Open Access

Open Journal of Internet Of Things (OJIOT)
Volume 5, Issue 1, 2019

www.ronpub.com/ojiot
ISSN 2364-7108

Energy Savings in Very Large Cloud-IoT Systems

Yi Xu [A], Sumi Helal [B], Choonhwa Lee [C], Ahmed Khaled [D]

[A] Google, 1600 Amphitheater Parkway, Mountain View, CA 94043 yix@google.com
[B] School of Computing and Comm., Lancaster University, Lancaster LA1 4WA, U.K., s.helal@lancaster.ac.uk
[C] Dept. of Computer Science and Engineering, Hanyang University, Seoul 04763, Korea, lee@hanyang.ac.kr
[D] Computer Science Dept., Northeastern Illinois University, Chicago, IL 60625, USA, aekhaled@neiu.edu

ABSTRACT

Opposite to the original cloudlet approach in which an edge is utilized to bring the cloud and its benefits closer to the applications, in cloud- and edge-connected IoT systems where the applications are deployed and run in the cloud, we exploit the edge somewhat differently, either by bringing the physical world and its data up closer to the cloud or by caching parts of the applications down closer to the physical world. Aggressive optimizations seeking substantial IoT energy savings are needed to maintain the scalability of large-scale IoT deployments and to stay within cloud cost constraints (avoiding costly elasticity when working with a budget limit). In this paper, we present a novel optimization approach that relies on the simple principle of minimizing all movements: movements of data from the IoT up to the Edge and Cloud, and movements of application fragments from the cloud down to the edge and the IoT itself. Our approach is novel in that it involves and utilizes the dynamic characteristics and variability of both the data and applications simultaneously. Another novelty of our approach is the definition and use of "sentience-efficiency" as a precursor to "energy-efficiency" for achieving truly aggressive savings in energy. We present our bi-directional optimization approach and its implementation in terms of algorithms within an architecture we name the cloud-edge-beneath architecture (CEB). We present a performance evaluation study to measure the impact of our optimization approach on energy saving.

TYPE OF PAPER AND KEYWORDS

Research Paper: *Internet of Things, cloud-IoT architecture, edge architecture optimizations, pervasive computing, cloud computing, scalability, performance.*

1 INTRODUCTION

As IoT proliferates into a massive scale, data and related services (applications) will be pressed to move to the cloud given its economies of scale and highly anticipated reductions in services costs. Another key advantage of the cloud is its ability to facilitate multi-stakeholder access to the IoT applications, especially in smart city scenarios. The cloud central involvement in large scale IoT deployments will therefore emerge as an IoT architecture in which the physical sensors and devices must remain external to the cloud and cannot be farmed or provided dynamically as cloud resources.

Considering the anticipated growth of IoT in terms of devices, many driven by smart city deployments

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2019)* in conjunction with the VLDB 2019 conference in Los Angeles, USA. The proceedings of VLIoT@VLDB 2019 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

including smart parking, smart meters, etc. (over 20B by 2020 according to Gartner), cloud-IoT systems must be carefully architected to scale up to such massive scale in devices and the applications that would utilize them. The extensive interactions between the cloud (IoT applications and services) and the physical sensors and devices will pose significant challenges to the scalability and energy demand of any cloud-IoT system.

Cloud Scalability: Extensive external interactions between cloud services and the physical sensors could pose significant challenges to the scalability of the overall system. The excessive interactions could result in expensive cloud "attention", not only per device such as a sensor, but per each sensor duty cycle. For instance, if sensors push data once every minute, then millions of sensors will produce billions of sensor-cloud interactions, daily; and billion sensors will produce trillion interactions. This will require tremendous processing power, memory resources and huge incoming/outgoing cloud traffic, leading to heavy and constant draw on cloud elasticity. As a result, the cloud economies of scale per sensor will not stand, rendering the cloud too expensive to pay for, given the existing use-based price models.

Energy Constraint of IoT Devices: Unlike elastic cloud resources which can be provisioned on demand, devices and sensor cannot be provided dynamically. Many of these sensors and devices are battery-powered which makes them vulnerable to power drainage. In smart city scenarios, a sensor may be queried by hundreds of applications each of which requires constant evaluation of events based on the sensor readings. This could lead to continuous data sampling by the sensor nodes and transmission through the sensor network which incurs substantial energy cost to the sensor hardware as well as the entire sensor network. Without optimization, sensors' energy could be depleted rapidly, failing services and making them unreliable and unavailable.

Therefore, a structural basis for optimizing the cloud's interactions with IoT sensors and devices is critically needed or cloud-IoT systems will not be dependable. To achieve this goal, both the supply of data from the IoT devices and the demand on this data from cloud applications will need to be carefully optimized. In [30], we proposed the cloud-edge-beneath (CEB) architecture to enable the efficient operation of such cloud-IoT systems. An event-driven application model was also proposed within the same framework in [32] to enhance the programmability of cloud-IoT system applications. In [31], we demonstrated the optimization-enabling aspects of CEB and introduced the bi-directional waterfall optimization framework whose goal is minimizing overall system dynamics to maintain acceptable levels of scalability and minimize sensor energy consumption.

In this paper, we build on our prior work on CEB and its optimization framework and present an implementation of the bi-directional waterfall model in terms of detailed algorithms and an experimental evaluation study. Prior work focused on details of the CEB architecture, details of one algorithm – the application fragment caching algorithm (AFCA-1) summarized in this paper in section 4, and on the bi-directional waterfall model. In this paper, we summarize and include our prior work, in addition to presenting the details of three other optimization algorithms in section 5, 6 and 7. The paper is organized as follows. In Section 2, we present important related work and layout the optimization goals and guiding principles for large-scale cloud-IoT systems. In Section 3, we provide a brief summary of CEB and its event-driven application model (details can be found in [30] and [32] but a summary is provided here for readability). We also summarize the bi-directional waterfall optimization framework which is based on the event-driven instance of CEB. Our optimization approach and framework are implemented through several optimization algorithms presented in Sections 4, 5, 6 and 7, which aim to achieve a greater cloud scalability and energy-efficiency of sensor devices. In Section 8, we evaluate the performance of the proposed implementation of the optimization framework utilizing a semi-synthesized city-scale application/data benchmark. Conclusion and future work are presented in Section 9.

2 Related Work and Optimization Principles

2.1 Related Work

Special data acquisition techniques have been developed for event detection supporting real-time wireless sensor network application execution. A typical scheme is polling [35], in which a data sink sequentially polls its underlying sensors for new data. In contrast, a bottom-up sensor-driven model [25] has also been proposed, assuming that sensors are capable of pushing data to applications when an event occurs. To improve the efficiency of data delivery and enable data sharing, messaging paradigms such as publish/subscribe [28] and push-pull [17] have been widely adopted in sensor data acquisition. Optimization techniques to balance push and pull have been extensively discussed in [17][29][11] which focus on network topology and routing algorithms. Furthermore, a new model discussed in [7] utilizes the mixed push/pull strategy and takes advantage of the optimization opportunity provided by the event structure and its data coherency relaxations

(e.g., time-frequency sampling relaxations). However, none of the above approaches fit cloud-IoT systems. This is especially true when one considers the massiveness of sensors and applications that tend to be invisible to each other. In order to overcome this problem, we develop optimization strategies based on the relative characteristics of sensor requests (demand side from the cloud) and sensor data (supply side from beneath). These two features can be easily captured in our model and support our claims of effectiveness and significance in promoting our approach's energy efficiency.

In addition, among the traditional efforts to achieve sensor network efficiency (e.g., energy efficiency), a widely-studied approach is to minimize transmit power subject to some QoS constraints. It was pointed out that the total energy consumption should be understood as transmission energy consumption together with hardware (or circuit) energy consumption [10]. A certain transmit power level is necessary to satisfy certain QoS requirements. For instance, if the data-rate increases, the required transmit power level increases as well. However, at the same time, the transmission time decreases, so that the change in energy spent for transmission mirrors the resulting shift in the trade-off between transmission time and transmission power. Consequently, several efforts have been performed to minimize the energy consumption the optimal power-time tradeoff subject to the given SIR requirements. On the other hand, the use of physical layer symbol error rate (SER) optimization was investigated to minimize wireless sensor network (WSN) energy consumption [12]. The study proposed a technique for SER optimization that balances the energy saving due to rising SER and the corresponding extra amount of energy spent on frame retransmission. Another energy optimization strategy was proposed for wireless sensor networks by which each node is able to select its optimal listening mode according to its local state, which reduces the global network cost [13]. To reduce energy cost in WSNs, a more comprehensive effort [3] focused on the computation of optimal transmission power, routing, and duty-cycle schedule that optimize the WSNs energy-efficiency. In that effort, a feedback algorithm computes the proper transmission power level between nodes; then, a routing protocol can make use of the transmission power as a metric by choosing routes with optimal power consumption to forward packets. Finally, the cross-layer routing information is exploited to form a duty-cycle schedule in the MAC layer.

The optimization approaches discussed thus far share the limitation that the inputs of the optimization equation are solely derived from the metrics and other characteristics of the sensor network and sensor hardware. By bringing more influential inputs to the optimization problem, additional powerful optimization opportunities may be realizable; optimization opportunities were explored by investigating the characteristic of sensor data and by adopting a transmission suppression scheme, both temporal and spatial, to filter and aggregate data transmitted to the data sink in order to reduce energy cost due to radio transmission [27]. Also, a more sophisticated statistical model of real-world processes that maps the raw sensor data onto physical reality was introduced for the sensor query process [8]. This approach presented a model of real-world process, and claimed that sensors should be used to acquire data, only when the statistical model is not sufficiently rich to answer the query with acceptable confidence. The approach enables so called declarative query to achieve high energy efficiency for interacting with networks of wireless sensors. Both optimization schemes [27][8] take data and their models as additional inputs to the optimization equation and do achieve further energy efficiency.

In this paper, we also utilize data models as a crucial additional input for optimization. Furthermore, we exploit an additional opportunity for optimization and improving system efficiency by taking cloud applications as input and part of the optimization choice variables. By simultaneously combing and learning the relative characteristics of both demand (applications) and supply (data), we are able to achieve powerful optimization opportunities and aggressive levels of scalability. To this end, we revisit and extend the traditional caching technology, which has been widely adopted for sensor-based computing [8][23], to improve the energy efficiency and latency of the overall sensor system. However, in any of these approaches, the entities to cache are usually limited to sensor readings. In the meantime, a novel caching scheme was proposed in which operators in query graph that carries the semantics from application layer can be pushed down inside the network to perform "in-network" processing with the intent of reducing data transmission [24]. In our work, to further improve the system scalability and energy efficiency, we extend the traditional caching scheme and propose an optimization framework in which both sensor readings and fragments of applications can be cached at different layers of CEB in opposite directions. Compared to query shipping widely adopted in distributed database systems whose stored data are relatively constant [34], data in cloud-IoT systems are dynamic and constantly changing. Such system dynamism poses a major challenge in deciding the proper application fragments and sensor data to cache in the system to achieve maximal scalability and energy efficiency, while adapting to the dynamic changes.

Additionally, caching application closer to the sensor layer allows the system to learn both the characteristics of sensor data and their consumers (applications) at the same time, which helps optimize the energy consumption of the sensor nodes. Data predictions could be utilized to skip sensor samplings to save sensor energy based on data correlation [33][15]. In this paper, we observe that with both the history of sensor data observed from the sensor layer and the application semantics retrieved from the application layer, a relatively low sampling rate can be achieved and adjusted based on a relaxed requirement of data accuracy (i.e., QoS).

2.2 Optimization Principles and Goals

Before exploring any specific optimization opportunity, we lay down simple principles specific to cloud-IoT systems that will guide our own algorithm designs.

Generally speaking, capturing the dynamics of the monitored environment and reacting to changes are the goal of our model. In order to achieve this goal, sensor devices are sampled periodically by the applications in the cloud to capture the most updated environmental conditions and trigger corresponding actions once a pre-specified event occurs. However, relative characteristics of the sensor data and the relevant applications in the cloud, if learned by the system, can suppress the dynamism of the system in a way that only a subset of the data or data changes are required to be supplied to the application without affecting application behavior. This is similar to the principle of minimum amount of work leading to minimizing total energy consumption in the cloud-IoT systems without affecting the semantics of the applications. More precisely, our suppressed system dynamics approach aims at minimizing the actions (conveyance of data request down by the applications or movement of data up by the sensors) that must be taken in the cloud-IoT system, while at the same time ensuring the adequacy and timeliness of the minimized actions.

Suppressed system dynamics promises greater and unprecedented energy-efficiency by additionally pursuing sentience-efficiency – a utilization of hidden joint semantics of data and applications that offers significant reduction in the work needed to execute IoT applications, and hence, reduces system dynamics and overall energy expenditure. For example, even if a sensor datum changes or if an application explicitly asks for certain data, nothing may need to be done in response in certain conditions, as we will show later in our optimization algorithms. Powering applications with the minimum sentience required is a precursor to doing so energy–efficiently. Hence, in our approach, we pursue energy efficiency in a sentience-efficient system. Any optimization solution that we pursue must follow the suppressed system dynamics principle, and hence must firstly be sentient-efficient, and secondly, energy-efficient. This promises significant improvements in cloud scalability as well as significant savings in the total energy as will be explained later.

In a cloud-IoT system, cloud applications are constantly requesting data, and as sensor data changes, sensors continuously send data up to the cloud. An efficient cloud-IoT system must utilize influential optimization opportunities exploiting the distributed nature of the multi-tiers across the paths of data and application requests. For instance, the system must adaptively match the mix of cyber data demands in the cloud from the various independent applications. It most certainly should exploit caching. It could optimize further, if it better understands the application behavior as well as the sensor data behaviors. To this end, in the cloud-edge-beneath (CEB) architecture, the Cloud layer senses the characteristics of the applications. The Beneath layer senses its own sensor data characteristics. The Edge layer is able to "solve the puzzle" and consolidate and share hints from the Cloud and Beneath.

In the paper, we present detailed optimization ideas and algorithms to maximize sentience and energy efficiency within CEB. We are currently utilizing an eventing model for programming applications in CEB. The programming model could affect (enable or limit) the potential optimization space. We will consider other application models in the future, including publish/subscribe and functional programming models. We consider only application models that allow high degree of freedom in fragmenting and caching app fragments within the cloud-IoT system.

3 Summary of CEB Architecture and Bidirectional Waterfall Optimization Framework

In [30], we proposed the Cloud, Edge, and Beneath (CEB) which is an open architecture and framework for deploying and managing cloud-IoT systems whose applications are programmed, hosted and run on the cloud. The architecture organizes sensor nodes and the cloud along with intermediate edge layer and draws on well-established and extensible standards. Our current implementation is based on a specific application model that abstracts sensor data into events. Based on and limited to this specific application model, we proposed a bi-directional waterfall optimization framework [32].

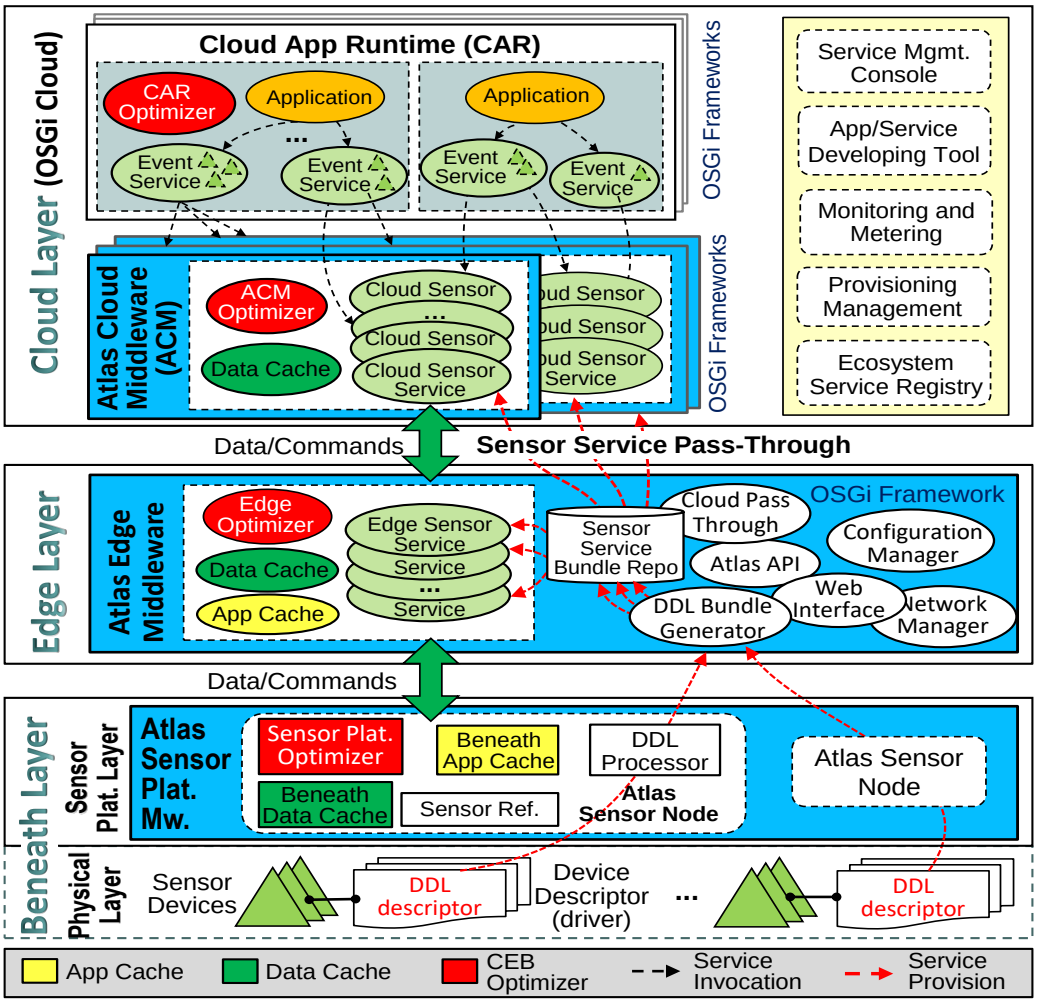


Figure 1: Overview of the CEB architecture

3.1 CEB Architecture Overview

CEB (Fig.1) is a multi-tier architecture which we collectively refer to as the "Cloud-Edge-Beneath" where the beneath refers to the physical sensors and their sensor platforms. Sensor platforms are low-power computing and communication platforms through which physical sensors connect to the edge. In practice, edge as an intermediate layer (e.g., standalone server) connects and manages a group of geo-related sensors. Finally, the cloud is where sensor-based services and applications are developed, deployed and run. This three-tiered structure aims to achieve scalability, since sensor networks operate independently, and are connected to the cloud through a scalable number of power-unconstrained edge servers.

CEB is built on top of Atlas [14] which is an implementation of the service-oriented device architecture (SODA) [9]. Atlas automates the process of sensor integration through Atlas sensor platform and Atlas middleware which are eventually integrated into the cloud availed for use by cloud applications. Next, we explain each layer of CEB concisely.

The beneath layer consists of the physical layer and the sensor platform layer. The former refers to the sensors and their "drivers" – documents written according to the Device Description Language (DDL) [6]. DDL documents contain the information required for automatic (on power-up) device integration, including service registration, discovery and the main operations of the sensor hardware. The sensor platform layer hosts one prong of the Atlas middleware which is responsible for identifying the connected devices, using their DDLs to generate corresponding sensor service(s) on the edge and beyond.

The edge runs the one prong of the Atlas middleware which uses OSGi [20] as its basis to provide service discovery and configuration. The middleware includes a bundle generator, which, when contacted by an

Equation (1):

$$
\begin{aligned}
E &= sesor(value)|sensor[a,b] && \text{(atomic event)}\\
&= |{\sim}E && \text{(negation)}\\
&= |E \vee E|E \wedge E && \text{(or/and)}\\
&= |E?\,E:E && \text{(condition operation)}\\
&= |E * time * E && \text{(sequence)}\\
&= |\{E\} && \text{(scope block)}
\end{aligned}
\qquad \textbf{(1)}
$$

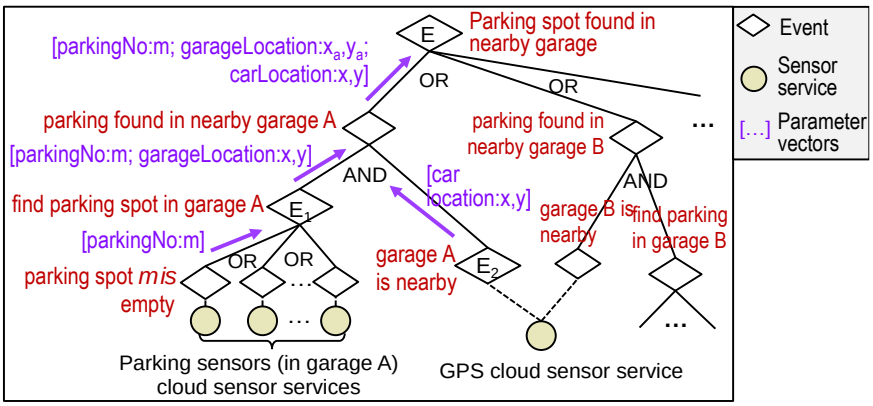


Figure 2: ERT-based event evaluation

initializing Atlas sensor platform, creates a pair of software bundles for each sensor: 1) edge sensor service to be hosted at the Atlas edge middleware, and 2) cloud sensor service to be passed through to the Atlas cloud middleware in the cloud layer. The pair of sensor services communicate with each other, enabling data and control between the edge and cloud layer.

The Cloud layer is built on OSGi Cloud [21] in which applications are composed by loosely-coupling modules as OSGi services hosted at a distribution of cloud nodes. The cloud layer provides solutions that address the cloud-wide discovery, configuration and 'wire-up' of services across different OSGi frameworks in the dynamic cloud environment into applications and services. To help explain our work in this paper, we give more details of two specific components in the cloud layer.

Atlas Cloud Middleware (ACM): Cloud layer holds another prong of the Atlas middleware. For every edge, there exists a corresponding ACM at the cloud layer. It hosts the cloud sensor service bundles passed from the edge and, when the sensor is activated, provision them as services ready to be subscribed to by other cloud services or applications. ACM acts as the cloud gateway to the lower layers, and meanwhile, it hosts the most basic "clouding" of sensors based on which sensor-based cloud applications can be built.

Cloud Application Runtime (CAR): It is the container where application-specific services are deployed and managed. An application makes an invocation to the cloud sensor services at the Atlas cloud middleware to acquire raw sensor readings from the physical deployment.

Note that both ACM and CAR are composed of OSGi frameworks which are installed and provisioned with cloud VMs. Optimizers and caches (application and data) are included at different layers to orchestrate distributed optimizations throughout the CEB cloud-IoT system.

3.2 E-SODA Application Model

CEB could support different application models to utilize different computational abstractions (e.g., events, activities, context, episode, and phenomena). In this paper, we use a specific application model – E-SODA which we first proposed in [30]. It abstracts sensor data into service events. E-SODA follows a rule-oriented paradigm in which an application is composed of a list of event/condition/action rules. In implementation, an application is a composition of interrelated services together performing the function of rule evaluation. Among those services, in this paper, we focus on the Event Services which subscribe to and invoke the cloud sensor services at the ACM to implement event-level abstractions of sensor data. An event service listens to the occurrence of a particular event denoted as its

Equation (2):

$$TFM = < W, I_e >$$
W = nil | date/time – date/time | time – time
I_e = Interval (# of seconds) between two successive evaluations
date = MM/DD/YY
time = hh:mm:ss

(2)

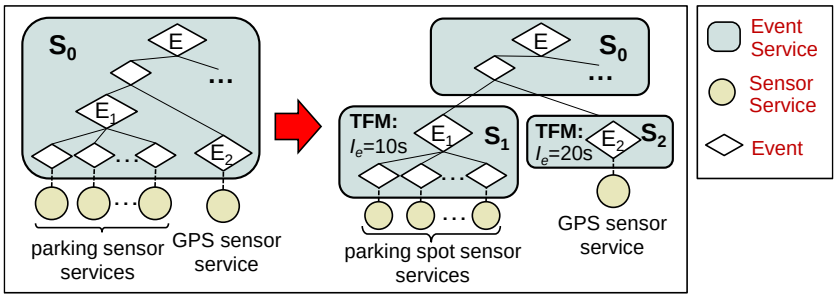


Figure 3: TFM applied to E1 and E2 in the smart parking application

representative event which is a logical expression over sensor values. The event is evaluated against an event representation tree (ERT) based on real-time sensor values. Equation (1) shows a snippet of event composite grammars and Fig. 2 illustrates an ERT tree-based event evaluation to be explained shortly.

In E-SODA, we introduce an application specific relaxation operator, the time/frequency modifier (TFM), which is intended to specify and vary the evaluation rate of events. In itself, TFM is an application-level optimization for what we call sentience-efficiency, and is specified as Equation (2).

Fig. 3 illustrates an application of TFM to the car parking application depicted in Fig. 2. Initially, an event service S_0 pulls sensor readings from parking and GPS sensors to evaluate event E. Later, a TFM (Ie=10s) is applied to E1 to relax the evaluation over parking sensors and another TFM is applied to E_2 to relax the evaluation of GPS sensor to 1/20s. For all parking sensors connected to the same edge, one query can be issued to request data from all sensors and listens to one response that carries all sensor data.

3.3 Bi-Directional Waterfall Optimization Framework

Based on the CEB architecture and E-SODA application model, we summarize our bi-directional waterfall optimization framework [31]. In non-optimized cloud-IoT systems, applications reside in the cloud requesting and processing data originating from the physical layer. To optimize cloud-IoT system operation, we propose a *bi-directional waterfall optimization framework* which allows not only data to move upward but also applications, or more precisely application fragments, to move downward and get cached at lower layers. Under the E-SODA application model in which sensor data are abstracted as events, application fragments that flow from the cloud to the lower layers are event representation trees (ERT). A cached event is evaluated at the layer it is cached to and its event value is pushed back to its upper layer only when it changes (we call this: selective push). For any event cached to a lower layer, a single "shadow event" is created to act as a proxy of the cached event to its consumer, and receiver of selective push messages.

With application caching, *cloud scalability* can be addressed effectively due to the fact that the workload on the cloud is dispersed across a group of edges or even sensor platforms at the beneath layer. Also, optimization opportunities for the *energy consumption* of the sensors can be further provided, because a cloud-IoT system can obtain a local view of both data and applications at any layer, and therefore the interactions and interplays between application and data can be monitored and analyzed at these layers. We have investigated the following four optimization opportunities that can be applied at different layers of CEB (Fig. 4):

- *Cloud-to-Edge Application Fragment Caching Algorithm (AFCA-1) – cloud scalability*: AFCA-1 selects application fragments from the cloud to cache at the edge layer so as to maximize the potential benefits of reducing the usage of cloud resources, while staying within the limitation of the resources in edge servers. Unlike the cloud with elastic resource

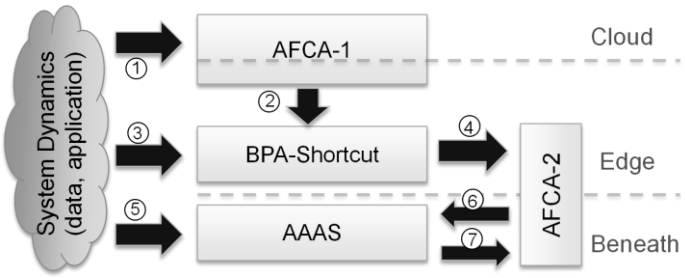


Figure 4: Interplay of optimization algorithms

supply, edge servers have limited resources. AFCA-1 is explained in details in our prior work [32].

- *Shortcut Evaluation and Branch Permutation Algorithm (BPA) – saving sensor energy*: In processing the application fragments cached at the edge layer, shortcut evaluation can be utilized, when a subset of sensor data suffice to derive the occurrence of an event, saving the sensor power due to the skipped sensor samplings. BPA permutes the branches of the ERT affecting the order of sensor sampling and sub-event evaluation to enhance the chances of shortcutting.
- *Application-Aware Adaptive Sampling Algorithm (AAAS) – saving sensor energy*: Atomic events defined in (1) imply application's interest of sensor data. By caching atomic events (the most primitive application fragment) to the beneath layer, the sensor sampling rate can be minimized, while ensuring the adequacy and timeliness of sensor samplings required by the application semantics.
- *Edge-to-Beneath Application Fragment Caching Algorithm (AFCA-2) – saving sensor energy*: AFCA-2 selects the atomic events to cache at the beneath layer to achieve optimized energy efficiency of the sensor nodes. It takes into consideration both the BPA-guided shortcut evaluation as well as AAAS.

3.4 Optimization Algorithms Interplay

AFCA-1 selects events as fragments of the cloud applications and cache them at the edge layer. After events are cached at the edge layer, the BPA-guided shortcut evaluation is then activated at the edge to process the cached application fragments in an energy-efficient way. Specifically, BPA structures the ERT of the cached events to permute the order of the leaf nodes (i.e., atomic events) with the goal of maximizing the occurrence of shortcut to achieve optimized energy efficiency of the sensor nodes. Then, based on the restructured ERT, AFCA-2 is performed to cache the atomic events as more fine-grained application fragments further down to the beneath layer. To assess if an atomic event should be cached at the beneath layer, AFCA-2 calculates the penalty caused by compromising shortcut evaluation on the event (if cached) as well as predicting the benefits to be achieved by performing AAAS at the beneath layer. If the benefit outweighs the penalty, atomic events is cached further down to the beneath layer, which consequently activates the execution of the AAAS algorithm to further reduce the energy cost of the sensor nodes.

Due to the dynamics of the cloud applications and sensor data in the cloud-IoT systems, re-evaluation of the algorithms would be necessary periodically to adapt to any dramatic changes. Change at any layer of the CEB architecture may cause a series of executions or revocations of application caching, which requires very lightweight application caching schemes. In the remaining sections, we present all our optimization algorithms (except for AFCA-1 whose details can be found in [32]) and present a performance evaluation study of the proposed algorithms utilizing a smart-city scale application/data benchmark.

4 Cloud-Aware, Cloud to Edge Application Fragment Caching (AFCA-1)

As discussed earlier, caching application fragments from the cloud layer to the edge layer reduces the workload of processing events on the cloud servers. In addition, with application caching, the data transmission between the cloud and edge layer switches from "pull" to "selective push" (edge pushes an event value to the cloud only when that value changes). This reduces the usage of bandwidth between the cloud and edge as well as computing and other resources in the cloud allocated for data transmission. Consequently, cloud scalability is improved as fewer cloud instances can be provisioned to handle the same amount of tasks. Meanwhile, caching events to the edge layer consumes its resources (e.g., processing and memory). Unlike the cloud whose

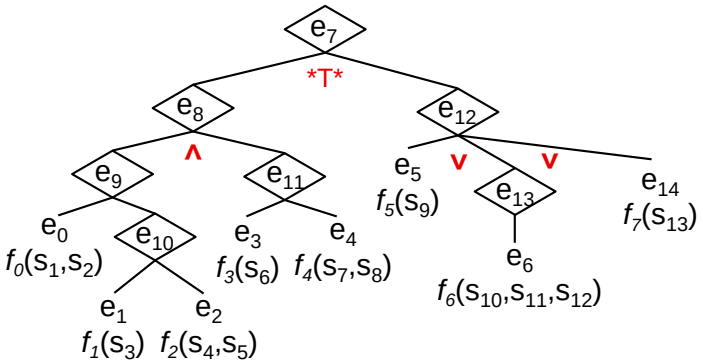


Figure 5: Event representation tree of an event e_7

resources can be provisioned on demand, edge layer consists mostly of commodity servers that have limited resources, and hence we cannot unlimitedly cache applications from the cloud. Guided by this discussion in our prior work [32], we proposed the AFCA-1 algorithm to select the application fragments (i.e., events) from the cloud to cache down at the edge layer with the specific objective: *to minimize the cloud dimension (i.e., number of cloud instances), under the constraint of staying within the resource limitations of the edge servers.*

Importantly, to understand how application caching affects the cloud scale, we have to determine the dominant resources that decide the dimension for all cloud components affected by application caching and examine how their usages are affected by application caching. Additionally, the dominant resources of the cloud components may change over time which makes them variables that affect the logic of AFCA-1. Earlier, we presented an experimental study that guides the determination of and adaptation to such critical variables [32].

5 Power-Aware Processing of Application Fragments At Edge Layer

Under the bi-directional waterfall optimization framework, after application fragments are cached down to the edge layer guided by AFCA-1, the edge layer thereafter obtains a local view of both application and sensor data which allows the interplays between application and data to be observed. Based on such analysis, optimizations can be carried out to process the cached application fragments in a power-efficient manner. In this section, we present two specific optimization algorithms which can be applied collaboratively to optimize sensor energy consumption in cloud-IoT systems.

5.1 Motivation

After an event is cached from the cloud layer to an edge guided by AFCA-1, the edge layer takes the responsibility of evaluating the cached event and reporting its value to the cloud applications. In our approach, event evaluation is performed by traversing the event representation tree. Fig. 5 shows an example in which the evaluation of e7 follows the path: e_0-e_1–e_2–e_{10}–e_9–e_3-e_4–e_{11}-e_8–e_5–e_6–e_{13}–e_{14}-e_{12}–e_7.

With the help of data caching at edge layer, before sending sampling request to the beneath layer, the edge always checks the validity of the cached value for each sensor in an attempt to use the cache and avoid issuing sampling requests to the beneath layer.

Consider the case where an event in the ERT has its two children connected by $\vee$ (logical or) such as e_{12}. Obviously, e_{12} evaluates to true if either of its children (e_5, e_{13} and e_{14}) is true. So there is no need to evaluate the rest of the events when one has already evaluated to true. Similarly, no need to evaluate its siblings to the right when a child event has evaluated to false for both $\wedge$ composite events (e.g., $e_8 = e_9 \wedge e_{11}$) and $*T*$ composite events (e.g., $e_7 = e_8 *T* e_{12}$). In addition, the conditional operation $e_a ? e_b$: can also be converted to combination of logical OR and AND operations$(e_a \wedge e_b) \vee (\wedge)$. This inspires what we call the shortcut evaluation, similar to what can be found in compiler expression optimization, applied in the event evaluation process to reduce the number of events to be evaluated and hence the need to sample sensors for their data. Therefore, taking advantage of data caching and shortcut evaluation, energy consumption of sensor devices can be reduced without affecting the proper behavior of applications because of the skipped data transmissions and sensor samplings. This is an obvious form of sentience-efficiency.

However, our optimization does not stop here. We consider another critical factor that affects the

performance of the event evaluation – the order in which the events in an ERT tree are evaluated. In our specification, the left branch always gets evaluated before its right branches. However, according to the shortcut evaluation strategy, there is a possibility that the value of an event can be derived by evaluating only part of its branches. In our event model, branches of an event that are connected by commutative operators $\wedge$ or $\vee$ (parallel operations) can be swapped without changing the result of event evaluation. Therefore, which branch gets evaluated first can lead to very different performance in terms of the number of events to evaluate and the number of sensors to sample. We therefore propose the branch permutation algorithm that dynamically adjusts the structure of an ERT to manipulate the order in which the tree nodes are evaluated with the purpose of achieving further energy efficiency of sensor sampling. Such branch permutation scheme is motivated by the following observations:

1. For an event whose children are connected by $\vee$, if its left child has high probability of being evaluated to true, then shortcut evaluation will be likely to take place. Similarly, for an event whose children are connected by $\wedge$, if its left child has high probability of being evaluated to false, then shortcut evaluation will likely occur.
2. Or if the branches are not balanced, we would prefer the shortcut evaluation to occur on the "heavier" branch in order to sample fewer sensors.
3. The weight of branch is determined by not only the number of sensors in that branch but also the cache miss rate on those sensors (the lower cache miss rate, the less branch weight). Cache miss rate of a sensor is partially determined by its cache coherence (i.e., time-to-live). In addition, sensors that are more frequently accessed by cloud applications (e.g., shared by large number of applications) tend to have lower cache miss rate. This is because their caches are updated more frequently so that, when their data are required, the data in the cache are more likely to be fresh.

5.2 Branch Permutation Algorithm

Based on above discussion, we give the branch permutation algorithm (BPA) as shown in Listing 1. Generally, BPA starts at the atomic events of an ERT and follows a bottom-up order to perform branch permutation for each event whose branches are connected by either $\wedge$ or $\vee$ operator. For such an event, the algorithm estimates the respective costs of its evaluation under all the possible permutations of its branches (i.e., orders of evaluating its branches). Thereafter, the algorithm chooses the permutation with the minimum cost and reorder the event's branches based on it. The minimum cost will become the cost of evaluating the event and will be saved and later on utilized to estimate the evaluation cost for its ancestor events. In order to perform the algorithm, two information have to be acquired by edge: 1) the probability of event being evaluated to false denoted as probFalse (event) for all leaf events of the ERT, and 2) cache miss rate of sensor s denoted as m(s) for all sensors whose data is required to evaluate the ERT. This information are derived by combining the semantics of both events and data which are obtained by edge through recording recent sensing history.

According to our previous study [7], we consider sensor sampling and data transmission as the two major contributions to the overall energy cost of a sensor device, while neglect processing cost. We use coefficient α_1 and α_2 to represent the energy consumption of a sensor receiving a data request from and sending data to edge respectively and use β to represent the energy cost for one sensor sampling (reading).

Lines 1-2 calculate the sensor energy cost of evaluating a leaf event (atomic event) in the ERT where sensor data are pulled from the sensors for event evaluation. The cost of a pull operation is $\alpha_1 + \alpha_2 + \beta$ (receiving query + sending data + sampling) and it happens only when a cache miss occurs. Lines 9-23 deal with the node (i.e., event) in the ERT of which the operator connecting its branches are either logical OR or AND. It first permutes all of the event's branches and calculates the energy cost of evaluating the event under all possible branch permutations (lines 10-19). Then, the permutation that leads to the lowest energy cost will be chosen according to which the event's branches are re-ordered, and consequently, the lowest energy cost becomes the energy cost of evaluating the event node (lines 20-22).

Line 24 in the algorithm computes the cost of evaluating an event whose children are connected by *T* (sequential operation). δ represents, if event's left-child event *event.first_child* occurs, the frequency of evaluating event's right-child *event.left_child.next_sibling* during $[t_{current}, t_{current}+T]$. Then the total number of times that the right child is to be evaluated can be from 1 to $\delta \cdot T$ and the respective probabilities are $1-p_r, p_r(1-p_r), p_r2(1-p_r),\ldots, p_r \cdot \delta \cdot T \cdot (1-p_r)$ where p_r stands for the probability that event's right child is false. Then the expected number of times that event's right child will be evaluated is Equation (3). Thus, the expected sensor energy cost of evaluating event is calculated as shown in line 29.

In conclusion, by learning and consolidating hints from both applications (ERT) and the sensor data (cache coherence, probability of event), the branch permutation algorithm is able to gain insight as to how sensor data influence the behavior of applications (event evaluation)

BPA: double branchPermutation(event)

Parameter: *event* Root of the tree
Return: minimized estimated cost of evaluating the ERT
Algorithm:

```
1.  if event · first_child == null   // event is an atomic event
2.    return ∑_{s∈ES(event)} (α1 + α2 + β) × m(s)
3.  else
4.   for child = event · first_child ; child != null; child = child· next_sibling
5.    costOfChild = branchPermutation (child);
6.    eventCostMap.put(child, costOfChild);  // <event, evaluation cost> map
7.  endfor
8.  switch (event · operator)
9.   case ˅:
10.   branchPermutations[n] = permute branches of event;
11.   for i =1 to n   // get the i^th permutation
12.    cost[i] = 0;
13.    c = 0;  probOfNotShortcut = 1;
14.    for child = event · first_child ; child != null; child = child · next_sibling
15.     c += eventCostTable.get(child);
16.     cost[i] += probOfNotShortcut * (1 – probFalse(child)) * c;
17.     probOfNotShortcut *= probFalse(child);
18.    endfor
19.   endfor
20.   find minimum cost[k], 1 ≤ k ≤ n ;
21.   re-order the branches of event according to branchPermutations[k];
22.   return cost[k];
23.  case ^ : [ similar to case ˅ ...]
24.  case *T* :
25.   c_l = eventCostMap.get(event · first_child);
26.  c_r = eventCostMap.get(event · first_child · next_sibling);
27.  p_l = probFalse (event ·first_child);
28.  p_r = probFalse (event · first_child· next_sibling) ;
29.   return p_l c_l + (1 − p_l)[(1−p_r^δT)/(1−p_r) − δT) c_r + c_l]
30.  default:
31.  cost = 0;
32.  for i in eventCostTable.keyset(); cost += eventCostTable.get(i); endfor
33.  return cost;
34. endswitch
35. endifelse
```

Listing 1: Pseudocode of Branch Permutation Algorithm (BPA)

Equation (3):

$$E_n = (1-p_r) + 2p_r(1-p_r) + 3p_r^2(1-p_r) + \cdots + \delta T p_r^{\delta T-1}(1-p_r) = \frac{1-p_r^{\delta T}}{1-p_r} - \delta T \quad (3)$$

and based on which it suppresses the demands of non-influential sensor data without affecting the semantics of the applications. The performance of the BPA will be evaluated in the later section.

6 Power-Aware Processing of Application Fragments At Beneath Layer

6.1 Motivation

The edge can extract fine-grained application fragments from the cached event services and cache them in the beneath layer. Such fragments are the atomic events defined over a single sensor and define a desired range of data value (filters) associated with a sampling frequency (TFM). Caching atomic event to the sensor platform enables event processing at the beneath layer which pushes sensor data to the edge layer, only when the value of the cached atomic event is different from the last reported value. Such "filtered push" can further reduce the transmissions between the sensor platform and edge server, leading to lower sensor-node energy consumption.

Furthermore, the cached atomic event metaphorically and practically implies the interest and increased curiosity of the application toward a scope of the sensor data domain. We observe that, since the application only cares if the sensor data falls in or outside of this scope, the sensor sampling rate can be reasonably and confidently set lower, when the current reading is far from the boundaries of the range compared to when the reading is close to the range boundaries. As a result, if we allow a sensor node to adapt its sampling rate at runtime, we can potentially save further sensor energy by reducing the sensor sampling rate, while guaranteeing data currency.

6.2 Existing Adaptive Sampling Models and Algorithms

Several existing algorithms were proposed to achieve adaptive sampling rate by exploiting the temporal correlation among sensor data, while maintaining high data quality. In our early work [33], we proposed a lazy sampling algorithm built on top of the CEB architecture in which the sensor sampling rate is adapted based on the dynamics of the sensor data. However, lazy sampling requires relatively high computation capacity for sensor platforms, especially when sensor data is highly dynamic. At the same time, several models are built based on the temporal correlation exhibited by sensor readings to predict sensor readings, of which the Auto-Regressive Moving Average model (ARMA) [2] is widely adopted. When utilizing ARMA for data prediction, the trend or seasonal components in a history of sensor data needs to be first identified and removed to get stationary residuals. Thereafter, ARMA model can be applied to represent the residuals, and the subsequent sensor data can be predicted by forecasting the residuals and transform to the sensor reading. The ARMA model includes two parts, the auto-regressive part (AR) and the moving average part (MA).

Researchers adopted the ARMA as the data prediction model to skip sensor sampling by predicting next sensor data [5]. The number of skipped samplings increases as long as the predictions remain to be considered accurate based on the proposed data quality model. Initially, the sensor platform samples the first w consecutive readings, and based on these readings, the reading for the epoch w+1 is not only sampled but also predicted. The two values are compared to check the accuracy of the current prediction. If the prediction is considered accurate, the ARMA model can be utilized to the next epoch. Therefore, the sampling of data at epoch w+2 will be skipped. Here, the model defines *CurrentSkipSampleLimit* (*CSSL*) which denotes the number of samples that will be replaced by prediction and is set to 1. For accurate predictions, the *CSSL* is incremented. Otherwise, it is set to zero and the sensor platform has to sample at each time stamp. However, a long sequence of correct predictions may increase the *CSSL* infinitely. To avoid this situation, the approach introduces a constant guard margin – *MaximumSkipSamplesLimit*(*MSSL*) to limit maximum number of samples that can be skipped. More specifically, MSSL is defined to be a constant ($Buffer_{size}$–2) where $Buffer_{size}$ is the size of the buffer that holds the historical sensor data [5].

While ARMA model has a widespread adoption [5][16][19], none of the sampling algorithms are application-aware, missing out significant energy saving clues. In the next section, we propose an adaptive sampling approach that utilizes application semantic cached at the beneath layer to achieve greater energy efficiency of the cloud-IoT systems.

6.3 Application-Aware Adaptive Sampling Algorithm (AAAS)

In CEB, atomic events that represent the most primitive application fragment of E-SODA application model can be further moved from the edge layer and cached (evaluated) at the beneath layer. Atomic events transform a sensor value provided by a sensor to a Boolean, with l_x and u_x indicating the lower and upper boundary of the sensor data range within which the atomic event is evaluated to true. Such beneath layer application caching gives us an opportunity to improve the aforementioned adaptive sampling approach in two aspects:

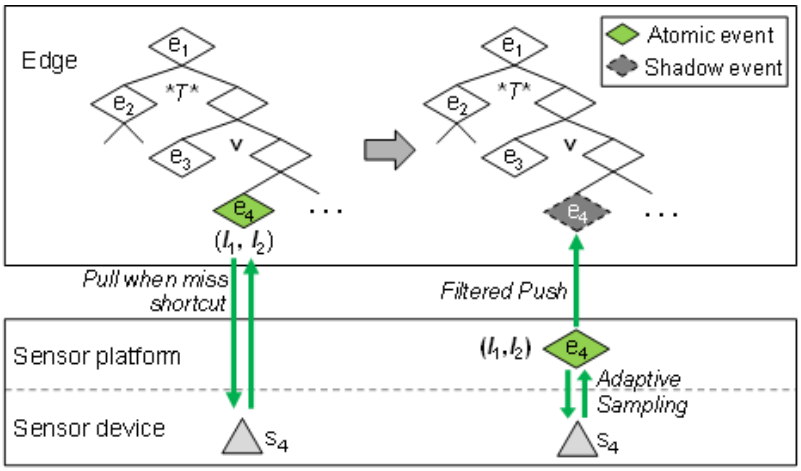


Figure 6: Before/after atomic event e_4 flows to the Beneath Layer

First, we can improve the ARMA-based data prediction model by giving a less stringent accuracy validation model with lower requirement of data quality that is based on the range $[l_x, u_x]$ of the sensor data specified by the cached atomic event. Therefore, a sensor does not track any small change in data readings, which results in fewer observations, and hence better energy efficiency.

Second, instead of being a constant, MSSL can be adjusted based on the offset of the most recent sensor data relative to the range $[l_x, u_x]$ of the sensor data specified by the cached atomic event. If the offset is large, the MSSL can be set larger, setting a larger maximum skip sampling length which affords higher energy savings at acceptable risk. On the other hand, MSSL is set smaller when the current sensor reading is closer to the boundary of the range $[l_x, u_x]$. The adjustment sets a more conservative (smaller) maximum skip sampling length, which would maintain data accuracy and could still save some energy. In addition, there is a chance that the duration of the cached atomic event (i.e., event value is true) is too short to be detected by the MSSL.

Eventually, the MSSL for the data prediction is calculated as in (4), where d is the minimum offset of the most recent sensor data relative to the range $[l_x, u_x]$, and m represents the MSSL value that can be calculated from the user specified maximal probability of missing an event together with the duration of the shortest possible event, and ($Buffer_{size}$-2) makes sure that there are at least two non-predicted sensor reading in the sliding window.

$$MSSL = min(d,\ m,\ Buffer_{size} - 2) \qquad (4)$$

As a result, when the sensor data is relatively static, the prediction of sensor data is more accurate. In this case, the sampling rate of a sensor node is adjusted based on the offset of the sensor reading and the boundaries defined by the cached atomic event.

7 Edge To Beneath Application Fragment Caching Algorithm (AFCA-2)

7.1 Motivation

According to the bi-directional waterfall optimization framework, through the caching of atomic events from the edge layer to the beneath layer, filtered push replaces pull to transmit sensor data from the beneath to the edge layer. The replacement of pull (reactive) with push (proactive) can also reduce the time latency spent to evaluate the ERT. Moreover, with the knowledge of application semantics at the beneath layer, application-aware adaptive sampling can be performed to reduce the sensor sampling rate leading to better energy efficiency of the sensor nodes.

However, caching an atomic event to the beneath layer may not always guarantee the best benefit that could be had, considering the potentially competing benefit of the shortcut evaluation that may occur only at the edge. For example, in Fig. 6, e_4 is an atomic event defined over sensor s_4 that specifies the interested range of the readings of s_4 to be $[l_1, l_2]$. Before caching e_4 at beneath, due to shortcut evaluation there is a chance that the evaluation of e_4 is bypassed by the edge such as when e_2 is evaluated to false or e_3 is evaluated to true. However, after e_4 is cached at the sensor platform in the beneath layer, e_4 becomes blind and cannot contribute to any shortcut occurrences in the edge; it could only push its data to the edge for event evaluation, when the value of e_4 is detected to change even if not needed. In this sense, caching an atomic event to beneath layer can reduce the transmission cost of sensor node by

Algorithm: Calculate P_{ns} *for* e_a

```
1.  node = e_a, P_ns =1;
2.  while node.parent != null
3.    if node · parent · first_child != node
        and node · parent · operator in { ∨, ∧, *T* }
4.     for n = node · parent · first_child; n != node; n = n · next_sibling
5.       eventL.add(n); // eventL contains all siblings left to node
6.     endfor
7.     ST.push(eventL, node.parent.operator);
8.    endif
9.    node = node.parent;
10. endwhile
11. for (eventL, operator) = ST.pop()
12.   switch operator
13.     case ∧ :
14.       for n in eventL   p_ns = p_ns × probFalse(n);  endfor
15.       break;
16.     case ∨ :
17.       for n in eventL   p_ns = p_ns × (1 − probFalse(n));  endfor
18.       break;
19.     case *T* :
20.       p_ns = p_ns × (1 − probFalse(n)); break;
21.   endswitch;
22. endfor
23.   return P_ns;
```

Listing 2: Calculation of P_{ns} for an event e_a

performing filtered push and reducing the sampling cost of sensor node by enabling AAAS; nevertheless, it sacrifices the reduction of energy consumption (both transmission cost and sampling cost) due to the ignorance of the occurrence of shortcut evaluation at the edge. Based on above reasons, a caching benefit evaluation model is proposed next to decide for each atomic event in the ERT cached at the edge whether it should be further cached at the beneath layer or remain in the edge for potential short cut evaluation.

7.2 Beneath Caching Benefit Evaluation Model (BCBEM)

The idea of the Beneath Caching Benefit Evaluation Model (BCBEM) is straightforward. Given an atomic event, it calculates the estimated overall energy saving that can be achieved by caching it to the beneath layer. To calculate the overall energy saving, AFCA-2 needs to consider the benefits brought by the filtered push and AAAS as well as cost of compromising shortcut evaluation. If the calculated energy saving is positive and exceeds a pre-specified threshold (counteract the application caching overhead), the atomic event will be cached at the beneath layer which causes the filtered push to replace pure pull for the data transmission of the sensor node and also starts the execution of AAAS. Again we assume that data transmission (receiving and sending packet) and sensor sampling are the two major contributors to the overall energy consumption of a sensor node.

First, the energy consumed per second by sensor s to evaluate the atomic event e_a (associated with sensor s) before caching it to the beneath is calculated in (5). Where α_1, α_2 and β are the energy coefficients of a sensor node defined in section 5. P_{ns} indicates the probability of shortcut not occurring to e_a on one event evaluation. f_s denotes the evaluation frequency (1/sec) for e_a defined by TFM.

$$C_{before} = P_{ns} \cdot (\alpha_1 + \alpha_2 + \beta) \cdot f_s \qquad (5)$$

To calculate P_{ns} for event e_a, two-round traversal of the ERT is needed which is shown in Listing 2.

At the first round, ERT is traversed from bottom to up, starting at the node e_a. When a node with the operator falls in {∨, ∧, *T *} is encountered and e_a is not at its leftmost branch, all the children of the node left to the branch where e_a is located (i.e., can cause shortcut evaluation) along with the operator will be pushed to a stack *ST* (lines 3-8). This process continues until it reaches the root of the ERT. In the second round, the algorithm pops ERT elements (i.e., event lists and operators) iteratively from the *ST*, and meanwhile calculates P_{ns}. For example, in Listing 2, P_{ns} for e_4 is calculated as $(1 - probFalse(e_2)) \times probFalse(e_3)$ which indicates that e_4 will get evaluated, only when e_2 occurs while e_3 does not.

Next, we calculate the energy cost per second of sensor *s* after caching the atomic event e_a to its sensor platform. This includes two components: 1) the transmission cost of using filtered push, and 2) the sampling cost after using AAAS. To calculate the first component, the edge monitors the probability that the result of e_a is evaluated to be different from the previous evaluation (push data) over time, denoted as P_c. Then this component of energy consumption is calculated as $C^1_{after} = P_c \cdot \alpha_2 \cdot f_s$.

To estimate the second component, the accuracy of sensor data prediction cannot be foreseen by the edge layer. However, we know that the sampling frequency f_s' falls in the range of [1/*MSSL*, f_s]. That is, $f' = \tau \cdot f_s$, $1/(MSSL \cdot f_s) \leq \tau \leq 1$. The calculation of MSSL is given in (4). Then, two strategies can be used in estimating f_s'. One is optimistic, which assumes that f_s' is 1/*MSSL* (i.e, $\tau = 1/(MSSL.f_s)$ and the other is pessimistic, which assumes that f_s' equals to f_s (i.e.,$\tau = 1$). Therefore, the second component of energy cost is calculated as $C^2_{after} = \tau \cdot \beta \cdot f_s$, and the overall energy cost per second of sensor *s* after caching to the beneath is

$$C_{after} = C^1_{after} + C^2_{after} = (P_c \cdot \alpha_2 + \tau \cdot \beta) \cdot f_s \qquad (6)$$

Now, caching evaluation model calculates the energy saving rate after caching e_a to the beneath layer and compares the result with a positive constant threshold σ (counteract event caching overhead). If the saving rate is higher than σ, then the atomic event e_a will be cached to the beneath layer. Otherwise, it remains in the edge layer and uses pull (after shortcut fails) to acquire sensor data from the beneath layer. Then, the condition of performing caching e_a to the beneath is

$$C_{before} - C_{after} = P_{ns} \cdot \alpha_1 + (P_{ns} - P_c) \cdot \alpha_2 + (P_{ns} - \tau) \cdot \beta > \sigma \qquad (7)$$

After e_a is cached at the beneath layer, τ keeps being tracked by the sensor platform and will be sent back to the edge layer, if it is greater than its original value by a certain amount (the actual energy saving by AAAS is less than what is expected by AFCA-2). This will trigger the AFCA-2 to re-evaluate the BCBEM for the cached atomic event to decide if its caching needs to be revoked. In addition, as we will explain in the next section, the BCBEM requires to be evaluated periodically in order to adapt to the dynamics from both the application and data domain.

7.3 Description of AFCA-2

To maintain reliable and beneficial optimization performance, system dynamics need to be monitored and reacted to properly. These dynamics primarily come from two domains - application and data, as we summarize as follows:

Application domain dynamics. New event services (i.e., ERTs) in the cloud are chosen to cache at the edge layer by AFCA-1, or event services that are cached at the edge layer are removed by AFCA-1. In either of the above cases, we say that the application domain dynamic is observed on the added or removed ERTs.

Data domain dynamics. Due to the dynamics of the sensor data, parameters used to calculate the BCBEM such as P_{ns} , P_c and τ may change at runtime. For any of these parameters, if the absolute difference between its current value and the value used to calculate the BCBEM exceeds a particular level (i.e., dynamics observation threshold), the dynamics of the data domain is said to be observed, meaning that the earlier caching benefit estimated by AFCA-2 becomes unreliable.

Combined application and data domain dynamics. In order to adapt to the dynamics from the data domain, the Branch Permutation algorithm takes actions periodically, which could change the structure of the ERTs (application dynamics) cached at the edge layer. The ERT structure change could consequently affect the value of P_{ns} for the atomic events of the ERT, which makes earlier caching benefit estimated by AFCA-2 become unreliable.

Therefore, the AFCA-2 has to keep track of the factors that are listed in the above three categories to monitor the system dynamics. In order to adapt to any of three categories of dynamics, we establish three specific actions that need to be performed by AFCA-2:

1. Action A. Evaluating the BCBEM for all the atomic events (leaves) of the affected ERT's.
2. Action B. Based on the newly estimated caching benefits, cache new atomic events to the beneath layer, or
3. Action C. Revoke an earlier cached atomic event from beneath, if the estimated caching benefit is negative

Algorithm: AFCA-2

Gobal Variable: *ActionList <action>* // In decreasing order of Δc
Thread-1:

```
1.  while an ERT t is observed to be affected by the system dynamics
2.   for e in all of t's atomic events
3.    if  t is not cached at beneath
4.      Δc = Equation (7);
5.      if Δc > σ (caching overhead constant)
          // Action (action_type, atomic event, Δc)
6.        Action a = new Action (DO_CACHE, e, Δc);
7.        insertIntoActionList (a);
8.      endif
9.     else
10.     Δc = -Equation (7);
11.     if Δc > σ (caching revocation overhead)
12.      Action a = new Action (REVOKE _CACHE, e, Δc);
13.      insertIntoActionList (a);
14.     endif
15.   end-if-else
16.  endfor
17. endwhile
```

Thread-2:

```
1. while ActionList is not empty
       AND edge resource usage does not exceed max quota
2.  Action a = ActionList.remove();
3.  if a belong to action B
4.    cache atomic event e in action a to the beneath layer;
5.  else
6.    revoke the caching of event e in action a from the beneath layer;
7.  end-if-else
8. endwhile
```

Listing 3: Pseudocode of AFCA-2 Algorithm

Apparently, without any regulations, the level of system dynamics determines the frequency of performing the above actions. As discussed earlier, unlike the cloud with elastic resource supply, edge servers have limited resources (computing and memory), which requires the AFCA-2 to be concerned about the resource limit at the edge server, while performing the *A*, *B* and *C* actions. To achieve this requirement, we first examine and compare the resource usage by action *A, B* and *C* via experiments (in the experiment section) and from the results, we observed the following fact:

Actions B and C use much more edge resources than A.

Based on this observation, we designed a dynamics adaptation scheme in which action *A* is performed, whenever the system dynamics is observed. After the completion of action *A*, a set of action *B* and *C* are created and enter standby mode (ready to be performed). For each of these actions, a value ΔC is calculated as a by-product of performing action *A* indicating the energy saving that can be achieved by taking that action. For action *B*, the value of ΔC is calculated in (7); and for action *C*, the value of ΔC is the negation of the value calculated in (7). Then AFCA-2 orders all the actions of *B* and *C* by its ΔC in decreasing order, and opportunistically performs these actions in sequence so long as the current resource usage of the edge server has not reached its maximum quota.

Based on above discussions, we describe the AFCA-2 algorithm in Listing 3.

7.4 The Effects of AFCA-2

Now we discuss the effects of performing AFCA-2 on the energy consumption of the CEB-based cloud-IoT systems. Given an event representation tree (ERT) cached at the edge layer, through branch permutation algorithm (BPA), events that act as the shortcut enablers (i.e., shortcut the evaluation of its sibling events) tends to be evaluated earlier than the rest of the events (i.e., placed at the left branch of the tree). In AFCA-2, we can infer from (7) that the nodes: 1) located at the left branches (i.e., higher P_{ns}), and 2) whose value change slowly (i.e., lower P_c and τ) tend to be selected and cached down to the beneath layer. Therefore, running AFCA-2 after BPA causes the atomic events who play as shortcut enablers and whose value changes relatively slowly to be cached at the beneath layer. Because the value of these events rarely changes, the sensor sampling as well as the data transmission caused by evaluating these events are greatly suppressed. Also, even if the value of such event changes (would be sent to the edge layer), it will be very likely to shortcut the evaluation of its sibling events at the ERT. Therefore, AFCA-2 can significantly suppress the system actions and improve the energy efficiency of sensor devices in the cloud-IoT system. In a later section, we validate the above analysis through experiments.

8 Experimental Evaluation

In this section, we quantify and measure the effects of various combinations of the AFCA-1, BPA/Shortcut, AAAS and AFCA-2 on energy saving of the sensor devices in the cloud-IoT systems utilizing our CEB architecture. We first set up a prototype of CEB architecture on which smart home sensor based applications are deployed to monitor a variety of events. To prepare test cases that reach city scale, we synthesized a benchmark for both sensor data and cloud applications based on a real dataset. It is noted that evaluation of CEB scalability is not presented in this paper. Initial scalability results can be found elsewhere [32].

8.1 A Benchmark for Cloud-IoT Data and Applications

We use our previously developed benchmark for large-scale cloud-IoT systems explained with details in [30]. The data/application benchmark is for a smart home cloud-IoT system with a scale of 2000 houses in which a variety of applications (emergency-detection, security, activity recognition, and healthcare) in the form of events are created based on a huge set of household and resident-worn sensors. It is based on the PLCouple1 dataset collected from the PlaceLab [22] and the events and sensor data have been further synthesized to be extended to 2000 smart homes.

Based on the data/application benchmark, we investigate the energy-saving performance of our proposed optimizations. To do so, we first create the main metrics of sensor energy consumption for performance evaluations.

8.2 Evaluation Metrics

One of the main performance metrics, which is measured throughout all experiments, is the energy saving rate R_{save} of the specific optimization approach or combination group, all with respect to the reference no-optimization or "pure-pull" scheme. The energy saving rate in all experiments is given by

$$C_{pure-pull} = \sum_{s \in SENSORS} (\alpha_1(s) + \alpha_2(s) + \beta(s)) \cdot N(s) \quad (8)$$

$$C_{optimized} = \sum_{s \in SENSORS} (\alpha_1(s)\, N_{recv}(s) + \alpha_2(s)\, N_{send}(s) + \beta(s)\, N_{sampl}(s)) \quad (9)$$

$$R_{save} = 1 - \frac{C_{optimized}}{C_{pure-pull}} \quad (10)$$

Equation (8) indicates the total energy cost for all the sensors (i.e., *SENSORS*) in the cloud-IoT system by using the "pure-pull" scheme. The $\alpha_1(s)$, $\alpha_2(s)$ and $\beta(s)$ denote the energy cost factors explained earlier (in Branch Permutation Algorithm) for the particular sensor s, and N(s) means the total number of data requests received by sensor s during the experiment. Equation (9) represents the total energy cost for all the sensors by adopting a particular optimization approach, where $N_{recv}(s)$ denotes the total number of messages (packets) received by sensor s during the experiment, $N_{send}(s)$ denotes the total number of messages sent by sensor s, and N_{sampl}(s) represents the total number of samplings acted by sensor s.

Another performance metric measured in our experiments is the percentage of the atomic events chosen by AFCA-2 to cache to the beneath layer (to perform AAAS) among all the atomic events at the edge

Table 1: Four Experiment Study Groups

Experiment Groups	Conversion from Gaussian and CGS EMU to SI [a]
1	Pure Pull
2	Add Shortcut Evaluation to group 1
3	Add BPA to group 2
4	Add AFCA-2 (selective push, AAAS) to group 3

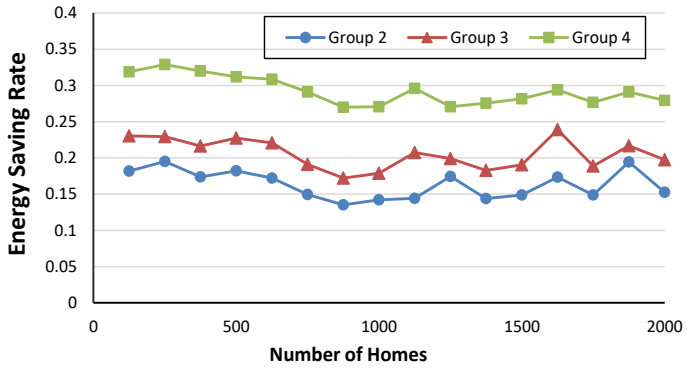


Figure 7: Average energy saving rate for three experiment groups with different number of homes

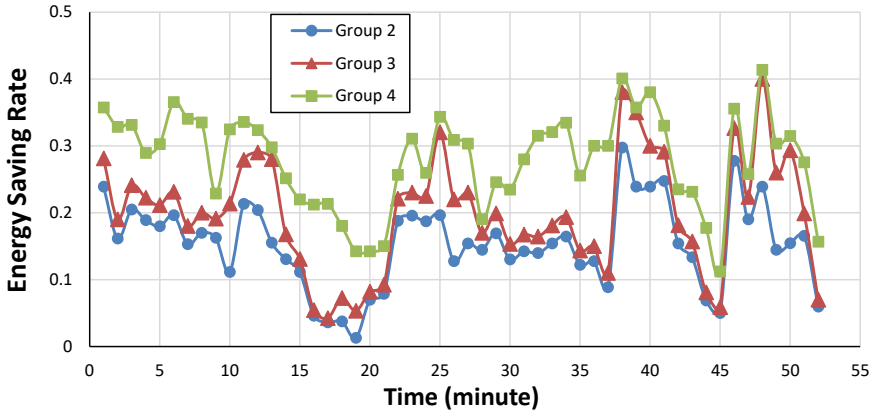


Figure 8: Energy saving rate for three experiment groups along timeline (number of homes = 2000)

layer. The increase or decline of this metric reflects the growing inclination of AFCA-2 towards the AAAS or Shortcut Evaluation algorithms respectively, which consequently affects the push-pull envelop between the edge and the beneath layer.

8.3 Performance Evaluation Results

We first present four groups of experiments; three of which correspond to combinations of our optimization approaches, and one being the reference, no optimization (pure pull) experiment. Then we conduct experiments for each group and compare their performances. Furthermore, during the experiments, we vary several parameters of the tested event set (e.g., dynamism of the events) to examine its effects on the overall energy savings as well as the decision making by AFCA-2 in selecting the atomic event for application caching at the beneath layer.

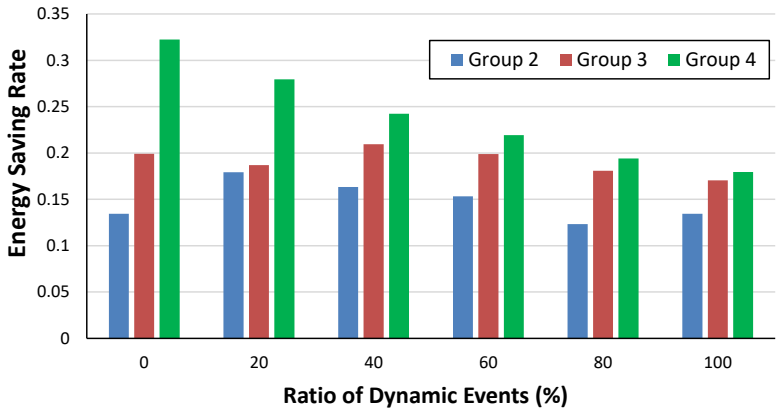


Figure 9: Performance for a spectrum of dynamic events

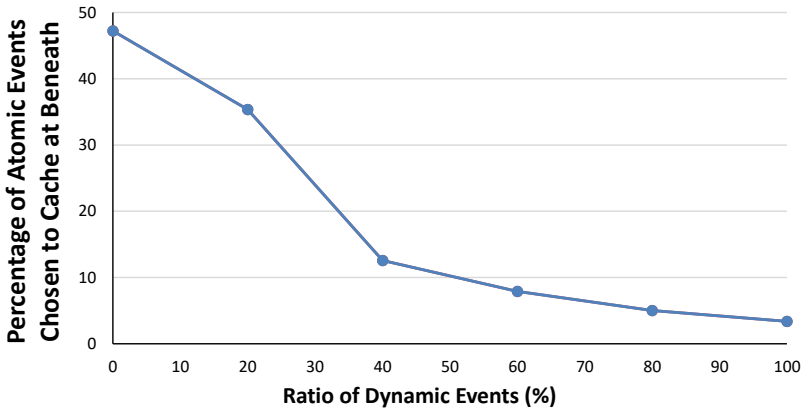


Figure 10: Percentage of atomic events chosen to cache at beneath layer by AFCA-2 with different ratios of dynamic events

8.3.1 Experiment I: Comparison of Combination Group of Optimizations

Table I lists four experiment groups each evaluates and analyzes the effect of applying a particular combination of optimization methods that we propose in this paper.

We compared the performance (i.e., average R_{save}) of the experiment groups 2-4 by choosing the number of smart homes that participate in the experiment as the stress variable, and showed the results in Fig. 7. The results demonstrate that as more optimization algorithms are combined, the CEB system performance is improved. Shortcut Evaluation was found to be responsible for 16% of energy saving on average. The Branch Permutation Algorithm add-on to application caching showed marginal additional energy savings of about 4%-5%. The combined application of Shortcut, BPA and AAAS almost doubled the savings in energy to a hefty 28%. In addition, from the experiment results, changing the number of smart homes does not obviously affect the performance of the optimizations.

Fig. 8 records the energy saving rate (i.e., R_{save}) along the timeline, when the number of homes is 2000. From the results, we see that the performance of the optimizations drops dramatically several times during the experiments (e.g., at time 18 and time 45). These drops result from the fact that the dynamics from the data and applications render the optimization decisions made earlier by the algorithms to stale. Therefore, re-evaluations of these algorithms were performed after these dramatic performance drops in order to adapt to the system dynamics which causes the subsequent optimization performance rise as shown in the figure.

Next we vary several parameters of the experiment test cases in order to investigate how their changes could affect the performance of the optimization approaches on sensor energy saving and the application caching decision made by AFCA-2. The first parameter we choose is the event dynamics.

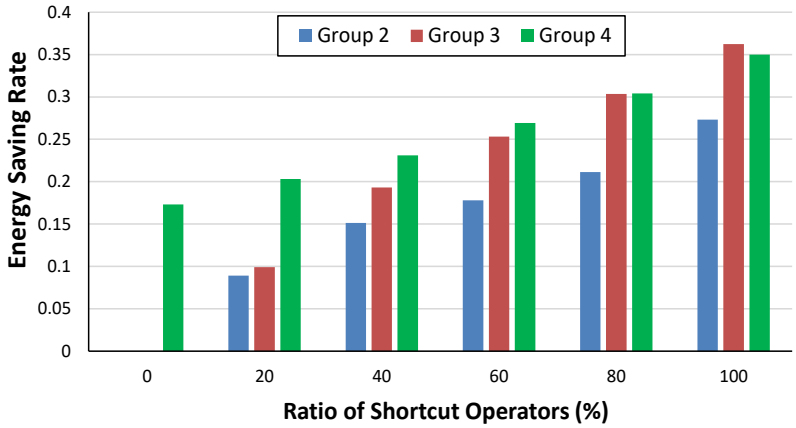


Figure 11: Performance for a spectrum of shortcut operators, AFCA-2 with different ratios of dynamic events

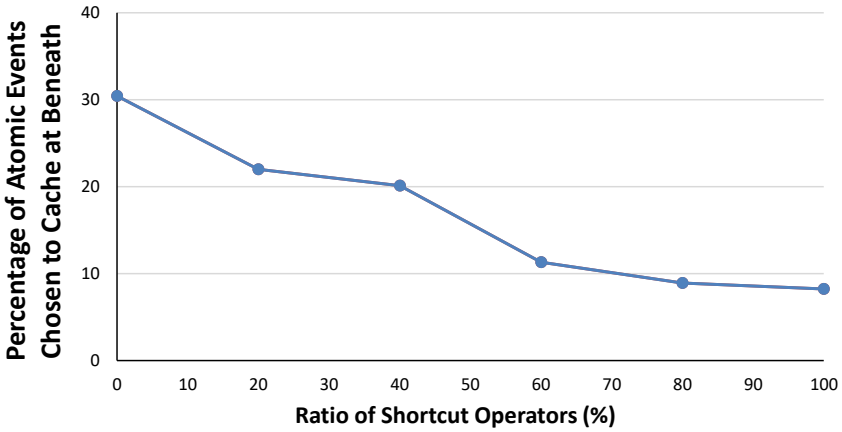


Figure 12: Percentage of atomic events chosen to cache at beneath layer by AFCA-2 with different ratios of shortcut operators

8.3.2 Experiment II: Spectrum of Dynamic Events

To validate the reaction of the optimizations to different event dynamics, we manipulated the basic events we created in the data/application benchmark by enlarging or reducing the range of query specified by their atomic events to reduce or increase the event dynamic changes respectively. We classify the events into two types: dynamic event, with the average rate of event value change higher than 0.20/sec, and static event, with the average rate of event value change lower than 0.05/sec. In the experiment, we changed the ratio of the dynamic events in the basic event set and recorded the results in Fig. 9. From the results, we can see that the performance of shortcut and BPA did not change much, when the ratio of the dynamic events varies. However, the performance of AAAS declined obviously, as the ratio increases. This is consistent with the results shown in Fig. 10 which records the number of atomic events that are cached from the edge to the beneath layer by AFCA-2 with different ratio of dynamic events in the cloud-IoT systems. From Fig. 10, fewer atomic events were chosen by AFCA-2 to cache at the beneath layer, when their value change rate is higher.

This is because, based on (7), higher event dynamics cause higher P_c and τ which makes AFCA-2 think the benefit to be obtained from AAAS would be lower. And since the benefits of shortcut evaluation obtained at the edge layer are not affected significantly by the variation of event dynamisms (as observed in Fig. 10), lower benefit of AAAS will make AFCA-2 inclined to making decision of not caching events from the edge to the beneath layer.

8.3.3 Experiment III: Spectrum of Shortcut Operators

In this experiment, we continue to examine the effects of another parameter to the performance of the optimization algorithms and the application caching decision made by AFCA-2 – the ratio of "shortcut" operator (i.e., AND, OR and *time*) among all the event operators. Again, we recorded the result of energy saving rate achieved by our optimizations at a spectrum of ratio of shortcut operators in Fig. 11.

Fig. 11 shows that the energy saving rate achieved by BPA-Shortcut increases along with ratio of shortcut operators. However, its speed of growth is much larger than the speed of growth for the overall benefit of the combination of Shortcut/BPA and AAAS. Especially, the adoption of AAAS does not help achieve much of additional energy saving, when the ratio of shortcut operators reaches around 60%. In Fig. 12, we record the percentage of atomic events chosen to cache at beneath layer by AFCA-2 with different ratios of shortcut operators. The percentage of cached atomic event drops from ~30% to 8%, as the ratio of shortcut operators increases from 0% to 100%. This is because when the number of shortcut operators is high, the change of shortcut evaluation occurred at the edge layer is consequently higher. Therefore, AFCA-2 thinks the energy saving achieved from shortcut evaluation at the edge layer would be superior to the savings from AAAS for most of the atomic events.

9 CONCLUSIONS

IoT applications and web services are pressed to reside on the cloud for many practical reasons especially in large-scale IoT deployments, including reductions in services cost and equal access to all stakeholders. However, this will require extensive interactions between the cloud (applications and services) and the physical world (devices to be controlled and sensors whose data is queried by the applications and services). This will pose challenges to the scalability and power awareness at scale. Edge computing offers great opportunities to architect scalable and energy-optimized Cloud-IoT systems. We exploit the edge to bring the physical world and its data up closer to the cloud and to cache "fragments" of the cloud applications down closer to the physical world. We presented a three-tiered waterfall optimization framework and developed four optimization algorithms that exploit the combined effect of data/application dynamics in managing scale and reducing energy use for IoT deployments. The novelty of the framework is the definition and use of "sentience-efficiency" which is a dynamic utilization of joint semantics of data/applications to reduce the work needed to execute applications and minimize the movements (data and applications). We investigated the energy-saving performance using a cloud-IoT smart home data/application benchmark of 2000 houses with variety of applications based on a huge set of household and resident-worn sensors, where the energy saving rate of a specific optimization approach or a combination group is the main performance metric and the percentage of the atomic events chosen to cache to the physical layer is another metric. The results demonstrate that as more algorithms are combined, the more the system performance is improved. The Shortcut Evaluation introduced an average of 16% energy saving, the BPA add-on to application caching showed an additional energy savings of about 4%. The combined application of Shortcut, BPA and AAAS showed energy saving of 28%. Changing the number of smart homes does not obviously affect the performance. To validate the reaction to event dynamics, we manipulated the range of query.

REFERENCES

[1] Amazon EC2. https://aws.amazon.com/ec2/, Last accessed 30th May 2019.

[2] ARIMA.https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average. April 2019.

[3] Y. Bai, S. Liu, M. Sha, Y. Lu, and C. Xu, "An energy optimization protocol based on cross-layer for wireless sensor networks," JCM, vol. 3, no. 6, pp.27-34, 2008

[4] R. E: Bellman, Dynamic Programming. Princeton University Press, Princeton, NJ, 1957.

[5] S. Chatterjea, and P. Havinga,, "An adaptive and autonomous sensor sampling frequency control scheme for energy-efficient data acquisition in wireless sensor networks," In DCOSS'08, pp. 60-78, June 2008.

[6] C. Chen, and A. Helal, "Device integration in SODA using the device description language," In 2009 Ninth Annual International Symposium on Applications and the Internet, pp. 100-106, July 2009.

[7] C. Chen, Y. Xu, K. Li, and S. Helal, "Reactive programming optimizations in pervasive computing," In 2010 10th IEEE/IPSJ International Symposium on Applications and the Internet, pp. 96-104, July 2010.

[8] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," In Proceedings of the 13th VLDB, pp. 588-599, August 2004

[9] S. Deugd, R. Carroll, K. Kelly, B. Millett, and J. Ricker, "SODA: Service oriented device architecture," IEEE Pervasive Computing, vol. 3, no. 1, pp.94-96, 2006.

[10] A. Feistel, M. Wiczanowski, and S. Stanczak, "Optimization of energy consumption in wireless sensor networks," In Proceedings of ITG/IEEE International Workshop on Smart Antennas (WSA), pp. 26-27, 2007.

[11] S. A. Hashish, and A. Karmouch, "Topology-based on-board data dissemination approach for sensor network," In Proceedings of the 5th ACM international workshop on Mobility management and wireless access, pp. 33-41, October 2007

[12] J. A. Hartwell, G. Messier, and R. J. Davies, "Optimizing physical layer energy consumption for wireless sensor networks," In IEEE 65th Vehicular Technology Conference, pp. 76-79, April 2007.

[13] R. Jurdak, P. Baldi, and C. V. Lopes, "State-driven energy optimization in wireless sensor networks," In Proceedings of 2005 Systems Communications (ICW'05, ICHSN'05, ICMCS'05, SENET'05), pp 356-363, August 2005.

[14] J. King, R. Bose, H. I. Yang, S. Pickles, S. and A. Helal, "Atlas: A service-oriented sensor platform: Hardware and middleware to enable programmable pervasive spaces," In Proceedings of the 31st IEEE Conference on Local Computer Networks, pp. 630-638, November 2006.

[15] D. Kossmann, "The state of the art in distributed query processing," ACM Computing Surveys, vol. 32, no. 4, pp.422-469, 2000.

[16] C. Liu, K. Wu, and M. Tsao, "Energy efficient information collection with the ARIMA model in wireless sensor networks," in IEEE Global Telecommunications Conference, pp. 1-5, December 2005.

[17] X. Liu, Q. Huang, and Y. Zhang, "Balancing push and pull for efficient information discovery in large-scale sensor networks," IEEE Transactions on Mobile Computing, vol. 6, no. 3, pp.241-251, 2007.

[18] A. Masoum, N. Meratnia, and P. J. Havinga, "A decentralized quality aware adaptive sampling strategy in wireless sensor networks," In UTC/ATC'12, pp. 298-305, September, 2012.

[19] K. Miranda, and T. Razafindralambo, "Using efficiently autoregressive estimation in wireless sensor networks," In CITS'13, pp. 1-5, May 2013.

[20] Open Services Gateway Init. (OSGi 4.2) Specification, osgi.org/download/r4v42/r4.cmpn.pdf, August. 2009.

[21] OSGi Cloud Computing (RFP133). https://www.osgi.org/bugzilla/show_bug.cgi?id=114, April 2013.

[22] PlaceLab. http://web.mit.edu/cron/group/house_n/data/PlaceLab/PlaceLab.htm, 2005.

[23] K. S. Prabh, and T. F. Abdelzaher, "Energy-conserving data cache placement in sensor networks," ACM TOSN, vol. 1, no. 2, pp.178-203, 2005

[24] M. A. Rahman, and S. Hussain, "Effective caching in wireless sensor network," In 21st AINAW'07, pp. 43-47, May 2007.

[25] S. Reilly, and M. Haahr, "Extending the event-based programming model to support sensor-driven ubiquitous computing applications," In IEEE International Conference on Pervasive Computing and Communications, pp. 1-6, March 2009.

[26] M. Satyanarayanan, V. 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.

[27] A. Silberstein, R. Braynard, and J. Yang, "Constraint chaining: on energy-efficient continuous monitoring in sensor networks," In Proceedings of the 2006 ACM SIGMOD, pp. 157-168, June 2006.

[28] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: A publish/subscribe middleware for sensor networks," *Personal and Ubiquitous Computing*, vo. 10, no. 1, pp.37-44, 2005.

[29] Z. J. Tao, Z. H. Gong, Z. Z. OuYang, and J. Y. Xu, "Two new push-pull balanced data dissemination algorithms for any-type queries in large-scale wireless sensor networks," In 2008 i-Span Conference, pp. 111-117, May 2008.

[30] 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, 2015.

[31] Y. Xu, and S. Helal, "An optimization framework for cloud-sensor systems," In IEEE 6th International Conference on Cloud Computing Technology and Science, pp. 38-45, December 2014.

[32] Y. Xu, and S. Helal, "Application caching for cloud-sensor systems," In Proceedings of the 17th

ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 303-306, September 2014.

[33] Y. Xu, S. Helal, M. Thai, and M. Scmalz, M., "Optimizing push/pull envelopes for energy-efficient cloud-sensor systems," In Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 17-26, October 2011.

[34] L. Ying, Z. Liu, D. Towsley, and C. H. Xia, "Distributed operator placement and data caching in large-scale sensor networks," In Proceedings of the 27th Conference on Computer Communications, pp. 977-985, April 2008.

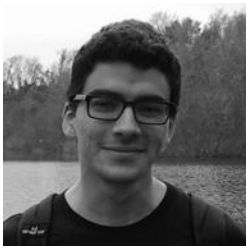
[35] Z. Zhang, M. Ma, and Y. Yang, "Energy-efficient multihop polling in clusters of two-layered heterogeneous sensor networks," *IEEE Transactions on Computers*, vol. 57, no. 2, pp. 231-245, 2008.

AUTHOR BIOGRAPHIES

Yi Xu received the Ph.D. degree in 2014, 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 and mobile computing, programming models and middleware for cloud-IoT systems, and internet of things.

Abdelsalam (Sumi) Helal received the Ph.D. degree in computer sciences from Purdue University, West Lafayette, IN, USA. He is currently professor and the Chair in Digital Health, School of Computing and Communications, and the Division of Health Research, Lancaster University, UK. Before joining Lancaster University, he was professor in the department of Computer and Information Science and Engineering, University of Florida, USA, where he directed the Mobile and Pervasive Computing Laboratory and the Gator Tech Smart House. His research interests span pervasive systems, the Internet of Things, smart spaces, with applications to digital health and assistive technologies for successful aging and independence.

Choonhwa Lee received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 1990 and 1992, respectively, and the Ph.D. degree in computer eng. from the University of Florida, USA, in 2003. He is currently a Professor with the Dept of Computer Science and Engineering, Hanyang University, Seoul, South Korea. His research interests include cloud computing, peer-to-peer and mobile networking and computing, and services computing technology.



Ahmed E. Khaled received the Ph.D. degree (August 18) in computer science from University of Florida, Florida, USA. He is currently assistant professor, computer science department, Northeastern Illinois University, USA. He received the B.Sc. and M.Sc. degrees in computer engineering from Cairo University, Egypt in 2011 and 2013, respectively. His current research interests include Internet of Things, smart spaces, and ubiquitous computing.