2014 IEEE 6th International Conference on Cloud Computing Technology and Science

An Optimization Framework for Cloud-Sensor Systems

Yi Xu and Sumi Helal
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32601, USA
{yixu, helal}@cise.ufl.edu

大规模物联网部署需要一个云传感器架构来促进生态系统的无阻集成和可编程性。除了这两个功能要求之外，任何这样的架构都必须解决性能和可扩展性的挑战。
1. 引入了Cloud-Edge-Beneath（CEB）架构
2. 通过内置的分布式优化框架解决了可扩展性和性能问题。在本文中，我们专注于CEB的优化框架，其遵循双向瀑布模型，其中不仅传感器数据可以向上移动到应用，而应用（片段）可以向下移动到更靠近数据源的CEB的较低层。

***Abstract*— Imminent massive-scale IoT deployments require a Cloud-Sensor architecture to facilitate an ecosystem of friction-free integration and programmability. In addition to these two functional requirements, challenging performance and scalability requirements must be addressed by any such architecture. We have introduced the Cloud-Edge-Beneath (CEB) architecture which addresses scalability and performance through a built-in distributed optimization framework. In this paper, we focus on CEB's optimization framework which follows a bi-directional waterfall model in which not only sensor data can move upward to applications, but applications (fragments) can move downward to lower layers of CEB closer to data sources. The framework enables many optimization ideas and opportunities, including our own. We present the bi-directional waterfall framework along with a sketch of several of our optimization algorithms enabled by the framework. We also present an example of an experimental study to determine dominant resources in the cloud – a variable which as will be seen greatly affects the logic of some of the optimization algorithms.**

Keywords- Cloud-sensor systems; application caching; cloud computing; scalability; performance; optimization.

I. INTRODUCTION

motivation 1. 包括终端用户、设施管理者在内的各方群体都希望获得基于传感器的服务，在没有标准化的cloud-hosted的web服务模型下这是很难实现的。

Recent advances in Internet of Things (IoT) and pervasive and ubiquitous computing reveal exciting visions of smart many things: smart cities, smart homes, smart cars. As smart spaces and smart cities proliferate into a massive scale, sensor data and services (applications) will be pressed to move to the cloud given its economies of scale and highly anticipated reductions in services costs. This migration will also be driven by the fact that masses of various stake holders (ranging from end users to facilities managers to utilities and traffic control operators, to mention just a few) will want access to the sensor-based services, which is difficult to achieve without a standardized cloud-hosted web services model. In this case, we end up with "cloud-sensor systems" that critically require an architecture to facilitate an ecosystem of friction-free integration (cyber-physical) and programmability. In addition to these two functional requirements, we argue that any proposed architecture for an efficient, scalable cloud-sensor operation must address the following two challenges.

1、无冲突（网络-物理设备），2、可编程的生态系统架构
3. 云扩展性
4. 传感器设备的能源限制

Cloud Scalability: Extensive external interactions between cloud services and the physical sensors could pose significant challenges to the scalability of the overall system. Connecting hundreds of millions of sensors directly to the cloud is extremely demanding on cloud resources and will not scale. It results in expensive cloud "attention", not only per sensor, but per each sensor duty cycle. For example, if sensors push data once every minute, then millions of sensors will produce billions of sensor-cloud interactions, daily. This will require tremendous processing power, memory resources and huge incoming/outgoing cloud traffic. As a result, the cloud economies of scale per sensor will not stand, meaning that the cloud will become too expensive to pay for given the existing use-based price models.

云服务和物理传感器之间的广泛的外部交互可能对整个系统的可扩展性造成巨大挑战。 将数亿个传感器直接连接到云对云资源极其苛刻，而且难以扩展。

Energy Constraint of Sensor Devices: Unlike elastic cloud resources which can be provisioned on demand, sensor devices cannot be provided dynamically. Many sensors are battery embedded which makes them extremely 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 leads to continuous data sampling by the sensor nodes and transmission through the sensor network which incurs substantial energy cost to 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 use of sensors is critically needed or cloud-sensor systems will not be dependable. To achieve this goal, the supply of data from the sensors and the demand of this data from cloud applications will need to be carefully optimized. We adopt a three-tier architecture called the Cloud-Edge-Beneath (CEB), which enables sensor integration, sensor-based application development, and cross-layer optimizations addressing cloud scalability and sensor energy constraints. Based on CEB, we propose a bi-directional waterfall optimization framework under which several optimizations can be carried out to achieve greater cloud scalability and energy-efficiency of sensor devices. We also present an experimental study to determine dominant resources in the cloud which affects the logic of some of the algorithms. The experimental study lays ground for similar studies to guide the determination of critical variables for some of the proposed algorithms as well as for any new algorithm that may require knowledge of cloud's dominant resources.

双向瀑布优化架构，主要解决云扩展性及传感设备的能源限制

实验确定云中影响一些算法逻辑的主要资源

The paper is organized as follows. Related work is presented in section II followed by a brief description of the CEB architecture and an event-driven application model built on the architecture in section III. The bi-directional waterfall optimization framework is presented in section IV. In section V, we experimentally analyze dominant resources in the cloud layer in order to guide the operation of bi-directional application and data flows. Conclusion and future work are presented in section VI.

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.52

II. Related Work

The work proposed in this paper is different from [1] [2] in which powerful and scalable platforms (e.g., Hadoop) are utilized to process queries or data analytics over Big Data collected from sensor networks and stored in the platform. In our work, massive and high velocity streams of sensor data is stored in the cloud-sensor system only transiently as a result of queries and executing cloud-hosted applications. Hence, keeping and accumulating sensor data for a very long period of time and the storage and associated massive processing of the data are outside the scope of this paper. Instead, our work contributes to a scalable and optimized architecture of a middleware system connecting a massive number of sensors as services to the cloud and providing a programming model and platform for programmers to develop cloud applications that are responsive to the information from the physical world carried by the sensor data.

To enable cloud applications requiring physical sensors from outside and "beneath" the cloud, several approaches including [3] and [4] utilized virtualization as supporting technology and propose two-layer architectures that integrate physical sensors as services into the cloud. However, such two-layer architectures can lead to scalability issues affecting the availability and dependability of the cloud-sensor systems as its load (i.e., size of application and sensor) continues increasing. In this paper, we introduce, justify, and take advantage of a three-layer architecture (CEB) to enhance system scalability as well as increase energy efficiency via several novel optimizations.

Caching technology has been widely used for sensor-based computing [5] [6] in order to improve the energy efficiency and latency of the sensor network. However, the entities to cache are usually limited to sensor readings. [7] proposed a novel caching scheme 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. 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 of the sensor readings and fragments of applications can be cached at different layers of CEB in opposite directions. Compared to query shipping [8] - widely adopted in distributed database system whose stored data are relatively constant, in cloud-sensor systems sensor data are dynamic and constantly changing. Such system dynamism poses a big challenge for us 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.

III. CEB Arechitecture Overview and E-SODA Application Model

In [9], we proposed the Cloud, Edge and Beneath (CEB) which is an open architecture and framework for deploying, programming and managing sensor-based systems whose applications are hosted and run on the cloud. The architecture organizes sensor nodes and the cloud along with intermediate layers and draws on well-established and extensible standards. In addition, the CEB well supports different application models that program the smart space into different levels of sentience abstractions. In this paper, we present a specific application model that abstracts sensor data into event level, and based on such instance we propose the bi-directional waterfall optimization framework in the section IV.

A. CEB Architecture Overview

The CEB (Fig. 1) is a multi-tier architecture which we collectively refer to as the "Cloud-Edge-Beneath" where the beneath layer 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 could be a standalone server, or a computing device which connects and manages a group of sensors. The grouping of sensors could implementationally be based on type (e.g., analog, digital), functionality (e.g., weather, security), or location (e.g., within a building, community, or campus). Finally, the cloud is where sensor-based services and applications are developed, deployed and run. This three-tiered structure guarantees scalability since sensor networks operate independently, and are connected to the cloud through a scalable number of edge servers and the cloud environment itself offers a scalable infrastructure.

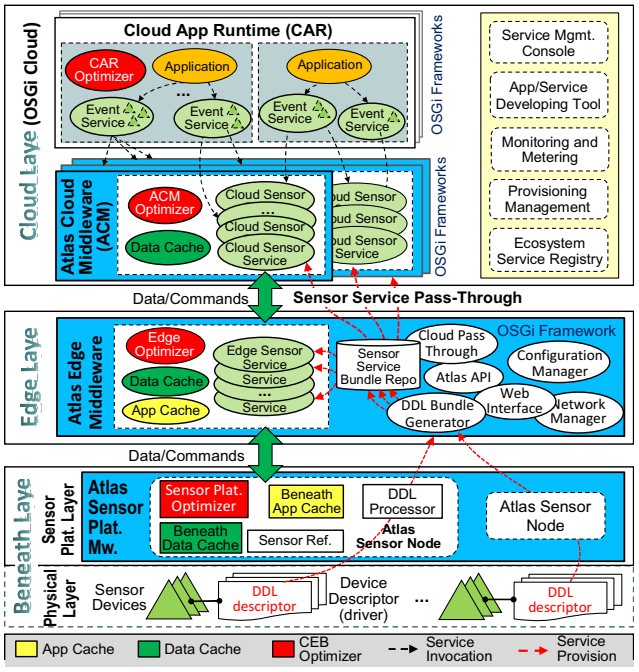

Figure 1. Overview of the CEB architecture.

CEB is built on top of Atlas [10] which is an implementation of the service-oriented device architecture (SODA) [11]. 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 layers of CEB concisely in a bottom-up order.

The beneath layer consists of the physical layer and the sensor platform layer. The former refers to the sensors and

传感器和DDL描述

their descriptions written in Device Description Language (DDL) [12] – a standard that defines succinct specifications of a wide range of devices in a human-readable language. The descriptor contains the information required for service registration, discovery and the main operations of the sensor. The sensor platform layer hosts one prong of the Atlas middleware which is responsible for sensor node configuration, initialization, data acquisition, data caching, and device control. In this layer, an Atlas node contains a DDL processor which performs pre-processing of DDL descriptors, and on power-up identifies the connected devices, and passes the corresponding DDLs to the edge layer to generate corresponding sensor service(s).

DDL预处理上电时识别连接的设备，并且将相应的DDL传递到边缘层以生成传感器服务

The edge runs the one prong of the Atlas middleware which uses OSGi [13] as its basis to provide service discovery and configuration. The middleware includes a bundle generator, which when contacted by an 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, 2) *cloud sensor service* to be uploaded to the Atlas cloud middleware in the cloud layer. The pair of sensor services communicate with each other enabling data and command passing between the edge and cloud layer.

OSGi技术是Java动态化模块化系统的一系列规范

OSGi是edge服务发现和配置的基础

bundle generator

1、edge sensor service

2、cloud sensor service

它们彼此通信使得数据和命令在edge和cloud层之间传输

The Cloud layer is built on OSGi Cloud [14] in which application is composed by loosely-coupled modules as OSGi services hosted at a distribution of cloud nodes (host OSGi framework). 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 elaborate our work in this paper, we explain two specific components (see Fig. 1) in the cloud layer most relevant to our optimization framework to be discussed:

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.

ACM充当到底层的云网关，同时，它承载了可以构建基于传感器的云应用的传感器的最基本的"云"(服务)。

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

CAR是部署和管理特定应用程序服务的容器。应用程序调用Atlas云中间件的云传感器服务，以从物理部署获取原始传感器读数。

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-sensor system. CEB supports different application models that program smart space sensors into different levels of sentience abstractions (e.g., events, activities, and phenomena). In this paper, we choose a specific application model E-SODA proposed in [9] that abstracts sensor data into events. Other sentience abstractions will be considered in the future.

优化器和缓存(application\data)包含在不同层中

B. *E-SODA Application Model*

E-SODA follows a rule-oriented paradigm in which an application is composed of 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 *representative event* which is a logical expression over sensor values. The event has a Boolean value which is evaluated to true when the event occurs otherwise to false. To provide information about an event that occurred, event parameters propagate along an *event representation tree* (ERT). These parameters are generated from the leaf (atomic events) of the ERT and propagate towards the root (composite event) of the ERT. Equation 1 shows a snippet of event composite grammars and Fig. 2 illustrates an ERT tree-based event evaluation.

ERT：事件表示树

事件的参数通过ERT自底向上传播

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

Since the working mechanism of E-SODA is rule-based, the rate at which rules are evaluated determines the traffic of sensor data within the system. We introduce an application-specific relaxation operator, the time/frequency modifier (TFM), which is intended to assist the programmer in specifying the evaluation rate of events and rules. In itself, TFM is an application-level optimization for what we call sentience-efficiency, and is specified as follows:

由于E-SODA的工作机制是基于规则的，因此评估规则的速率决定了系统内传感器数据的流量。

引入了一个应用特定的松弛算子，时间/频率修改器（TFM），旨在帮助程序员指定事件和规则的评估速率。

$$
\begin{aligned}
&TFM = < W, I_e >\\
&W = \text{nil} \mid \text{date/time} - \text{date/time} \mid \text{time} - \text{time}\\
&I_e = \text{Interval (\# of seconds) between two successive evaluation}\\
&\text{date} = \text{MM/DD/YY}\\
&\text{time} = \text{hh} : \text{mm} : \text{ss}
\end{aligned}
\tag{2}
$$

TFM是感知效率的应用程序级优化

Where W is a time window in which the affected event needs to be evaluated with interval I_e. The introduction of TFM gives programmers the capability of configuring and relaxing event evaluations based on user specific demands or sensor semantics. For example, with TFM programmers can control the frequency of evaluating an event based on expected rate of change in its sensor readings (e.g. slow evaluation for home AC temperature sensor). Another benefit of TFM is that it allows for aggregate query (piggyback) to be performed on multiple sensors connected to the same edge and applied to the same TFM. This reduces the bandwidth usage and other resources allocated for data transmissions. Fig. 3 illustrates an application of TFM to the car parking application (see Fig. 2). Initially, an event service S_0 pulls sensor readings from parking and GPS sensors to evaluate event E. Later, a TFM (I_e=10s) is applied to E_1 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.

引入TFM的优点:

1. TFM的引入使程序员能够基于用户特定需求或传感器语义来配置和放松事件评估。

例如，使用TFM，程序员可以基于其传感器读数的预期变化率（例如，家用AC温度传感器的慢速评估）来控制评估事件的频率。

For all parking sensors that are 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.

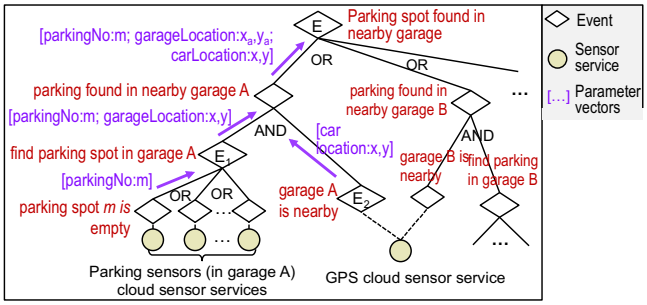

Figure 2. ERT of an example event: finding a parking spot in nearby parking garages (Parameters include the garage location, car location, and parking number).

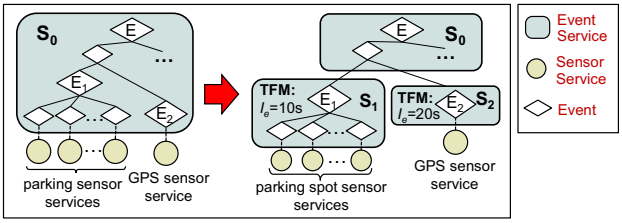

Figure 3. TFM applied to E_1 and E_2 in the smart parking application.

Based on the CEB architecture and E-SODA application model, we next present the bi-directional water optimization framework enabled by application caching strategy.

IV. Bi-directional Waterfall Optimization Framework

In plain cloud-sensor systems, applications reside in the cloud which request and process data originating from the physical layer. We extend the plain model and 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. Fig. 4 depicts an overview of the bi-directional waterfall optimization framework. 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 events. A cached event (red arrows) is evaluated at the layer it is cached to and its event value is pushed back (green arrows) to its upper layer only when it changes (i.e., selective push). For any event cached to the lower layer, a single "shadow event" is created to act as a proxy of the cached event to its consumer.

With application caching, the workload on the cloud can be dispersed or diffused across a group of edges or even sensor platforms at the beneath layer. This reduces the amount of data processing that the cloud has to perform and also lessens the amount of data that needs to be transmitted to and received by the cloud which reduces the bandwidth overhead for data transmission. Therefore, *cloud scalability* as one challenge of large-scale cloud-sensor systems (section I) can be addressed effectively. Metaphorically, the bi-directional waterfall approach allows for most interesting data to flow up and for most curious applications to propagate down. Consequently, a cloud-sensor 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. Based on this framework, a number of optimizations can be implemented to optimize the *energy consumption of the sensors* which is the second challenge in large-scale cloud-sensor systems. We have investigated the following four optimization opportunities that can be applied at different layers of the CEB (see Fig. 4):

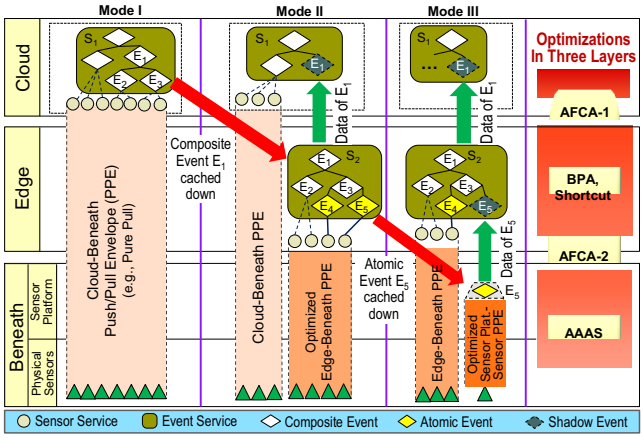

Figure 4. Bi-directional waterfall optimization framework.

1) Cloud-to-Edge Application Fragment Caching Algorithm (AFCA-1) – cloud scalability: Caching applications from the cloud to edge reduces the usage of cloud resources (e.g., processing, memory, bandwidth) and hence benefits cloud scalability. However, unlike the cloud with elastic resource supply, edge servers have limited resources. AFCA-1 selects application fragments to cache at the edge layer so as to maximize the potential benefits while staying within the limitation of the resources in edge servers. Importantly, to understand how application caching affects the cloud scale, we have to first 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 affects the logic of AFCA-1. In the paper (section V), we present an experiment study that guides the determination of such critical variables.

2) Shortcut Evaluation and Branch Permutation Algorithm (BPA) – sensor energy efficiency: 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. In order to optimize the performance of the shortcut evaluation, BPA permutes the branches of the ERT affecting the order of sensor sampling and sub-event evaluation to raise the chance of shortcutting.

双向瀑布框架，不仅是数据向上移动，应用程序片段也会向下移动并缓存在底层

仅当事件值发生改变时，才会被推回到上层（选择性上推）

shadow event

application caching的优点：
1. 负载分散，减少了云必须执行的数据处理量，并且还减少了需要发送到云并由云接收的数据量，这减少了用于数据传输的带宽开销。提高云的可扩展性。
2. 可以在任何层获得一个数据和应用的本地视图，进一步监视监视和分析应用和数据之间的交互和相互作用

将应用从云缓存到边缘减少了云资源（例如，处理，内存，带宽）的使用，因此有利于云可扩展性。然而，边缘服务器具有有限的资源。AFCA-1选择应用程序片段在边缘层缓存，以便在边缘服务器的资源限制内保持最大的潜在优势。

传感数据的子集足以确定事件的发生，可以跳过传感器采样。

为了优化快捷评估的性能，BPA对影响传感器采样和子事件评估顺序的ERT的分支进行置换，以提高shortcut的可能性。

3) Application-Aware Adaptive Sampling Algorithm (AAAS) – sensor energy efficiency: Atomic events (defined in Equation 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.

4) Edge-to-Beneath Application Fragment Caching Algorithm (AFCA-2) – sensor energy efficiency: 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.

B. 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.

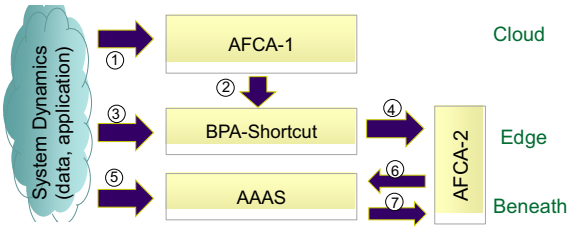

Figure 5. Interplay of the optimization algorithms.

Due to the dynamics of the cloud applications and sensor data in the cloud-sensor systems, *AFCA-1* needs to be re-evaluated periodically to adapt to any dramatic changes. Consequently, an earlier cached event may be revoked after a period of time and new events may be chosen to cache at the edge layer (shown as 1 in Fig. 5). If so, the *BPA-guided shortcut evaluation* will be executed (2) in evaluating the newly cached events. In addition, *BPA* also needs to be periodically evaluated to adapt to the changes from the sensor data (3). And if *BPA* changes the structure of ERT, *AFCA-2* is then activated to re-select the atomic events to cache at the beneath layer (4) from the updated ERT which subsequently affects the execution of *AAAS* at the beneath layer (6). At last, due to the dynamics in the sensor domain, the performance of *AAAS* may also change over time (5). Significant performance drop of the *AAAS* would be reported to the edge layer which triggers the execution of *AFCA-2* to assess if a revocation of the application caching is necessary (7). In conclusion, 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. Next we explain our implementation of such application caching within CEB.

C. Implementation of Caching Application

1) At Edge: The Atlas middleware within CEB implements a granular and modular design of service discovery and composition powered by SODA, which allows application caching to be implemented in a light-weight manner. Specifically, the machine instances in the cloud and edge layer all install the same OSGi environment (e.g., Equinox) under which services (e.g., event services and sensor services) are deployed and run. Therefore, as the major components of the cloud applications developed in E-SODA, event services make for an ideal application fragments to be cached from the cloud to the edge layer. After an event service is cached at the edge layer, it can automatically re-bind to the edge-level sensor services (peers of the cloud-level sensor services it subscribes to) via the soft service binding implemented by CEB [9] with minimum overhead. Fig. 6 illustrates the caching of an event E_2 from the cloud to the edge layer. S_0 is an event service of a cloud application and event E_2 is a sub-event of S_0's representative event E_1. To evaluate the value of E_2, S_0 invokes the local event services S_3, S_4 and S_5 which initially subscribe to the cloud sensor services for sensor a, b and c hosted remotely at the ACM. Later the cloud decides to cache event E_2 to the edge layer by sending the service bundles of S_3-S_5 to the edge (stored at the bundle repository in the Edge Atlas Middleware). Then, edge installs these bundles to its Atlas middleware. Upon activating service S_3-S_5, they can automatically subscribe to the edge sensor service for sensor a, b and c without specific configuration. In addition, for any event (e.g., E_2) cached to the edge layer, a "shadow service" of the cached event is correspondingly created in the ACM acting as a proxy of the cached event (e.g., E_2) to its consumers (e.g., E_1). It should be noted that caching application fragments from the cloud to the edge is light-weight as it does not require specific reconfiguration of its service providers nor consumers. This is critical to our bi-directional waterfall approach because it allows an application fragment cached at the edge to be later revoked with low overhead, which is a pre-condition for adaptive optimization capable of adapting to the system's dynamics.

2) At Beneath: The application fragments cached at the beneath layer are atomic events. With the help of the component model [9] adopted by the E-SODA for service composition, event composition formulas are separated from

the service's underlying processing mechanism. Specifically, they are described in an .xml file (i.e., component descriptor) as service metadata. Therefore, these composition fomulas of atomic event (application fragment) can be easily extracted from its service object at runtime and loaded onto the on-node processing unit at the sensor platform that serves as the application cache at the beneath.

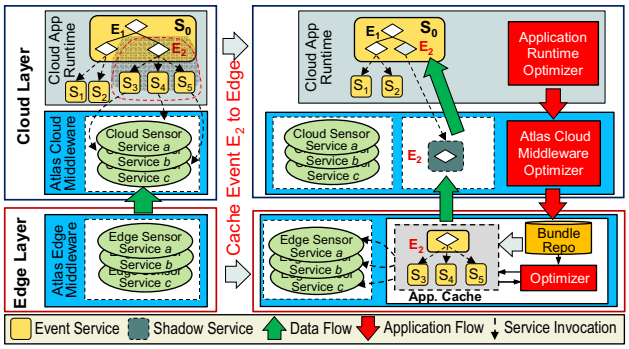

Figure 6. Implementation of application (event) caching.

In the following section, we present an experimental study to determine the dominant resources in the cloud components which are affects by application caching. The experimental study lays ground for studies to guide the determination of critical variables for AFCA-1 which we published in [17] as well as for any new algorithm that may require knowledge of cloud's dominant resources.

V. Determining Dominant Resources in Cloud Components Affected by Application Caching

Of all the components in the cloud layer (see Fig. 1), caching applications to the edge layer primarily affects the cloud resource usage in two components: 1) cloud application runtime (CAR) where the applications and event services are hosted; and 2) Atlas cloud middleware (ACM) which hosts the cloud sensor services and shadow services that request sensor data on behalf of services in CAR. Before we investigate each cloud component to learn their dominant resources. We first we set up a prototype of CEB architecture as experimental testbed on which smart home sensor based applications (based on E-SODA application model) 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.

A. An Experimental Cloud-Sensor System Setup

We adopted Amazon Web Service to implement the cloud layer of CEB which prunes components not relevant to the experiment leaving only ACM, CAR, and a provisioner for cloud instances and cloud services. Every cloud instance is provisioned with an AMI that uses an install of Equinox 3.6.2 as OSGi container and is stored on Amazon Simple Storage Service ready to be instantiated by EC2. In our design, all instances provisioned for ACM and CAR have the same resource capacity including CPU, memory and others. Specifically, every cloud node is instantiated on a standard machine type *m1.medium* whose resource capacity is shown in Table I. The provisioner monitors the usage of CPU, memory and other resources for every cloud instance at runtime and is configured to increase or decrease the number of cloud instances when pre-specified auto-scaling threshold is breached for a period of time (known as contingency time). In this paper, for all cloud instances, we adopt the same maximum memory utilization and CPU utilization denoted as γ_1 and γ_2 respectively.

We emulated the execution of edge servers which all have constant processing capability (equal to one ECU) and 1.7GB memory. Since AFCA-1 focuses on the interactions between the cloud and edge layer only, we replaced the sensors at the beneath layer with emulators.

TABLE I. Cloud Instance Resources (ECU stands for EC2 Compute Unit providing the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 1.7 GHz 2007 Xeon processor [15])

ECU	Memory	Storage	Bandwidth
2	3.75 GB	160 GB	–

B. A Cloud-Sensor Data/Application Benchmark

In E-SODA programming model, sensor data are abstracted into "events" which make up the core modules (i.e., event services) of the cloud applications. Therefore, creating application benchmark is in fact equivalent to creating event sets. Specifically, to prepare a data/application benchmark for large-scale cloud-sensor systems, the sensor data and events must satisfy the following requirements.

- Large size and diversity of sensor devices. The set of sensors should reach at least a small city scale with different types.
- Diversity of sensor data. Readings from sensors exhibit different characteristics, for example, in data type (double, integer, binary), variation, and tolerance to time fidelity (i.e., time-to-live).
- Large size and diversity of events. The event set should be big enough and exhibit different dynamism, abstractions (environment, activities, and phenomenon), and levels of complexities.

Based on above requirements, we constructed the data/application benchmark for smart home cloud-sensor system with a scale of 2000 houses in which a variety of applications (emergency-detection, security, activity recognition, and healthcare) are created based on a huge set of household and resident-worn sensors. Although this is still not a large city-scale, through the experiments we could clearly see how the cloud behaved and scaled as we increased cloud applications and sensors deployments.

The data/application benchmark is based on the PLCouple1 dataset collected from the PlaceLab [16] which is a live-in laboratory in Cambridge, MA. The dataset contains sensor data from 178 wired and wireless sensors for a 2.5 month period when a couple stayed in the PlaceLab. Sensors including location, humidity, switch, pressure, light, temperature, gas, current and flow sensor are installed in the house. To benchmark the event sets, we manually created 200 events based on the household sensors which basically fall into following categories: activity recognition, emergency detection, security, and healthcare. The same set of events is extended to every synthesized smart home in our experiment. As for the synthesis of sensor data (the dataset), we developed a synthetic data generator that takes the

2000 houses
4种种类应用
1、emergency -detection
2、security
3、activity recognition
4、healthcare
共200个事件

传感器类型: household resident-worn
剑桥大学的PlaceLab的PLCouple 1数据集：一对夫妇在PlaceLab呆了2.5个月
178个传感器所采集的数据

学习事件的特性，将其作为参数，生成用于其他smart home的传感器数据

同时修改了一些事件的概率，比如火灾警报、房子入侵等，以方便测试

PlaceLab dataset and our created event set as inputs and learn several characteristics of these events in the event set such as probability of event occurrence and event dynamics. Then these event characteristics are used as parameters to generate sensor data for all other synthesized smart homes (2000 in the experiment). In addition, to support irregular events whose probability of occurrence is too low to capture in the PlaceLab (e.g., house intrusion, fire alarm), we modified the probability of some events based on empirical data to allow these events to happen in the larger number of the synthesized houses.

Based on the experimental testbed and our synthesized data/application benchmark, we investigate the dominant resources for the cloud components – ACM and CAR respectively. To do so, we first theoretically analyze the potential dominant resources and then conduct groups of experiments to validate our analysis.

C. *Dominant Resource in ACM*

Analyzing resource usage in ACM is relatively simple because it hosts mostly the cloud sensor services (see Fig. 1) which are very simple and similar in their sizes and functionalities – no more than data transporters between cloud applications and edges with minimum data processing.

For memory resource, most of its consumption comes from the deployment of cloud sensor services. In our implementation, one cloud sensor service bundle, after being deployed and activated, takes 6.01 KB. Given the maximum memory utilization allocated for dynamic usage by a cloud instance is γ_1, the approximate number of cloud sensor services that can be supported in one cloud instance is $(\gamma_1 \times 3.75GB)/6.01KB = 6.24\ \gamma_1 \times 10^5$.

For processing resource (quota of 2 ECUs), most of its consumption is attributed to the processing of the cloud sensor services of which we reasonably consider only the data transmitting and receiving as the major contributions while ignoring others (e.g., data processing and reading/writing cache). We use cc_s and cc_r to represent the number of CPU cycles per packet send and receive respectively. Assume that the overall rate at which all the cloud applications requests data from sensor s is f which also becomes the rate at which the cloud sensor service for s sends data to the cloud applications when pull is adopted. We also use f' to denote the rate at which the cloud sensor service for s delivers the data request for s (i.e., cache miss at the cloud data cache) to the edge layer. If we assume, for all the sensors, the average for each of f and f' to be $\overline{f}$ and $\overline{f'}$, the maximal number of cloud sensor services that an ACM can hold would be: $(\gamma_2 \times 2 \times 1.2G) / [(\overline{f} + \overline{f'}) \times (cc_r + cc_s)]$.

CPU周期数

For bandwidth, typical cloud provider (e.g., AWS) does not define an exact bandwidth capacity configured for its cloud instances. Therefore, we measure the bandwidth resources experimentally. We consider only the bandwidth capacity of the cloud-side channel which aggregates all traffics from and towards the edge servers while ignoring the capacity of the edge-side channel which carries traffic from and to a single edge. This is because unlike the cloud which auto-scales to the fluctuating network load, the edge-in practice-is assumed to be configured to scale to the specific deployment in its sensors' jurisdiction. In addition, considering the large number of edges connecting to a cloud instance, the traffic bottleneck would more likely be the cloud-side.

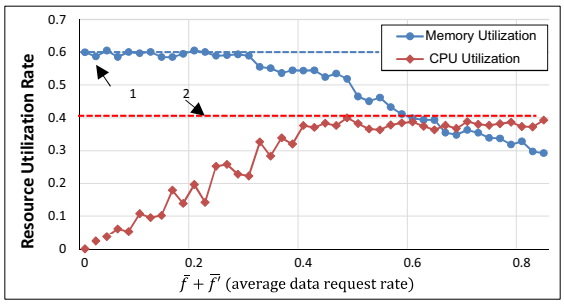

Figure 7. Effect of average data request rate on ACM dominant resource.

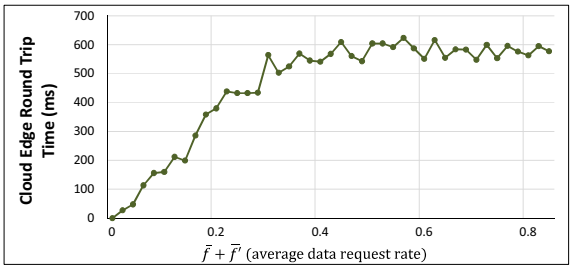

Figure 8. Round trip time between the cloud and edges.

Based on the above analysis, for the processing and memory resources, we can predicate that γ_1, γ_2 and $\overline{f} + \overline{f'}$ affect which one dominates the other. To validate our assumption, we conduct experiments in which a constant number of sensors are simulated. In the experiment, (γ_1, γ_2) are set to be (0.6, 0.4) and the value of ($\overline{f} + \overline{f'}$) varies to simulate the variation of the scale of the cloud-sensor system. From the experiment results shown in Fig. 7, with increasing $\overline{f} + \overline{f'}$ the dominant resource of the ACM changes from memory to CPU, which validates our prediction. In addition, we measured the network performance by measuring the round trip time (RTT) between cloud and edge and show the results in Fig. 8. Our implementation adopts TCP/IPv4 in which every packet has 64 bytes. In Fig. 8, the network delay first increases as the data request frequency increases and eventually stops at ~600ms due to performing cloud auto-scaling (decreasing the number of sensor services per cloud instance). Since maximum RTT is still considered acceptable, we exclude the bandwidth from being ACM's dominant resource.

Through the theoretical analyses and experiment validations, we conclude that, processing and memory are the only two resources that could determine the dimension of the ACM. And which one plays the dominant role depends on the data request rate ($\overline{f}$, $\overline{f'}$) and the auto-scaling threshold (γ_1, γ_2).

D. Dominant Resource in CAR

Compared to CARs, ACMs are busier transmitting packets because they play as hubs aggregating and passing requests/data between the cloud applications in CARs and the edges. On the contrary, CARs are more processing intensive than ACMs (i.e., cost more CPU and memory). Therefore, for CAR the bandwidth cost accounts for a smaller portion of its total resources usage than it does for ACM. In addition, CARs communicate with ACMs (i.e., in-cloud instances) which are allocated much higher bandwidth capacity than outbound interfaces. Consequently, like ACM we exclude bandwidth from being a dominant resource of the CAR. As for memory and processing, unlike ACM which hosts primarily single type services, CAR host services with different types and sizes (i.e., atomic event service, composite event service, etc.). Therefore, it is impossible to theoretically measure and compare the memory and processing resources usage. Thus we again conducted experiments to help find the dominant resources for CAR. In the experiments, we deployed a constant number of events in CAR and changed the frequency of event evaluation (i.e., the rate of event processing and data transmission). The experiment results are shown in Fig. 9 through which we observe that either the processing or memory can determine the dimension of CAR, and the activeness of the cloud applications (i.e., event evaluation frequency) can determine which one dominate the other.

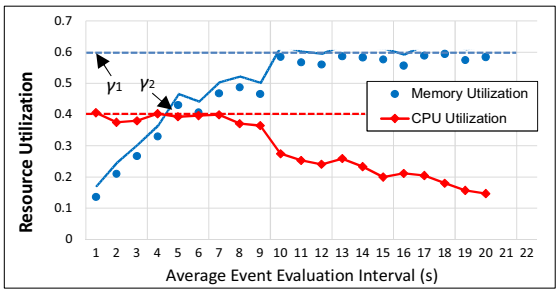

Figure 9. Effect of event evaluation interval on CAR dominant resource.

In conclusion, for both ACM and CAR, the dominant resources that determine their dimension can be either processing or memory. Additionally, via experiments, we find out the influence factors that could determine which resource dominates the other. Based on this conclusion, we propose the AFCA-1 algorithm in [17].

VI. Conclusion

We briefly presented the overarching architecture - CEB as an optimization enabling platform for the large-scale cloud-sensor systems. Based on CEB, we adopted a particular event-driven application model and proposed the bi-directional data/application caching optimization framework. We presented an overview of the optimization algorithms that can be performed under the optimization framework and explained their interactions. In future work, we will give the details for each algorithm. At last, we present an example of an experimental study to determine dominant resources in the cloud which affects the logic of some of the algorithms. The experimental study lays ground for similar studies to guide the determination of critical variables for some of the proposed algorithms as well as for any new algorithm that may require knowledge of cloud's dominant resources.

References

[1] C. Jardak, J. Riihijarvi, F. Oldewurtel, and P. Mahonen, "Parallel processing of data from very large-scale wireless sensor networks," In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, June 20-25, 2010, Chicago.

[2] B. Yu, A. Cuzzocrea, D. Jeong, and S. Maydebura, "A Bigtable/MapReduce based cloud infrastructure for effectively and efficiently managing large-scale sensor networks," In Proceedings of the 5th International Conference on Data Management in Cloud, Grid and P2P Systems. Berlin, 2012.

[3] K. Lee, and D. Hughes, "System architecture directions for tangible cloud computing," In International Workshop on Information Security and Applications 2010, China.

[4] M. Yuriyama, and T. Kushida, "Sensor-Cloud infrastructure physical sensor management with virtualized sensors on cloud computing," In Proceedings of the 13th International Conference on Network-Based Information Systems 2010.

[5] K. Prabh and T. Abdelzaher, "Energy-conserving data cache placement in sensor networks". ACM Transactions on Sensor Networks 2005.

[6] M. A. Rahman and S. Hussain, "Effective caching in wireless sensor network," In Proceedings of the Advanced Information Networking and Applications 2007.

[7] L. Ying, Z. Liu, D. Towsley, and C. H. Xia, "Distributed operator placement and data caching in large-scale sensor networks," In Proceedings of IEEE INFOCOM 2008, Phoenix, AZ.

[8] D. Kossmann, "The state of the art in distributed query processing," In ACM Computer Surveys, 2000.

[9] Y. Xu, and A. Helal, "Scalable cloud-sensor architecture for the Internet of Things," Submitted for publication. UF Mobile and Pervasive Computing Lab Technical Report: http://www.icta.ufl.edu/projects/publications/ceb.pdf

[10] J. King, R. Bose, H. Yang, S. Pickles, and A. Helal, "Atlas - a service-oriented sensor platform: hardware and middleware to enable programmable pervasive spaces," In Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications. Tampa, Florida, November 2006, 630-638.

[11] S. Deugd, R. Carroll, K.E. Kelly, B. Millett, and J. Ricker, "SODA: service oriented device architecture," IEEE Pervasive Computing, Volume 5 Issue 3, July 2006.

[12] C. Chen and A. Helal. 2009. "Device integration in SODA using the device description language," In Proceedings of the IEEE/IPSJ Symposium on Applications and the Internet, July 2009, Seattle, Washington, USA.

[13] OSGi 4.2 Specification. http://www.osgi.org/Download/Release4V42

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

[15] Amazon EC2, http://aws.amazon.com

[16] PlaceLab, http://architecture.mit.edu/house_n/data/PlaceLab/PlaceLab.htm

[17] Y. Xu and A. 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 (MSWIM), September 21–26, 2014, Montreal, Canada, in press.