and Secure Onloading of Edge Functions Using AirBox", in Proceedings of *IEEE/ACM Symposium on Edge Computing (SEC)*, Washington, DC, pp. 14-27, 2016.

[3] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, "Fog Computing and Its Role in the Internet of Things", in Proceedings of *first Ed. MCC Work. Mob. cloud Comput.*, pp. 13-16, 2012.

[4] S. Chatterjea and P. Havinga, "An Adaptive and Autonomous Sensor Sampling Frequency Control Scheme for Energy-Efficient Data Acquisition in Wireless Sensor Networks," in Proceedings of *DCOSS'08*, Santorini, Greece, pp. 60-78, Jun. 2008.

[5] S. de Deugd, R. Carroll, K. Kelly, B. Millett and J. Ricker, "SODA: Service Oriented Device Architecture," *IEEE Pervasive Computing*, vol. 5, no. 3, pp. 94-96, 2006.

[6] U. Drolia, K. Guo, R. Gandhi and P. Narasimhan, "Poster Abstract: Edge-Caches for Vision Applications," in Proceedings of *IEEE/ACM Symposium on Edge Computing (SEC)*, Washington, DC, pp. 91-92, 2016.

[7] J. King, R. Bose, H. i. Yang, S. Pickles and A. Helal, "Atlas: A Service-Oriented Sensor Platform: Hardware and Middleware to Enable Programmable Pervasive Spaces," in Proceedings *of 31st IEEE Conference on Local Computer Networks*, Tampa, FL, pp. 630-638, 2006.

[8] D. Linthicum, "Responsive Data Architecture for the Internet of Things," *Computer*, vol. 49, no. 10, pp. 72-75, Oct. 2016.

[9] M. B. A. P. Madumal, D. A. S. Atukorale and T. M. H. A. Usoof, "Adaptive Event Tree-Based Hybrid CEP Computational Model for Fog computing Architecture," in Proceedings of *Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, Negombo, pp. 5-12, 2016.

[10] K. Matoba, K. i. Abiru and T. Ishihara, "Service Oriented Network Architecture for Scalable M2M and Sensor Network Services," in Proceedings of *15th International Conference on Intelligence in Next Generation Networks*, Berlin, pp. 35-40, 2011.

[11] MIT, "PlaceLab Datasets," http://web.mit.edu/cron/group/house_n/data/PlaceLab/PlaceLab.htm, accessed: 23 December 2016.

[12] A. Mukherjee, D. De and D. Guha Roy, "A Power and Latency Aware Cloudlet Selection Strategy for Multi-Cloudlet Environment," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp.1-1, July. 2016.

[13] OSGi Alliance, "Open Services Gateway Initiative (OSGi) 4.2 Specification," http://www.osgi.org/Download/Release4V42, accessed: 15 September 2016.

[14] D. Robins, "Complex Event Processing", in Proceedings of *Second International Workshop on Education Technology and Computer Science*, pp. 10, 2010.

[15] S. Sarkar, S. Chatterjee and S. Misra, "Assessment of the Suitability of Fog Computing in the Context of Internet of Things," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1-1, October 2015.

[16] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14-23, 2009.

[17] I. Stojmenovic, "Fog computing: A cloud to the Ground Support for Smart Things and Machine-to-Machine Networks," in Proceedings of *Australasian Telecommunication Networks and Applications Conference (ATNAC)*, Southbank, VIC, pp. 117-122, 2014.

[18] Wikipedia, "ARIMA," http://en.wikipedia.org/wiki/Arima, accessed: 12th October 2016.

[19] Y. Xu "Architecture and Optimization for Cloud-Sensor Systems," Ph.D. Dissertation, University of Florida, 2014.

[20] Y. Xu and S. Helal, "An Optimization Framework for Cloud-Sensor Systems," in Proceedings of *IEEE 6th International Conference on Cloud Computing Technology and Science*, Singapore, pp. 38-45, 2014.

[21] Y. Xu and A. Helal, "Application Caching for Cloud-Sensor Systems," in Proceedings of *ACM MSWIM'14*, Montreal, Canada, pp. 303-306, 2014.

[22] Y. Xu and A. Helal, "Scalable Cloud–Sensor Architecture for the Internet of Things," *IEEE Internet of Things Journal*, vol. 3, no. 3, pp. 285-298, June 2016.

[23] J. Zhang, W. Xie, F. Yang and Q. Bi, "Mobile Edge Computing and Field Trial Results for 5G Low Latency Scenario," *China Communications*, vol. 13, no. Supplement 2, pp. 174-182, 2016.

AUTHOR BIOGRAPHIES

Adhithya Balasubramanian is a Masters Student in the department of Computer Science at University of Florida, Gainesville, FL, USA, where he works as a Research Assistant at the Mobile and Pervasive Computing Lab. His research interests span cloud-sensor systems, internet of things, mobile computing, distributed systems and energy-aware computing.

Sumi Helal (Fellow 15) received the Ph.D. degree in computer sciences from Purdue University, West Lafayette, IN, USA. He is currently the chair professor in digital health at Lancaster University, UK. Before joining LU, he was a Professor of Computer Science and Engineering at University of Florida, USA, and the Director of its Mobile and Pervasive Computing Laboratory. His research interests span pervasive and mobile computing, internet of things, smart spaces, and smart health and well-being.

Yi Xu received the Ph.D. degree in computer science from University of Florida, Gainesville, FL, USA, where he worked at Mobile and Pervasive Computing Laboratory. He is currently working for Google, Mountain View. His research interests span pervasive & mobile computing, programming models and middleware for cloud-sensor systems and the internet of things.