國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

行動裝置感測器中介軟體之最佳化研究
Optimizing Mobile Middleware for Coordinated Sensor Activations

侯婷方
Ting-Fang Hou

共同指導教授：徐正炘 博士, 金仲達 博士
Advisors: Cheng-Hsin Hsu, Ph.D., Chung-Ta King, Ph.D.

中華民國 103 年 06 月
June, 2014

國立清華大學
資訊工程研究所

碩士論文

行動裝置感測器中介軟體之最佳化研究

侯婷方 撰

103
06

中文摘要

隨著手機技術的發展，手機上配備有更多的感測器。這些感測器的資訊被廣泛的使用和開發在情境感知的應用程式(Context-Aware applications)。利用感測器資訊推論外在環境狀況或使用者的活動情形。目前的手機系統未提供整合性的感測器排程，這些情境感知的應用程式會各自獨立操作感測器的開關及資料的讀取，導致不必要的電量消耗。在本篇論文中，我們強調有效的整合應用程式並找出最佳的感測器使用方法。對於單一手機，我們提出一個middleware介於應用程式和手機硬體之間，用以溝通應用程式並規劃和控制感測器的開關。目前手機被廣泛的使用於日常生活中，我們更進一步的讓感測器排程整合更多手機上或是基礎設備中的感測器，並將此想法應用到crowdsensing系統中。 首先，我們對單一手機設計、實作和分析一個middleware，此middleware 權衡感測器的電量消耗和情境感知的精準度，找出最佳的感測器使用方法。我們將問題分成兩種並用數學式子表示: (1) 滿足應用程式的要求，最小化電量消耗和 (2)在有限的電量下，最大化情境感知的精準度。我們分別提出兩個最佳化演算法和快速的演算法並用Java開發模擬器。從實驗結果表示，快速的演算法可以即時的解決問題、節省電量消耗和最佳化演算法平均只有~ 3%的差距。我們將演算實做到Android系統上並成功節省電量的消耗。 在論文的第二部分，我們推廣感測器排程的概念到多隻手機上並將其應用到crowdsensing 系統中。我們設計一個crowdsensing系統，系統依照手機使用者的位址和能力(ex: 剩餘電量)，找出最佳的工作分配方式以降低碳排放的量。我們用數學式子表示問題並提出兩個最佳化/快速的演算法。利用Java開發的模擬器所得到的結果表示，快速的演算法可以減少364倍的碳排放量、加速工作完成(8倍)和最佳化演算法只有~ 2%的差距。

Abstract

Existing context-aware mobile applications directly control sensors in the mobile devices in an uncoordinated and non-optimized manner, which leads to redundant sensor activations and energy waste. Optimal and coordinated sensor usage dictates a comprehensive mobile middleware solution with sensor scheduling on single device to bring together the information from all applications/sensors and intelligently select the best set of sensors to activate. While the widespread use of smartphones, we cooperate the sensors on multiple smartphones and infrastructure sensors to build a novel crowdsensing system.

In Chap. 3, we design, implement, and evaluate a novel green sensor management middleware for single device that rigorously makes tradeoffs between energy consumption of sensors and accuracy of inferred contexts. The problem is formulated rigorously as mathematical optimization problems that (i) minimize the total energy consumption while achieving the required accuracy and (ii) maximize the overall accuracy under a given energy budget. Two optimal algorithms for these two optimization problems are proposed, which provide the performance bounds. As they may lead to prohibitively long running time, two efficient heuristic algorithms are then presented, which run in real-time. Extensive trace-driven simulations are conducted using traces from real Android users to evaluate the performance of the proposed middleware and algorithms. The simulation results indicate that the heuristic algorithms: (i) always terminate in real-time, (ii) result in small optimization gap of up to $\sim 2\%$, and (iii) lead to better performance for larger problems. We also implement and evaluate the proposed middleware and algorithms on real Android smartphones, showing their practicality and efficiency.

For the extension, we consider the sensor scheduling on multiple smartphones and infrastructure sensors in Chap. 4. We apply the extensive consideration to crowdsensing system. We present a Smartphone Augmented Infrastructure Sensing (SAIS) system that offers better situation awareness to officials and civilians for minimizing the amount of generated carbon dioxide. The SAIS system minimizes the carbon footprint by solving the task assignment problem. We mathematically formulate the problems and optimally solve it using optimization problem solvers, and we also proposed an efficient task assignment algorithm (ETA) for lower running time. Our trace-driven simulations show the results of our efficient algorithm: (i) saves up to 364 times in carbon footprint, (ii) outperforms by up to 8 times in responding time, and (iii) achieves a small optimization gap of $\sim 2\%$.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Increasingly more mobile applications (apps) leverage the rich set of sensors on the smartphones to infer their contexts for enhancing user experiences [9]. A market research predicts that context-aware apps will affect 96 billion USD of consumer spending in 2015 [14], and several companies, such as Google and Qualcomm, have already provided libraries [10, 15] for context-aware apps. As an example, Nike+ Running [34], shown in Fig. 1.1, is a context-aware app using GPS, accelerometer, and other sensors to record the jogging routes. In the future, multiple apps running at the same time on a smartphone may request a multitude of overlapping contexts, e.g., location and time. While a context may be answered by different sets of sensors depending on the requested accuracy and availability of sensors, uncoordinated and non-optimized use of the sensors by the multiple apps may turn on redundant sensors, leading to waste of energy. For example, in Fig. 1.2, while the context required by My ACT [32] app can be inferred by accelerometers, if a navigation app (such as GoogleMap [16]) is already running on the smartphone with GPS turned on, the same context *should* be inferred simply by GPS without the overhead of activating accelerometers and processing the readings. This illustrative example demonstrates the opportunity of being *green* by using a centralized sensor scheduling to selectively activate energy-efficient sensors. While the smartphones become more popular, the smartphones are equipped by all people in daily life and come with many sensors. This change allows us to consider the centralized sensor scheduling for reducing the energy consumption to control the sensors on multiple smartphones. Moreover, we cooperate and share the sensory data from multiple smartphones and infrastructure sensors, such as surveillances and thermometers as a *crowdsensing* system.

Figure 1.1: Context-aware mobile apps (http://nike.com).

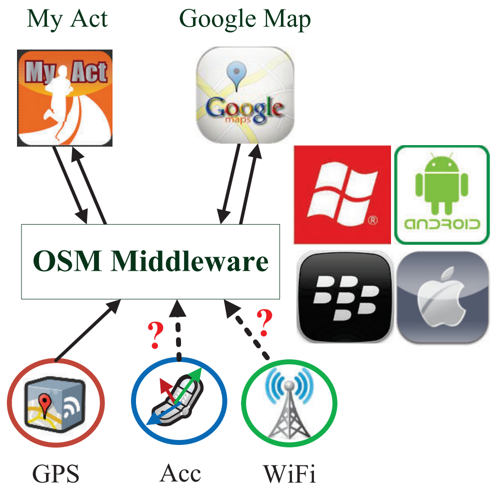


Figure 1.2: Opportunity for saving energy by selectively activating sensors.

1.1 Sensor Scheduling for Single Device

We consider the sensor scheduling for sing device. Choosing the *best* set of sensors to activate in order to satisfy the needs of various context-aware apps is very challenging. This is because there exists a tradeoff between context inference accuracy and energy consumption of sensors. On top of that, context-aware apps impose diverse accuracy requirements and smartphones have different remaining battery levels at different time. Therefore, efficiently determining the set of sensors to activate *dictates* a comprehensive mobile middleware solution, which brings together various information from apps and sensors. In this thesis, we propose *Optimal Sensor Management (OSM)* middleware, which sits between the context-aware apps and sensors. OSM achieves *coordinated* and *optimized* uses of sensors, and provides efficient sensor management service to the context-aware apps. To adapt to heterogeneous smartphones and diverse app/user preferences, the OSM middleware also provides APIs for context-aware apps to send user feedback on inferred contexts.

The core of the OSM middleware is the *sensor scheduling* algorithm, which is repeatedly invoked to adapt to system dynamics. We develop two mathematical formulations of the scheduling problems: (i) energy optimization, which strives to find the set of sensors that consumes the least energy while satisfying the sensing requirements, and (ii) accuracy optimization, which strives to maximize the overall accuracy under an energy budget. The OSM middleware allows the user to pick the preferred optimization criteria, which may also be heuristically determined, e.g., the middleware may minimize energy consumption only when the battery level is less than 33%. We propose two optimal scheduling algorithms, which give us the performance bounds. However, running the optimal algorithms for large scheduling problems may lead to high running time. Therefore, we also develop two heuristic, real-time scheduling algorithms for resource-constrained mobile devices. Moreover, we extend our algorithms to leverage the relationship between the request frequency (how often an app needs each context), sensor sampling rate (how often the sensor readings are collected), and energy consumption. Doing so gives us even more rooms for optimization, e.g., a more (less) accurate set of sensors may lead to higher (lower) accuracy, and thus can be invoked less (more) often. By changing the frequency and sampling rates, we essentially *cache* the higher-layer contexts, which have much lower space requirements than caching the low-level sensor readings. The caching mechanism is not possible without our detailed accuracy degradation model.

We evaluate the performance of the proposed scheduling algorithms using trace-driven simulations. The simulation results show that, compared to ordinary mobile OS's, such as Android, our proposed algorithms save sensing energy or increase accuracy by 41% and 72% on average. Moreover, our heuristic algorithms achieve close-to-optimal performance in terms of energy saving and inference accuracy (as close as a $\sim 3\%$ gap is observed), yet terminate in real-time. We also implement the proposed middleware and heuristic algorithms on Android and conduct real experiments to demonstrate their practicality and efficiency. The experiment results show that our middleware achieves high accuracy, while mobile users opt for minimizing energy consumption may prolong their battery life by 2 times. Furthermore, our Android implementation runs fast (< 50 ms running time of the scheduling algorithms) and scales well to larger problems. Last, our measurement results depict that the energy overhead of our middleware is fairly small: between 1% and 8%.

1.2 Sensor Scheduling for Multiple Devices

The popularity of smartphones changes the landscape of sensing systems which do not just consider a single smartphones. These smartphones allow us to augment infrastructure sensing by coorporating *crowdsensing* for cost reduction. Crowdsensing [13] refers to collect sensory data from many smartphones and can be further classified into *opportunistic sensing* and *participatory sensing*. The difference between the two sensing modes is the user intervention level, as illustrated in Fig. 1.3. Opportunistic sensing collects sensory data from smartphones without user intervention, while participatory sensing requires users to actively contribute the sensory data, such as shooting and uploading photos.

We present a Smartphone Augmented Infrastructure Sensing (SAIS) system, which aims to *minimize the total carbon footprint while satisfying all the queries*. To the best of

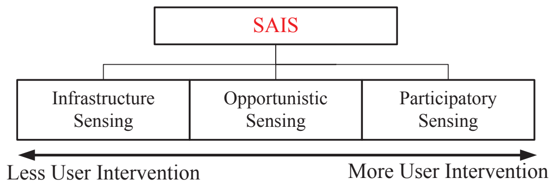


Figure 1.3: SAIS supports sensing modes with diverse human intervention levels.

our knowledge, using smartphone users to augment the infrastructure sensors by covering wider areas in on-demand manner, so as to reduce the carbon footprint, has not been thoroughly studied in the literature. In the current work, we assume that all smartphone users are both *producers* and *consumers* of information about events. That is, via a unified *dashboard* mobile application, officials and civilians can *help* one another to create better situation-awareness (i.e., they all get to ask/answer questions). For example, a smartphone user at a fair may query for the most popular booth at some points, who later on may offer traffic conditions to others when leaving the fairground. Such naturally-grown ad-hoc communities motivate smartphone users to participate. The core problem of SAIS system is (i) sensor scheduling and (ii) worker selection. The worker contains smartphone users and infrastructure sensors which can detect events. We consider the tradeoff between accuracy and the cost of carbon footprint, and jointly formulate and solve these two problems which focus on the selection of the sensors and *workers* as a task assignment problem. We propose an optimal algorithm which gives us the performance bounds and an efficient algorithm which lead a lower running time.

We evaluate a trace-driven simulation with our proposed algorithms. The simulation results show that our algorithms save the cost of carbon footprint up to 364 times and improve response time up to 8 times compared to baseline algorithms. Moreover, the efficient algorithm achieves a small optimization gap of $\sim 2\%$ with the optimal algorithm.

1.3 Contributions of Thesis

This thesis makes several contributions.

- We present the OSM middleware to selectively activate some energy-efficient sensors while satisfying the context sensing requirements from many context-aware apps. We rigorously study the sensor scheduling problem, which is the core issue in the OSM middleware. We present two optimization problem formulations: energy minimization and accuracy maximization. We propose four optimal/efficient algorithms to solve the two formulations.
- We extend our algorithms to solve the more general problems with heterogeneous

context frequencies, sensor sampling rates, and accuracy degradation models. The extended algorithms realize efficient caching of inferred contexts for lower energy consumption.

- We extensively evaluate the proposed middleware and algorithms in trace-driven simulations and real experiments. Our proof-of-concept Android implementation demonstrates the practicality and efficiency of our solution, while the simulations exercise more system parameters and larger problems.
- For the extension, we present the SAIS system as a *crowdsensing* which a novel system considers sensors on multiple smartphones and infrastructure sensors. We present a task scheduling problem. We propose two optimal/efficient algorithms, and evaluate the algorithms via trace-driven simulation.

Chapter 2

Related Work

2.1 Sensor Scheduling

Mobile context sensing has been studied in the literature. Most existing studies [7, 22, 27, 30, 43] consider location sensing. Ma et al. [30] propose a system to predict the future locations of a mobile user based on his/her previous locations. Their prediction algorithm employs sensor readings from GSM and WiFi for coarse localization, which is more energy efficient than using GPS sensors. Similarly, Yan et al. [22] propose to combine movement detection and path tracking for future location predictions to reduce the sampling rate of turning on GPS sensors. Chon et al. [7] present a location service considering the tradeoff between accuracy and energy consumption. Their system uses Bayesian networks to model accuracy and energy consumption, and Hidden Markov Model to predict users' locations. Lin et al. [27] also propose a system to minimize the energy consumption of context sensing under predicted user mobility patterns. Wang et al. [43] rely on extensive training sessions to quantify the tradeoff between accuracy and energy consumption, assuming various sensors are used. Different from our proposed OSM middleware, these studies [7, 22, 27, 30, 43] only consider location sensing, and thus their solutions are inapplicable to our problem.

Contexts other than location have also been recently investigated [29, 39, 47]. Yan et al. [47] employ accelerometers to classify the mobile user actions, e.g., stand, walk, and sit. They adjust the sampling rate of reading and processing accelerometer readings to tradeoff the energy consumption and classification accuracy. Schirmer and Hopfner [39] propose two algorithms to reduce the usage of energy hungry sensors on smartphones. The first algorithm picks the most energy efficient way to infer each context. The second algorithm leverages the relationships among the sensors to choose certain sensors to turn on. Lu et al. [29] study the tradeoff between energy consumption and sensing accuracy of three contexts: audio, action, and location. None of the studies [29, 39, 47] consider the inter-dependency among inference algorithms of different contexts: each context is inferred independently. Therefore, their solutions can not capitalize certain optimization opportunity, e.g., from the motivative example illustrated in Fig. 1.2.

Nath [33] leverages the inter-dependency between various contexts by using a conjecture algorithm. That is, his system dynamically learns the relationships among contexts, and tries to satisfy each context requirement by running the conjecture algorithm on the recently inferred contexts stored in a cache. Different from our work, Nath [33] does not address the scheduling problem of concurrent and overlapping requirements from multiple apps, nor does it utilize the fact that the sets of sensors used to infer different contexts may *not* be mutually disjoint. Kang et al. [21] also takes the inter-dependency among various contexts into consideration. However, their solution does not consider inference accuracy and thus may not work well for apps with high accuracy requirements. In contrast, our proposed OSM middleware rigorously models: (i) energy consumption of each sensor and (ii) accuracy of inferring every context using different sets of sensors. The resulting models are used to find the best set of sensors to activate under system-wide resource constraints. In this sense, our work is complementary to Nath [33] and Kang et al. [21].

Pervasive computing community has proposed various context-aware middleware solutions, although they are mostly for PCs and servers [4, 20, 37]. In contrast, we propose a middleware specifically designed for smartphones, which are battery-powered with stringent energy constraints. Moreover, each smartphone hosts a large number of apps requesting for diverse contexts with different accuracy requirements, and we concentrate on how to tradeoff inference accuracy and energy consumption.

Last, C. Lin [26] considered simpler versions of the sensor scheduling for a single smartphone. In his thesis, he proposed two simpler heuristic algorithms and most focused on the evaluation of middleware on android system. Different from his work, we mathematically formulated the objective problems and solved by two optimal algorithms. We also proposed two efficient algorithms and jointly considered the caching problem and frequency of apps requirements by controlling the sampling rates of sensors. Moreover, we apply the sensor scheduling to more sensors on multiple mobile device and infrastructure sensors.

2.2 Crowdsensing

Infrastructure sensing has been studied in the literature [8, 19]. Chuvieco and Congalton [8] presented a system to generate fire hazard map using images. Hsieh et al. [19] proposed a system for accurate vehicle tracking and classification using highway cameras. To ease the deployment overhead, crowdsensing using smartphones has also been considered in more recent work [6, 17, 40, 49]. Talasila et al. [40] employed smartphone sensors and human inputs to increase the accuracy of location sensors. Hasenfratz et al. [17] used MiCS-OZ-47 sensors and GPS readers on smartphones to monitor air quality. Zhou et al. [49] used mobile sensors to identify the bus routes and predict the bus arrival times. Chon et al. [6] used radio fingerprints of nearby WiFi access points to infer the current places, e.g., gyms and restaurants.

Certain participatory sensing systems pushed the user intervention level to an extreme by requesting smartphone users to manually perform sensing tasks or even move to specific sensing locations [5, 23, 45] for small rewards, which are also referred to as crowdsourcing systems [48]. Chon et al. [5] showed that a few smartphones can cover a wide sensing area. However, they did not consider the energy consumption of each mobile device. Lane et al. [23] found that the sensing and transmission energy is high when smartphones are idle. They therefore developed a prediction algorithm to determine the best sensing and transmission times. Yan et al. [45] leveraged human efforts for validating the image search results and studied the trade-off between accuracy and human validation time. Different from our work, these studies [5, 23, 45] did not take in-situ sensors into considerations.

Coric and Gruteser [11] leveraged the vehicle and infrastructure sensors to provide on-street vacant parking map. Their work is different from ours, because energy efficiency is not a concern in their system, while minimizing carbon footprint is the core of our work. More specifically, we dynamically choose the sensors from in-situ and smartphone sensors to minimize the energy consumption and ensure the sensing coverage. This problem has not been considered in the previous work. Last, our earlier studies [25] considered simpler versions of the worker selection for a single smartphone.

Chapter 3

Sensor Scheduling for Single Mobile Device

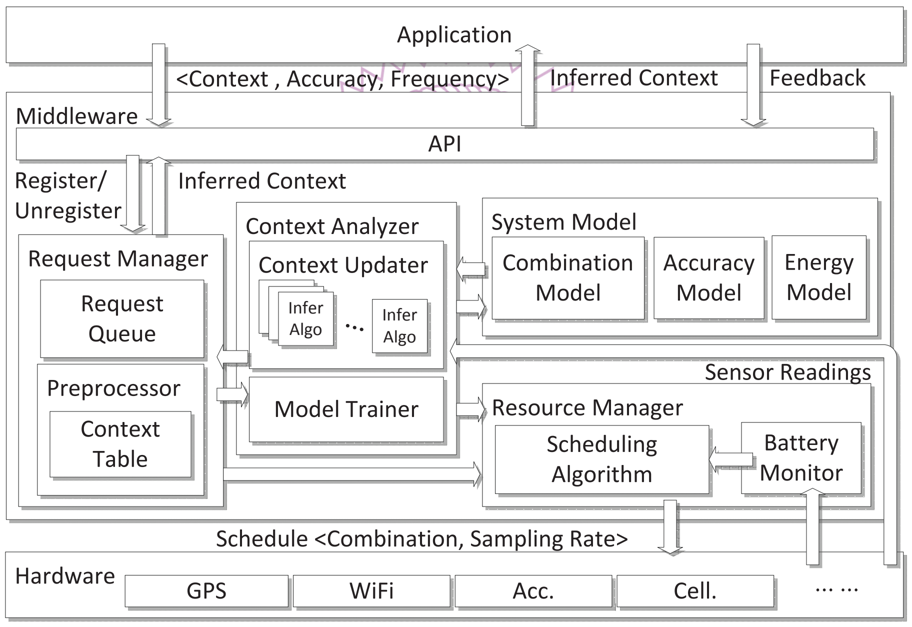


Figure 3.1: The proposed OSM middleware.

3.1 Framework :OSM

We present the architecture of the OSM in this section.

3.1.1 System Overview

Our proposed OSM middleware sits between apps and the hardware. Many context-aware apps run on smartphones, which may need different contexts at diverse accuracy and frequency. We collectively call a pair of accuracy and frequency as *request* throughout this chapter. Apps may *register* or *unregister* requests through an Application Programming Interface (API) at any time. The OSM middleware maintains a database of active requests, and determines what sensors to activate, and at what sampling rates, so as to satisfy all active requests from the apps. Making such decisions is not an easy task, because multiple sets of apps may register for different requests, and each request can be satisfied by using one or multiple sensors. Each set of sensors is referred to as a *combination* in this chapter. For example, a context `IsDriving` may be inferred by a combination of the GPS and the accelerometer. Moreover, a context may be inferred by various combinations, which renders the decisions even harder. For instance, `IsDriving` may also be inferred using the microphone.

3.1.2 System Architecture

As illustrated in Fig. 3.1, the OSM middleware consists of an API and four software components: (i) request manager, (ii) resource manager, (iii) context analyzer, and (iv) system model. These components support the scheduling algorithms by managing requests, monitoring battery level, inferring contexts, and modeling sensing accuracy. Our OSM middleware is modularized in the sense that each component can be enhanced without changing the architecture.

API. The OSM middleware provides APIs for context-aware apps. In particular, each app uses `register()`/`unregister()` to add/remove requests to/from a database managed by the request manager. Each `register()` also specifies a *callback* function for the request manager to call. The request manager calls the callback functions of requested apps to pass them the inferred contexts. The apps then act differently based on the contexts. We support both the periodic and event-driven modes, indicated by the frequency (zero for the event-driven mode); the default mode is periodic. In the unfortunate cases, where the inferred contexts are incorrect, the apps may call `feedback()` to notify the OSM middleware, which then updates some model parameters for dynamic adaptations.

Request manager. The request manager manages a request queue to keep track of all registered requests and apps. It also checks if the callback function invocation fails, and automatically unregisters all the requests from any failed (exited) apps. A preprocessor is used to aggregate the overlapping requests, which are stored in the context table. By aggregation, we refer to the process of finding the request with the highest accuracy requirement and the highest frequency for each context. Once we satisfy that request, all other requests are satisfied. Last, the request manager is also responsible for admission control, which rejects new requests when resources, such as battery level, are saturated.

The precise design of the admission control algorithm is our future work.

Resource manager. The resource manager focuses on resource conservation and consists of two components: the battery monitor and scheduling algorithm. The battery monitor continuously checks the battery in terms of voltage, current, and capacity. The scheduling algorithm takes the context table and system models as inputs, and generates schedules that specify the combinations of sensors to activate and their sampling rates. The scheduling algorithm can be configured to either: (i) maximize the overall accuracy under a given energy budget which is an input from the battery monitor, or (ii) minimize the total energy consumption while achieving target accuracy levels which are inputs from apps or users. That is, we support two optimization *criteria*, and the target energy budget and required accuracy are chosen by users or determined by heuristics. For instance, smartphones may maximize the overall accuracy only when the battery level is higher than 33%.

Context analyzer. The context analyzer analyzes the sensor readings to infer contexts, and comprises two components: (i) context updater, which hosts various inference algorithms for different combinations and contexts, and (ii) model trainer, which takes the feedback from apps as inputs and updates the model parameters in the system model.

System model. It contains three parts: (i) context model, (ii) accuracy model, and (iii) energy model. The context model stores the relationship among contexts, inference algorithms, and sensor combinations, e.g., the action inference algorithm uses the accelerometer and WiFi to classify the user actions, such as walk, run, and still. The accuracy model captures the accuracy of the contexts inferred by the inference algorithms. Diverse metrics, such as *precision* and *recall* can be used to quantify the inference accuracy. Precision is defined as the ratio between the number of correctly inferred contexts and the total number of contexts reported by the inference algorithms. Recall is defined as the ratio between the number of correctly inferred contexts and the actual number of the contexts according to the ground truth. In the rest of this chapter, we use precision as the accuracy metric given that a fixed recall is enforced for concrete discussions; we use accuracy when presenting general ideas. The energy model captures the energy consumption of each sensor at different sampling rates. Moreover, the energy model may be readily extended to also capture the energy consumption of the inference algorithms. These models may be parameterized, while the parameters are dynamically adjusted by the model trainer using measurement results.

The core component of the OSM middleware is the scheduling algorithm, which generates the best schedules specifying sensor combinations and their sampling rates. The schedules are sent to the sensors via mobile OS's. For example, sensors in Android smartphones are managed by the sensor manager, location manager, and WiFi manager. In the next two sections, we study the design of optimal/efficient scheduling algorithms.

3.2 Sensor Scheduling Problem Formulations

We first present two formulations, in which each request/schedule has a predefined and fixed frequency/sampling rate for tractability. We will relax this assumption in Sec. 3.3.4.

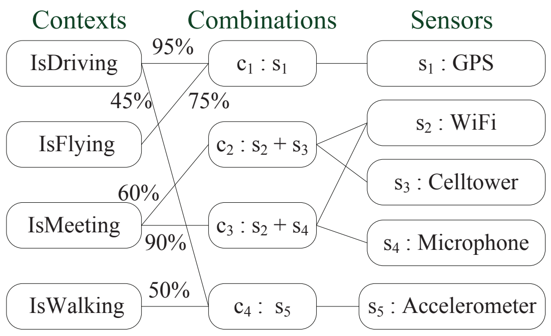


Figure 3.2: Relation among contexts, combinations, and sensors.

3.2.1 Notations

We first develop notations for the scheduling problem in the OSM middleware. Table 3.1 summarizes the symbols used throughout this paper. We let R be the total number of requested contexts to be scheduled and S be the total number of sensors on the smartphone. We define a request as $< y_r, f_r >$, where r ($1 \leq r \leq R$) is the requested context, y_r is the target accuracy, and f_r is the desired frequency. If not otherwise specified, we assume f_r is fixed for each context. We let C be the total number of potential sensor combinations. In theory, we could end up with $C = 2^S$, but in the reality, $C \ll 2^S$ as many combinations cannot be used to infer any context. We employ a boolean matrix $\mathbf{M}$ to capture the relation between combinations and sensors[1]. In particular, we let $m_{c,s} = 1$ ($1 \leq c \leq C$, $1 \leq s \leq S$) if combination c contains sensor s, and $m_{c,s} = 0$ otherwise. Fig. 3.2 illustrates the relationship among sensors, combinations, and contexts. Take this figure as example, we can select c_1 or c_4 with different sensors to infer `IsDriving` at different accuracy levels. Moreover, a combination may be used to infer multiple contexts at diverse accuracy levels. Specifically, for context r, combination c achieves an accuracy of $a_{c,r}$, where $1 \leq r \leq R$ and $1 \leq c \leq C$. We collectively call all $a_{c,r}$ as $\mathbf{A}$. Last, we use e_s to denote the energy consumption of sensor s, where $1 \leq s \leq S$ in the next scheduling window T. T is a system parameter, in the order of minutes. We write a schedule as $< x_s, p_s >$, where x_s indicates whether the sensor s ($1 \leq s \leq S$) should be activated, and

[1]Throughout this chapter, we use bold font to denote vectors or matrices.

p_s represents the sampling rate. Particularly, we let $x_s = 1$ if the sensor s should be turned on, and $x_s = 0$ otherwise. For now, we assume the sampling rate p_s is predetermined and fixed for each combination and sensor if not otherwise specified.

Next, we present the two scheduling problems and show their hardness.

Problem 1 (Energy Minimization: EM)**.** *Given requested contexts r ($1 \leq r \leq R$) and combinations c ($1 \leq c \leq C$), the EM problem selects a subset of combinations to achieve the minimum energy consumption while satisfying all the accuracy requirements y_r ($1 \leq r \leq R$). Upon the combination subset is chosen, the schedule is set based on the relation between combinations and sensors ($\mathbf{M}$).*

Remark 1 (Hardness of the EM Problem)**.** *The EM scheduling problem is NP-Complete.*

Proof. We prove the hardness of the EM scheduling problem by reducing the NP-Complete set cover problem to it in polynomial time. We first present the set cover problem. Given a set of U elements and a collection of sets $\mathbf{V} = \{V_z | 1 \leq z \leq Z\}$, where V_z (for all z) is a subset of the U elements, find the smallest $\mathbf{V}^* \subseteq \mathbf{V}$ so that $\bigcup_{V_z \in \mathbf{V}^*} V_z$ covers all U elements.

For any set cover problem, we create a corresponding EM problem as follows. We create a requested context r for each element u, and let $R = U$. The accuracy requirements y_r are set to 100% for all r. We create a combination c for each subset in $\mathbf{V}$, and let $C = Z$. We then set $a_{c,r} = 100\%$ if the subset corresponding to c contains the element corresponding to r, and $a_{c,r} = 0$ otherwise. Last, we choose the sensor energy consumption e_s ($1 \leq s \leq S$) and the combination/sensor relation $\mathbf{M}$ in a way that all combinations consume unit energy. Because all combinations consume the same amount of energy, solving the EM scheduling problem is essentially finding the minimum number of combinations to satisfy all requested contexts, and the results can be readily mapped back to minimize the number of subsets from $\mathbf{V}$ to cover all U elements. In addition, because: (i) validating a solution can also be done in polynomial time and (ii) the set cover problem is NP-Complete, the EM scheduling problem is NP-Complete. □

Problem 2 (Accuracy Maximization: AM)**.** *Given requested contexts r ($1 \leq r \leq R$), combinations c ($1 \leq c \leq C$), and an energy budget E, the AM problem selects a subset of combinations to maximize the achieved accuracy without exceeding the energy budget. Upon the combination subset is chosen, the schedule is set based on the relation between combinations and sensors ($\mathbf{M}$).*

Remark 2 (Hardness of the AM Problem)**.** *The AM scheduling problem is NP-Complete.*

Proof. We prove the hardness of the AM scheduling problem by reducing the NP-Complete 0/1 knapsack problem to it in polynomial time. We first present the 0/1 knapsack problem. Given a set of elements u ($1 \leq u \leq U$) with weight w_u and value v_u, find a set

$\mathbf{U}^*$ of elements so that $\bigcup_{u \in \mathbf{U}^*} v_u$ is maximized while $\bigcup_{u \in \mathbf{U}^*} w_u \leq Z$, where Z is the bag capacity.

For any 0/1 knapsack problem, we create a corresponding AM problem as follows. We create $C = U$ combinations and $R = U$ requested contexts, where $a_{c,r} = v_c$ if $c = r$ and $a_{c,r} = 0$ otherwise. We choose the sensor energy consumption e_s ($1 \leq s \leq S$) and the combination/sensor relation ($\mathbf{M}$) in a way that combination c consumes w_u energy. Last, we let energy budget $E = Z$. Solving the resulting AM scheduling problem is essentially finding the set of combinations to maximize the total accuracy without exceeding the energy budget, and the answers can be readily mapped back to maximizing the weight sum of chosen elements without exceeding the bag capacity. In addition, because: (i) validating a solution can be done in polynomial time and (ii) the 0/1 knapsack problem is NP-Complete, we show that the AM scheduling problem is NP-Complete. $\square$

Table 3.1: Symbol Table

Symbol	**Definition**
r	Requested contexts
y_r	Accuracy requirement of r
f_r	Frequency of r
S	Total number of sensors
R	Total number of requested contexts
C	Total number of combinations
$m_{c,s}$	Boolean relation between c and s
$\mathbf{M}$	Collection of all $m_{c,s}$
$a_{c,r}$	Accuracy of using c to infer r
$\mathbf{A}$	Collection of all $a_{c,r}$
e_s	Energy consumption of sensor s
x_s	Whether to activate sensor s
p_s	Sampling rate of sensor s
T	Scheduling window
E	Energy budget

3.2.2 Problem Formulations

With the notations defined above, we formulate the EM scheduling problem as:

$$\min \sum_{s=1}^{S} e_s x_s \tag{3.1}$$

$$s.t. \max_{c=1}^{C} \left\{ \left\lfloor \frac{\sum_{s=1}^{S} m_{c,s} x_s}{\sum_{s=1}^{S} m_{c,s}} \right\rfloor a_{c,r} \right\} \geq y_r, \forall r = 1, 2, \dots, R; \tag{3.2}$$

$$x_s \in \{0, 1\}, \forall s = 1, 2, \dots, S. \tag{3.3}$$

The objective function in Eq. (3.1) is to minimize the total energy consumption in each scheduling window. The constraints in Eq. (3.2) ensure that the resulting schedule satisfies all the accuracy requirements.

Next, we formulate the AM scheduling problem as:

$$\max \sum_{r=1}^{R} \max_{c=1}^{C} \left\{ \left\lfloor \frac{\sum_{s=1}^{S} m_{c,s} x_s}{\sum_{s=1}^{S} m_{c,s}} \right\rfloor a_{c,r} \right\} \tag{3.4}$$

$$s.t. \sum_{s=1}^{S} e_s x_s \leq E; \tag{3.5}$$

$$x_s \in \{0, 1\}, \forall s = 1, 2, \dots, S. \tag{3.6}$$

The objective function in Eq. (3.4) is to maximize the overall accuracy. The constraint in Eq. (3.5) ensures the total energy consumption in each scheduling window does not exceed the energy budget.

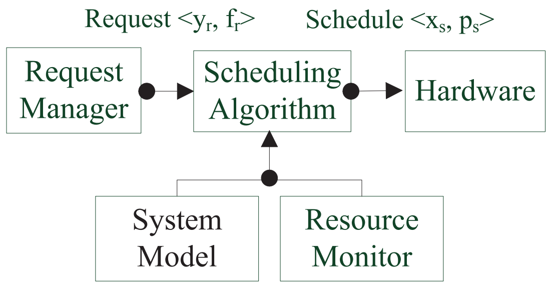


Figure 3.3: Input/output of the scheduling algorithm.

3.3 Sensor Scheduling Algorithms

We present and analyze four scheduling algorithms in this section. As illustrated in Fig. 3.3, the scheduling algorithms take inputs from Request Manager, solve the scheduling problems with helps from System Model and Resource Monitor, and send the resulting schedules to Hardware.

3.3.1 Optimal Sensor Scheduling Algorithms (EMA/AMA)

The formulations in Eqs. (3.1)–(3.3) and Eqs. (3.4)–(3.6) are both Integer Programming (IP) problems, which may be solved by commercial optimization solvers. We adopt CPLEX [12] optimizer to develop two optimization algorithms. In Eq. (3.2), we use the floor function to check whether the combinations are feasible. However, CPLEX solver does not support the floor function. To cope with this limitation, we define intermediate variables $\hat{z}_c$ $(1 \leq c \leq C)$ to represent the *floor* function values of z_c $(1 \leq c \leq C)$. For the EM formulations in Eqs. (3.1)–(3.3), we add the following two constraints:

$$z_c \geq \hat{z}_c; \tag{3.7}$$

$$\hat{z}_c + 1 \geq z_c. \tag{3.8}$$

Next, we let $\frac{\sum_{s=1}^{S} m_{c,s} x_s}{\sum_{s=1}^{S} m_{c,s}} = \hat{z}_c$, and rewrite the Eq. (3.2) as:

$$\max_{c=1}^{C}(z_c a_{c,r}) \geq y_r, \forall r = 1, 2, \ldots, R. \tag{3.9}$$

We solve the updated EM formulation, and refer to the optimal algorithm as *Energy Minimization Algorithm (EMA)*.

For the AM formulation in Eqs. (3.4)–(3.6), we also add Eqs. (3.7)–(3.8) and rewrite Eq. (3.4) as:

$$\max \sum_{r=1}^{R} \max_{c=1}^{C}(z_c a_{c,r}). \tag{3.10}$$

We solve the updated AM formulation, and refer to the optimal algorithm as *Accuracy Maximization Algorithm (AMA)*. While the EMA and AMA algorithms lead to optimal schedules in terms of energy consumption and inference accuracy, they may suffer from long running time and thus are not suitable for larger scheduling problems with more contexts, combinations, or sensors. Hence, we develop two heuristic algorithms in the next sections.

```
1: // Input: $\mathbf{A}, \mathbf{M}, \hat{\mathbf{R}}$; Output: $\mathbf{X}$
2: let $\mathbf{X} = \varnothing$
3: while $\hat{\mathbf{R}} \neq \varnothing, \max_{1 \leq c \leq C} g_c(\mathbf{A}, \mathbf{M}, \mathbf{X}, \hat{\mathbf{R}}) > 0$ do
4:     for each $c = 1, 2, \ldots, C$ do
5:         compute $g_c(\mathbf{A}, \mathbf{M}, \mathbf{X}, \hat{\mathbf{R}})$ using Eq. (3.13)
6:     select $c^* = \operatorname{argmax}_{c=1,2,\ldots,C} g_c(\mathbf{A}, \mathbf{M}, \mathbf{X}, \hat{\mathbf{R}})$
7:     $\mathbf{X} \leftarrow \mathbf{X} \cup \{s | m_{c^*,s} = 1\}$
8:     $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} - \{r | y_r \leq a_{c^*,r}\}$
```

Figure 3.4: Efficient Energy Minimization Algorithm (EEMA).

3.3.2 Efficient Energy Minimization Algorithm (EEMA)

The Efficient Energy Minimization Algorithm (EEMA) maintains a set $\hat{\mathbf{R}}$, $\mathbf{X}$ of unmet requests and the chosen sensors so far, and iteratively selects combination that leads to the highest *utility* value. We define utility of a combination as a fraction of *profit* and *cost*. The profit is the number of unmet requests that can be satisfied by the combination, and the cost is the additional energy consumption, if the combination is chosen. The algorithm stops after either: (i) all the requests are met or (ii) no more combinations can be chosen. Mathematically, we define the profit p_c of combination c ($1 \leq c \leq C$) as a function of $\mathbf{A}$ and $\hat{\mathbf{R}}$:

$$p_c(\mathbf{A}, \hat{\mathbf{R}}) = \frac{\sum_{1 \leq r \leq R, r \in \hat{\mathbf{R}}} \mathbf{1}_{[a_{c,r} \geq y_r]}}{\sum_{1 \leq r \leq R} \mathbf{1}_{[a_{c,r} \geq y_r]}}, \tag{3.11}$$

where $\mathbf{1}$ is the indicator function, $\mathbf{A}$ is the accuracy model, and $\hat{\mathbf{R}}$ is the set of unmet requests. We define the cost w_c of a combination c ($1 \leq c \leq C$) as a function of $\mathbf{M}, \mathbf{X}$:

$$w_c(\mathbf{M}, \mathbf{X}) = \sum_{1 \leq s \leq S, s \notin \mathbf{X}} m_{c,s} e_s, \tag{3.12}$$

where $\mathbf{M}$ is the boolean matrix of relation between combinations and sensors and $\mathbf{X}$ keeps track of the chosen sensors so far. Last, the utility $g_c(\mathbf{A}, \mathbf{M}, \mathbf{X}, \hat{\mathbf{R}})$ of a combination c ($1 \leq c \leq C$) is written as:

$$g_c(\mathbf{A}, \mathbf{M}, \mathbf{X}, \hat{\mathbf{R}}) = \frac{p_c(\mathbf{A}, \hat{\mathbf{R}})}{w_c(\mathbf{M}, \mathbf{X})}. \tag{3.13}$$

After all iterations, $\mathbf{X}$ is the resulting schedule. We note that the denominator of Eq. (3.13), $w_c(\mathbf{M}, \mathbf{X})$, could be zero because some sensors (such as proximity sensor) may be always on for basic smartphone features, and thus incur no additional energy consumption. To avoid dividing by zero, we replace the denominator of Eq. (3.13) with $\max(w_c(\mathbf{M}, \mathbf{X}), 1)$, where 1 is a small non-zero number.

Fig. 3.4 gives the pseudocode of EEMA. The while loop in lines 3–8 iteratively selects the sensors of the combination with the highest utility. More specifically, the loop in lines 4–5 computes the latest $g_c(\mathbf{A}, \mathbf{M}, \mathbf{X}, \hat{\mathbf{R}})$ using Eq. (3.13) for all combinations. Line 6 picks the combination c^* with the highest utility. Lines 7 and 8 update the current schedule and unmet requests.

The EEMA algorithm is inspired by an approximation algorithm of the weighted set cover problem [35]. However, our EM problem is even more complex in the sense that each combination comprises multiple sensors, which have diverse energy consumption. Moreover, multiple combinations may use overlapping sensors. Hence, the EEMA algorithm has to update utility function values in each iteration. This prevents us from proving an approximation factor for the EEMA algorithm.

Remark 3 (Complexity of EEMA)**.** *The time complexity of EEMA is $O(RC(S+R))$. R and C come from the loops starting from lines 3 and 4, respectively. S and R come from computing $w_c(\mathbf{M},\mathbf{X})$ and $p_c(\mathbf{A},\hat{\mathbf{R}})$, respectively. Lines 6, 7, and 8 dominate. Hence, the time complexity is $O(RC(S+R))$.*

```
1:  // Input: $\mathbf{A}$, $\mathbf{M}$; Output: $\mathbf{X}$
2:  let $e_{\mathbf{X}} = 0$, $\mathbf{W} = \{1, 2, \ldots, C\}$, $\hat{\mathbf{Y}} = 0$
3:  while $\mathbf{W} \neq \varnothing$ do
4:      for each $c = 1, 2, \ldots, C$ do
5:          compute $g'_c(\mathbf{A},\mathbf{M},\mathbf{X},\hat{\mathbf{Y}})$ with Eq. (3.15)
6:      select $c^* = \text{argmax}_{c\in\mathbf{W}}\, g'_c(\mathbf{A},\mathbf{M},\mathbf{X},\hat{\mathbf{Y}})$
7:      $\mathbf{W} \leftarrow \mathbf{W} - \{c^*\}$
8:      if $e_{\mathbf{X}} + w_{c^*}(\mathbf{M},\mathbf{X}) < E$ then
9:          $\mathbf{X} \leftarrow \mathbf{X} \cup \{s | m_{c^*,s} = 1\}$
10:         $e_{\mathbf{X}} = \sum_{s=1}^{S} e_s x_s$
11:         for each $r = 1, 2, \ldots, R$ do
12:             if $\hat{y}_r \leq a_{c^*,r}$ then
13:                 $\hat{y}_r = a_{c^*,r}$
```

Figure 3.5: Efficient Accuracy Maximization Algorithm (EAMA).

3.3.3 Efficient Accuracy Maximization Algorithm (EAMA)

The Efficient Accuracy Maximization Algorithm (EAMA) algorithm is similar to the EEMA algorithm. The EAMA algorithm maintains a set of $\hat{\mathbf{R}}$ of unmet requests, a list of available combinations $\mathbf{W}$ (i.e., those have not been selected), the energy consumption $e_{\mathbf{X}}$ and achieved accuracy $\hat{\mathbf{Y}}_{\mathbf{X}}$ with the current schedule $\mathbf{X}$. Its goal is to find a schedule $\mathbf{X}$ with the highest average accuracy without exceeding the energy budget E. While the cost function $w_c(\mathbf{M},\mathbf{X})$ is the same as the one used in the EEMA algorithm, we define a different profit function as:

$$p'_c(\mathbf{A},\mathbf{X},\hat{\mathbf{Y}}) = \sum_{1\leq r\leq R} a_{c,r}\mathbf{1}_{[a_{c,r}\geq \max(y_r,\hat{y}_r(\mathbf{X}))]}, \tag{3.14}$$

where $\mathbf{1}$ is the indicator function, and $\hat{\mathbf{Y}}(\mathbf{X}) = \{\hat{y}_r(x) | r = 1, 2, \ldots, R\}$ is the achieved accuracy with schedule $\mathbf{X}$. Moreover, the utility function $g'_c(\mathbf{A},\mathbf{M},\mathbf{X},\hat{\mathbf{Y}})$ is written as:

$$g'_c(\mathbf{A},\mathbf{M},\mathbf{X},\hat{\mathbf{Y}}) = \frac{p'_c(\mathbf{A},\mathbf{X},\hat{\mathbf{Y}})}{\max(w_c(\mathbf{M},\mathbf{X}),1)}. \tag{3.15}$$

Using the utility function, the EAMA algorithm iterates through all combinations in $\mathbf{W}$. In each iteration, the algorithm selects the combination with the highest utility function value yet satisfying the energy constraint. The algorithm stops once all the combinations have been considered.

Fig. 3.5 gives the pseudocode of EAMA. The while loop in lines 3–13 iteratively selects the sensors of the combination with the highest utility value. In particular, the loop in lines 4–5 computes the latest $g'_c(\mathbf{A}, \mathbf{M}, \mathbf{X}, \hat{\mathbf{Y}})$ using Eq. (3.15) for all combinations. Line 6 picks the combination c^* with the highest utility, and line 7 updates the available combinations. The if-clause between lines 8–13 checks if activating the sensors of combination c^* would lead to energy consumption within the energy budget. If yes, lines 9, 10 and the loop starting from line 11 update schedule $\mathbf{X}$, total energy consumption $e_{\mathbf{X}}$, and the achieved accuracy $\hat{\mathbf{Y}}$, respectively.

The EAMA algorithm is inspired by an approximation algorithm of the 0/1 knapsack problem [38]. However, the considered AM problem is more complex in the sense that the utility function depends on the current schedule $\mathbf{X}$ and the current achieved accuracy $\hat{\mathbf{Y}}$. This prevents us from providing an approximation factor for the EAMA algorithm.

Remark 4 (Complexity of EAMA)**.** *The time complexity of EAMA is $O(C^2(S + R))$. C^2 comes from the loops starting from lines 3 and 4. S and R come from computing $w_c(\mathbf{M}, \mathbf{X})$ and $p'_c(\mathbf{A}, \mathbf{X}, \hat{\mathbf{Y}})$, respectively. Lines 8–13 dominate. This yields the time complexity of $O(C^2(S + R))$.*

3.3.4 Heterogeneous Frequency and Sampling Rate

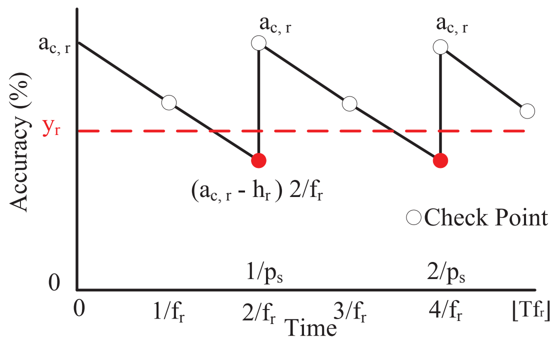


Figure 3.6: The dynamics of accuracy degradation.

We relax the assumption of fixed frequency/sampling rate in this section. So far, the proposed scheduling algorithms assume that request frequencies $f_s = 1/T$ are homogeneous and sampling rates p_s are pre-determined by individual inference algorithms. Such assumption is reasonable if the smartphone users are in relatively stable environments. However, when smartphone users are in more dynamic environments, the inference accuracy of each context will *degrade* over time within a scheduling window. Take location

Table 3.2: The Combinations, Contexts, and Sensors Used in Our Simulations

	Context						Sensor					
Combination	IsSitting	IsStanding	IsWalking	IsRunning	InMeeting	IsDriving	Acc.	Blue.	WiFi	Mic.	GPS	Cell.
YAN [47]	95	91	83.8	0	73.86	74	1	0	1	0	1	0
CenceMe [31]	68	78	94	74	68	74	1	1	0	1	1	0
EEMSS [43]	89.44	0	78.2	90	0	63.86	1	1	1	0	1	0
EEMSS2 [43]	99.44	0	88.2	100	0	73.86	1	1	1	1	1	0
SAMMPLE [46]	0	0	0	0	68	0	1	0	0	0	1	0
SAMMPLE2 [46]	0	0	0	0	57	0	1	0	0	0	0	0
OTHER1	50	59	66	70	96	91	0	0	1	0	0	1
OTHER2	35	54	56	60	86	76	1	0	0	0	0	1

as example, suppose a mobile user moves at a constant speed, the location accuracy decreases along the time, while higher speed results in faster rate of accuracy degradation.

To model the accuracy degradation, we let h_r be the degradation rate of context r, where $r = 1, 2, \ldots, R$. At the beginning of a scheduling window, the accuracy of inferred context r is $a_{c,r}$ (assuming combination c is used to infer r). Since the apps requesting for r get the inferred context at time 0 (related to the scheduling window), apps need another update at $0 + \frac{1}{f_r}$, where f_r is the frequency of r in Hz. If $a_{c,r} - \frac{h_r}{f_r} \geq y_r$, the OSM middleware passes the previously inferred context to the apps; otherwise, the OSM middleware infers the context again, i.e., the sampling rate p_s of each sensor s with $m_{c,s} = 1$ is larger than f_r. Therefore, we refer to $0+\frac{1}{f_r}$ as a *check point*. Fig. 3.6 illustrates the dynamics of inference accuracy.

We generalize the above observation as follows. For any context r ($r = 1, 2, \ldots, R$), there are $\lfloor T f_r \rfloor$ check points, and our goal is to set the sampling rate p_s of relevant sensor s to be the same as the first check point with the degraded accuracy lower than the requested accuracy. Mathematically, we write:

$$p_s = \left[\max\left(\left\lceil \frac{(a_{c,r} - y_r)/h_r}{1/f_r} \right\rceil \frac{1}{f_r}, T \right) \right]^{-1}. \tag{3.16}$$

In the ceiling function, the numerator computes the time duration before the accuracy degrades below the required accuracy, and the denominator is the time difference between two check points. Therefore, the ceiling function gives the number of check points an inferred context can last. Multiply the ceiling function by $1/f_r$ again gives a lower bound on the time interval between two context inferences, which should never exceed the scheduling window T (as captured by the $\max$ function). Upon going through all context r, where $r = 1, 2, \ldots, R$, we may end up with several p_s for each sensor s, i.e., s may be activated by multiple combinations in order to satisfy all the requests. We pick the smallest p_s among all combinations, which yield the heuristic algorithms for heterogeneous frequency and sampling rate.

3.4 Trace-Driven Simulations for Single Device

In Secs. 3.4 and 3.5, we fix the target recall to be 80% and use the resulting precision as the accuracy metric for concrete discussions. We interchangeably use accuracy and precision throughout these two sections.

3.4.1 Setup

We have developed a Java-based event-driven simulator to evaluate the proposed OSM middleware. We have also implemented the proposed scheduling algorithms: the EMA and AMA for optimal scheduling, and the EEMA and EAMA for efficient scheduling. The EMA and AMA algorithms utilize CPLEX [12] solver with a time limit of 1 min, which is quite long for real-time scheduling. For comparisons, we have also implemented a baseline algorithm, which goes through all the requests, and for each context, it selects the combination achieving the highest precision.

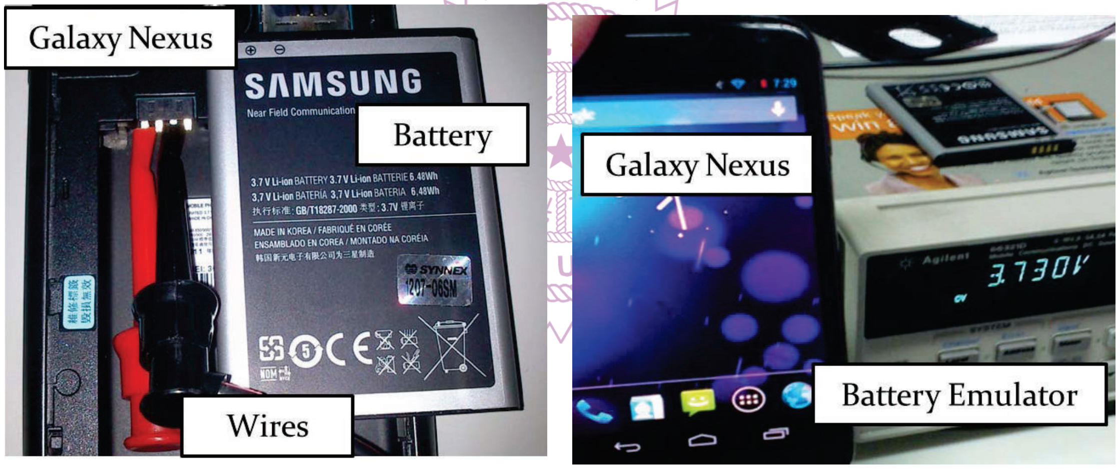


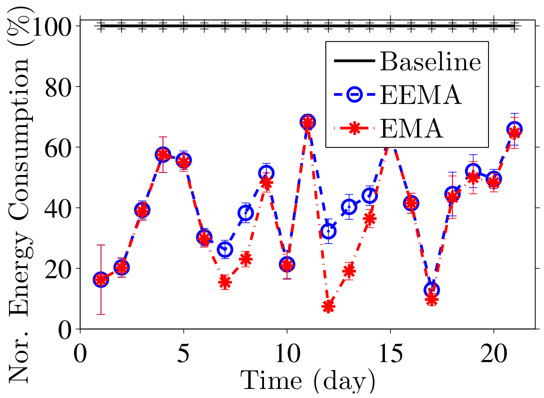
Figure 3.7: Setup for energy consumption measurements: (a) the smartphone and (b) the battery emulator.

We collect app usage traces from 5 Android users (engineers and students between 25 and 35 years old) for 21 days. In particular, we develop an Android app, which saves the *activity stack* once every 5 mins. The Android activity stack includes a list of running apps, and can be converted into a sequence of app launch/exit events. We assume that 10% of random apps are context-aware. Moreover, each app requests for a context randomly selected from the 6 contexts listed in Table 3.2. The same table also gives the precision reported in the literature [31, 43, 46, 47]. The resulting app traces are used to drive our simulator. Fig. 3.7 shows our energy consumption measurement setup. We take out the battery from the smartphone, and connect the Agilent 66321D battery emulator [2] to the smartphone. The battery emulator is also connected to a PC via USB, and we record the voltage and current at 200 Hz for 5 mins. We write an app to turn on/off individual sensors, and compute the power consumption difference with/without sensor activations. We repeat the measurement 10 times and give the average power consumption in Table 3.3, where is used in our simulations.

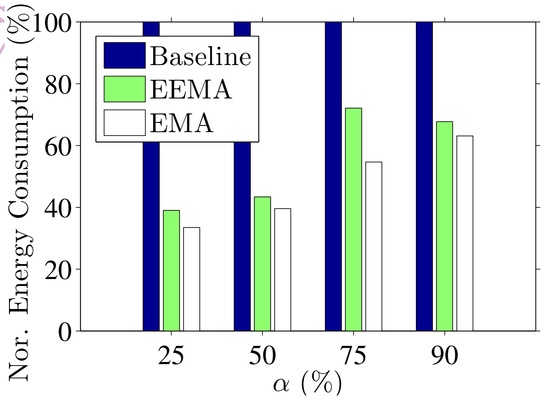
Table 3.3: Sensor Power Consumption of a Samsung Smartphone

Sensor	GPS	WiFi	Cell	Acc.	Mic.	Blue.
Power (mW)	546	16	20	187	62	195

We conduct the simulations on a Linux PC with an Intel 3.4 GHz CPU. We consider both the EM and AM problems. For the EM problem, we let α be the minimal precision of individual requests. More specifically, each request is associated with a random precision uniformly distributed in $[\alpha, 100\%]$. We consider $\alpha \in \{25\%, 50\%, 75\%, 90\%\}$, and report results from $\alpha = 25\%$ if not otherwise specified. For the AM problem, we consider the energy budget $E = \{45, 52.5, 60, 67.5, 75\}$ J, with a sampling rate of $1/300$ Hz and duty cycle of 1 min. E is the energy limit in each scheduling window. We report sample results from $E = 60$ J, if not otherwise specified. We use $T = 5$ mins as scheduling window size. We adopt three performance metrics: (i) energy consumption in J/day, (ii) mean precision in $\%$, and (iii) running time in ms. We report the simulation results with 95% confidence intervals whenever applicable.



(a)



(b)

Figure 3.8: Average energy consumption of the EM problem: (a) with $\alpha = 25\%$ and (b) diverse α.

We note that the number of contexts in Table 3.2 is rather limited. Therefore, we also develop a synthetic trace generator to generate traces with more contexts and combinations, which mimic the future scenarios where context-aware apps become very popular. In particular, we consider 12 sensors, and vary the number of contexts $\beta \in \{2, 5, 10, 15, 20\}$ and the number of combinations $\gamma = \{2, 5, 10, 15, 20\}$. We report the results from $\beta = 15$

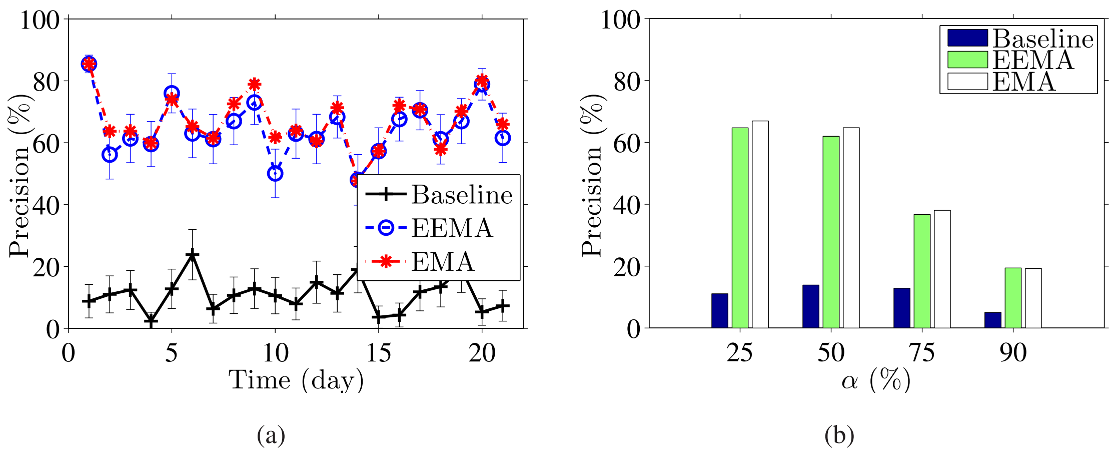


Figure 3.9: Average precision of the EM problem: (a) with $\alpha = 25\%$ and (b) diverse α.

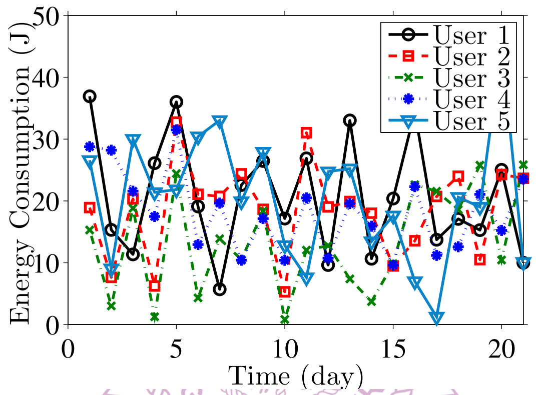
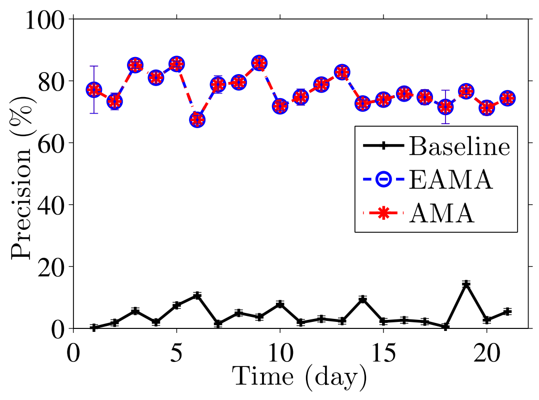


Figure 3.10: Energy consumption of individual users in a sample EM problem.

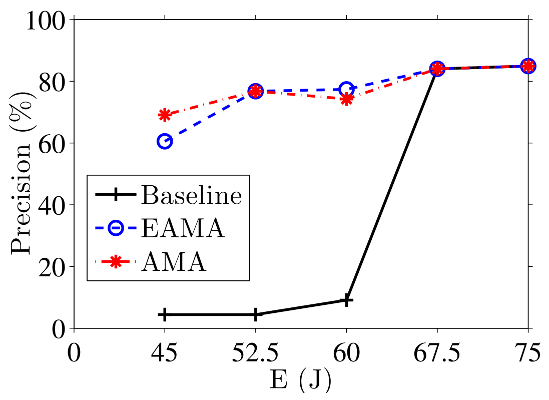
and $\gamma = 15$ if not otherwise specified. The mapping between combinations and sensors (M) is randomly chosen following a Bernoulli trail with probability 20%. We use the generated synthetic traces to drive the simulator to evaluate the proposed scheduling algorithms. Furthermore, we also consider heterogeneous frequency/sampling rate in simulations. We assign a random frequency uniformly distributed in $[1, 1/150]$ Hz to each request and give every context a random degrading ratio h_r uniformly distributed in $[\frac{1}{3}\%, \frac{1}{6}\%]$. That is, even with a precision of 100% at the beginning of a scheduling window, it will drop to 0% before the end of that scheduling window.

3.4.2 Simulation Results

Minimizing energy consumption. We first validate the correctness of the baseline, EMA, and EEMA algorithms: we find that all requests from apps are satisfied by the resulting schedules. Next, we normalize the energy consumption achieved by each algorithm to that of the baseline and plot it in Fig. 3.8(a). This figure reveals that our proposed EEMA algorithm: (i) outperforms the baseline by at least 30% and (ii) often closely follows the

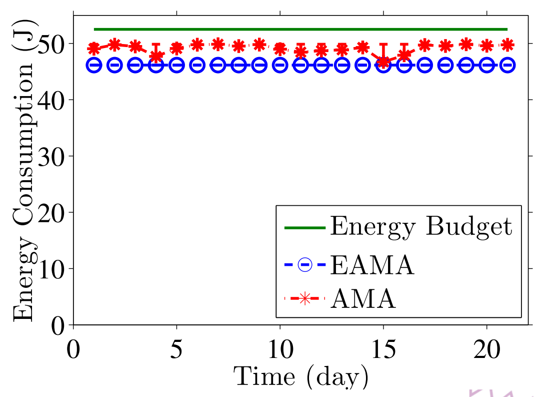


(a)

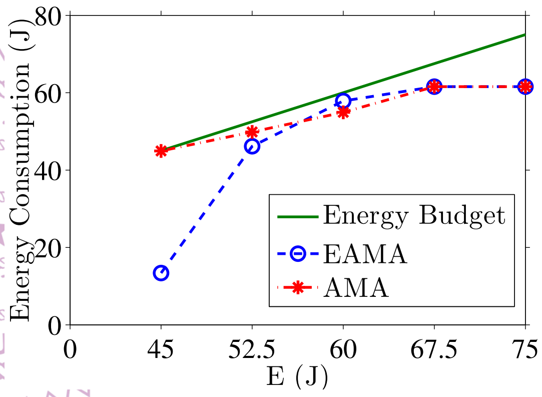


(b)

Figure 3.11: Average precision of the AM problem: (a) with $E = 52.5$ J and (b) diverse E.



(a)



(b)

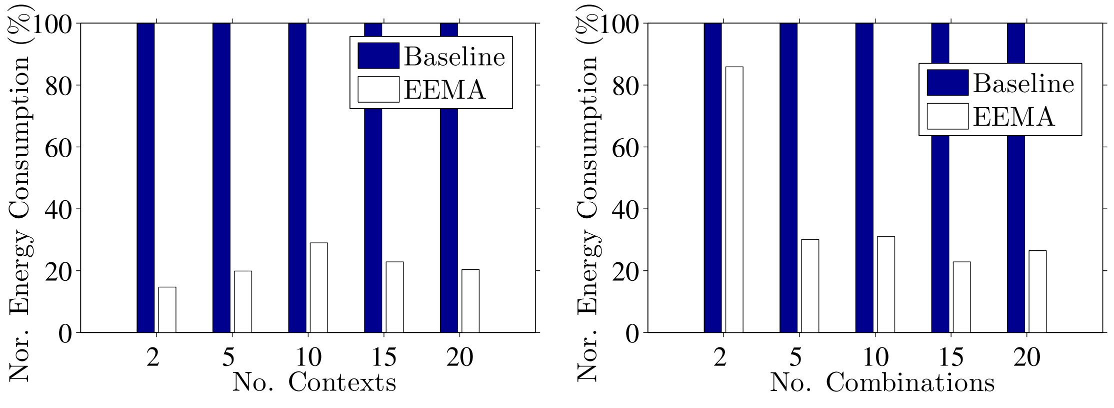
Figure 3.12: Average energy consumption of the AM problem: (a) with $E = 52.5$ J and (b) diverse E.

EMA in terms of energy consumption. We next plot the aggregate results under different α values in Fig. 3.8(b). This figure shows that the EEMA algorithm constantly leads to good results, although the optimization room reduces with larger α values. This is because when the requested precision is higher, all the algorithms are forced to activate the sensors that are more accurate and energy hungry. Fig. 3.8 shows the effectiveness of our proposed algorithms for the EM problem. Moreover, we show the mean precision of the algorithms in Fig. 3.9. Fig. 3.9(a) shows that the EEMA achieves at least 53.60% higher precision than the baseline and on average only $\sim 3\%$ gap than the EMA. In Fig. 3.9(b), the improvement of precision decreases when the α increases, as the energy budget is fixed across different α values. Last, we present the energy consumption of individual users in Fig. 3.10. Among the 5 users, we found that user 1 consumes the most overall energy, whereas user 3 consumes the least overall energy. A deeper analysis indicates that user 1 runs the most context-aware apps, on average 5.15 concurrent requests; user 3 runs the fewest context-aware apps, on average 2.48.

Maximizing precision. We first validate that all the resulting schedules from the baseline, AMA, and EAMA algorithms always satisfy the energy budget. We calculate the precision achieved by each algorithm with $E = 52.5$ J and plot it in Fig. 3.11(a). This figure reveals that our proposed EAMA algorithm: (i) outperforms the baseline by 72.38% on average and (ii) achieves a small optimization gap of $\sim 0.1\%$ than the AMA. Fig. 3.11(b) shows the overall precision under different energy budgets. This figure shows that the AMA algorithm constantly leads to higher precision than baseline. Moreover, the EAMA algorithm performs well when the energy budget is lower; it even occasionally outperforms the AMA algorithm (under 1-min cap of execution time), e.g., when $E = 60$ J. However, when the energy budget is high, the performance of all algorithms is similar, as the room for optimization is small. Moreover, we plot the energy consumption per day and the aggregate results under different E values in Fig. 3.12. Fig. 3.12(a) reveals that algorithms always produce schedules within under the energy budget, and Fig. 3.12(b) shows that the EAMA saves more energy when E decreases.

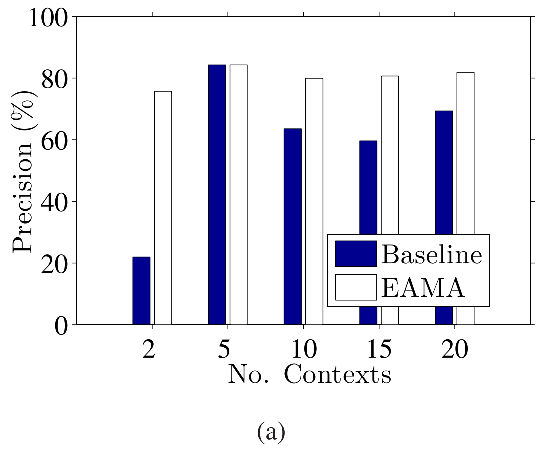
Table 3.4: Running Time of Various Algorithms (in ms)

α	**EMA**	**EEMA**	**E**	**AMA**	**EAMA**
25%	58	≤ 1	45 J	50153	≤ 1
50%	47	≤ 1	52.5 J	63316	≤ 1
75%	22	≤ 1	60 J	60416	≤ 1
90%	15	≤ 1	67.5 J	63267	≤ 1



(a) (b)

Figure 3.13: Scalability of the EEMA algorithm: (a) diverse contexts and (b) diverse combinations.



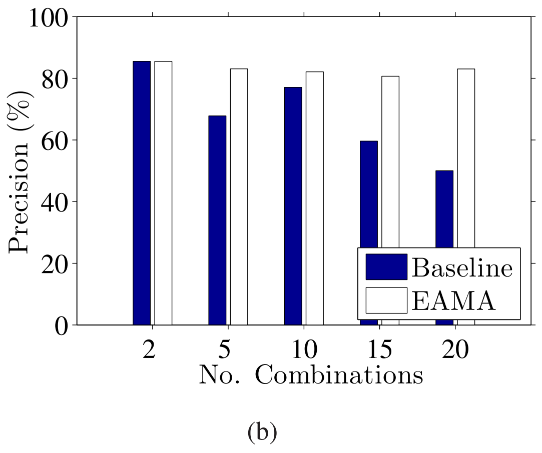


Figure 3.14: Scalability of the EAMA algorithm: (a) diverse contexts and (b) diverse combinations.

Running time. Table 3.4 reports the running time of all the considered algorithms. We observe that the EEMA and EAMA run 35.5 and 59288 times faster than the EMA and AMA. Since the EMA/AMA cannot run in real-time, we do not consider them in the rest of this chapter. In contrast, the EEMA and EAMA algorithms run in real-time on commodity PC. We will port the EEMA and EAMA algorithms to Android smartphone in Sec. 3.5 and measure their running time in a real prototype system.

Scalability. We report the performance of the proposed algorithms with larger problems using synthetic traces. Fig. 3.13 shows the energy saving of the EEMA algorithm, compared to the baseline. Fig. 3.13(a) shows that the EEMA algorithm outperforms the baseline, by at least 71.03% in energy saving. Fig. 3.13(b) shows that more combinations give more rooms for optimization. Fig. 3.14 reports the achieved precision of the EAMA algorithm. This figure shows that the EAMA outperforms the baseline by up to 53.71% in precision. However, when the problem sizes are small (e.g., 5 contexts or 2 combinations), there may not be any room for the EAMA algorithm to improve over the baseline.

Heterogeneous frequency/sampling rate. We next extend the EEMA and EAMA algorithms by taking heterogeneous frequency and sampling rate into consideration (Sec. 3.3.4), and refer to the new algorithms as EEMA* and EAMA*. We first validate that the EEMA* and EAMA* algorithms always satisfy the precision requirements, even though the sampling rates may be lower (for saving energy). Next, we report the normalized energy consumption in Fig. 3.15. Fig. 3.15(a) shows that the EEMA and EEMA* outperform the baseline by up to 64.58% and 84.64%, respectively. This also demonstrates the additional optimization room, $\sim 20\%$, offered by heterogeneous frequency/sampling rates. On the other hand, we find that the EAMA and EAMA* also outperform the baseline by up to 33.68% and 80.48%, when an even higher energy saving ($\sim 45\%$) due to heterogeneous frequency/sampling rates. The high energy saving can be partially attributed to the high precision achieved by the EAMA, which grants more space for the EAMA* to reduce the sampling rates to save more energy. Fig. 3.15(b) gives the aggregate energy consumption under different α values, which shows the room for optimization decreases with larger α values. Nonetheless, compared to the baseline, significant energy saving is achieved by our proposed algorithms under all α values.

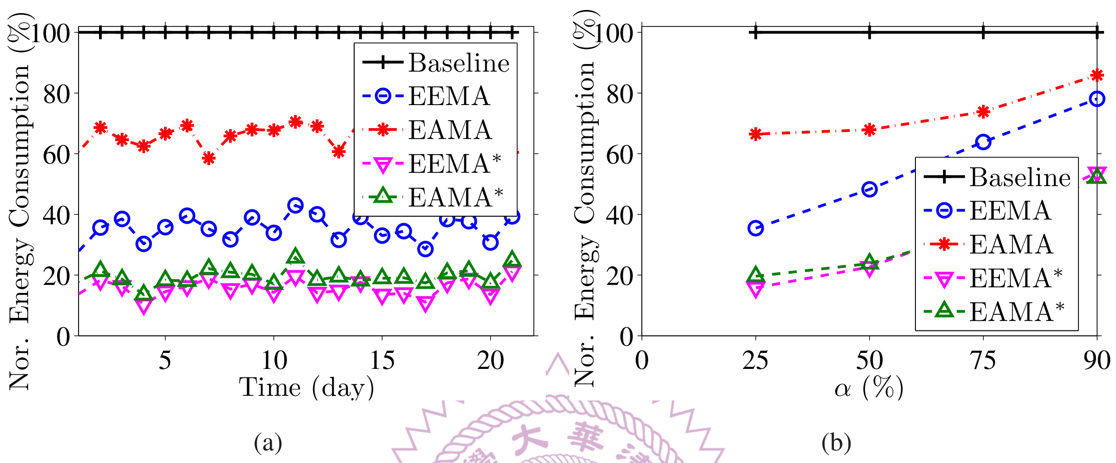


Figure 3.15: Average energy consumption with heterogeneous frequency and sampling rate: (a) with $\alpha = 25\%$ and (b) diverse α values.

3.5 Experiments

In this section, we report the learned lessons from the OSM Android implementation and real experiments.

3.5.1 Implementation

We have implemented a proof-of-concept prototype OSM system and the proposed EEMA/EAMA algorithms on Android (Fig. 3.16). We have also implemented a baseline algorithm for comparisons, in which apps independently activate the sensors based on their own requirements. Using the OSM API, we have built three context-aware apps: (i) *Calorie Calculator*, which periodically recognizes the user action (still/walk/run), (ii) *Mobile Assistant*, which determines the user location (library/home/laboratory/food court), and (iii) *Where to Eat*, which recommends food places based on user locations. We install our software on Samsung and HTC smartphones.

3.5.2 Setup

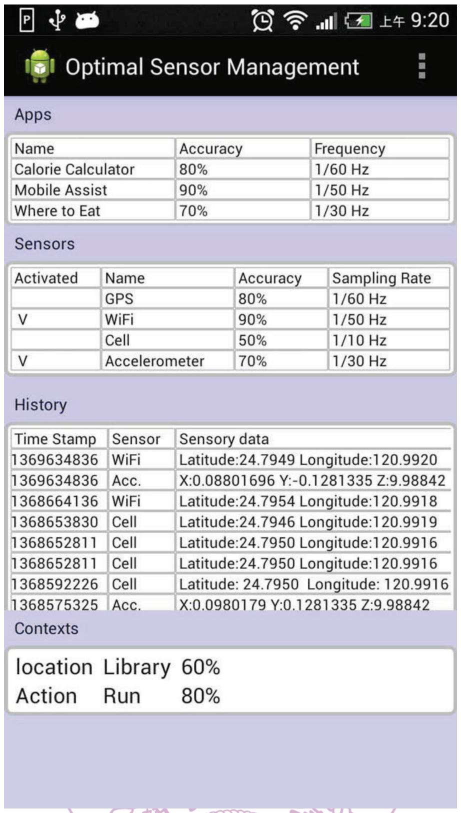
We recruit 7 students and instruct each of them to perform a 2-hour experiment on our campus. During each experiment, we generate random context requests from context-aware apps following a Poisson arrival process with a mean inter-request time of 1 min. The request precision follows a normal distribution with a mean of 70% and a variance of 1.66%. The context requests from the 3 apps can be classified into: (i) action and (ii) location contexts. For each student, we divide the 2-hour experiment into two halves. We periodically prompt students to provide the ground truth. In particular, we use the ground truth collected in the first hour to train the personalized models, and the ground truth collected in the second hour to compute the achieved precision. The second hour is further divided into 3 parts, in which the OSM middleware runs the baseline, EEMA, and EAMA algorithms, respectively.

On average, each student provides 2732 times of ground truth. We split the 7 students into two groups for two experiment scenarios. The first scenario includes 3 students with homogeneous scheduling window sizes of 20 mins, i.e., the scheduling algorithm is invoked once every 20 mins. The second scenario includes 4 students with heterogeneous scheduling window sizes as short as 3.33 mins. We consider three performance metrics: (i) energy consumption, which is in battery level (%) or Joules, (ii) execution time, which is further broken down to request preprocessing, scheduling algorithm, sensor activation/reading, and context inference times, and (iii) fraction of the correctly inferred contexts based on the ground truth.

3.5.3 Results

Fraction of correctly inferred contexts. Fig. 3.17 gives the sample inferred contexts. Fig. 3.18 presents the sample correct/wrong contexts, which reveal that the majority of inferred contexts are correct. This demonstrates the effectiveness of our inference algorithms. We give the overall fraction of correct inference in Table 3.5. This table confirms that our proposed algorithms are correct of 83+%, which is much higher than the mean precision requirements (inputs) of 70%.

Tradeoff between energy consumption and precision. The proposed EEMA and EAMA algorithms provide apps/users a choice to exercise the trade-off between energy consumption and precision. Compared to the baseline, we observe average energy saving of 61.34% and 22.08%, and average precision of 93.94% and 94.85%, respectively. Moreover, the precision always exceeds the precision requirements. In addition to the energy saving due to sensors, we also log the battery level throughout the experiments. We fully charge the smartphones, and report the drops of battery levels in Table 3.6. This


上午 9:20

Optimal Sensor Management

Apps

Name	Accuracy	Frequency
Calorie Calculator	80%	1/60 Hz
Mobile Assist	90%	1/50 Hz
Where to Eat	70%	1/30 Hz

Sensors

Activated	Name	Accuracy	Sampling Rate
	GPS	80%	1/60 Hz
V	WiFi	90%	1/50 Hz
	Cell	50%	1/10 Hz
V	Accelerometer	70%	1/30 Hz

History

Time Stamp	Sensor	Sensory data
1369634836	WiFi	Latitude:24.7949 Longitude:120.9920
1369634836	Acc.	X:0.08801696 Y:-0.1281335 Z:9.98842
1368664136	WiFi	Latitude:24.7954 Longitude:120.9918
1368653830	Cell	Latitude:24.7946 Longitude:120.9919
1368652811	Cell	Latitude:24.7950 Longitude:120.9916
1368652811	Cell	Latitude:24.7950 Longitude:120.9916
1368592226	Cell	Latitude: 24.7950 Longitude: 120.9916
1368575325	Acc.	X:0.0980179 Y:0.1281335 Z:9.98842

Contexts

location	Library	60%
Action	Run	80%

Figure 3.16: User interface of OSM.

table reveals that the proposed OSM middleware may prolong the battery life for up to 2 times. Since our proposed algorithms outperform the baseline by far, we no longer report results from the baseline algorithm in the rest of this section.

Implication of scheduling window size. We plot the battery consumption and achieved precision under different scheduling window sizes in Fig. 3.19. We make three observations: (i) longer scheduling window sizes lead to lower battery consumption at the expense of lower precision, (ii) EEMA constantly saves more energy compared to EAMA, and (iii) both EEMA and EAMA achieve very high precision. We note that EEMA may occasionally achieve higher precision than EAMA (e.g., with scheduling window size 10 min) as indicated in Fig. 3.19(b). This may be attributed to the smaller scale of our experiments; larger deployment is among our future tasks.

Execution time. We first report the total execution time, which is the time difference between starting the request preprocessing, and receiving the inferred contexts. The execution times with the EEMA and EAMA algorithms are at most 306 ms and 503 ms in wall clock time. A closer look indicates that majority of the execution time is due to sensor activation and context inference. For example, an accelerometer based inference algorithm may read 100 readings in 250-ms duration for analysis, which contributes to the execution time. Nevertheless, the execution time is negligible compared to the scheduling

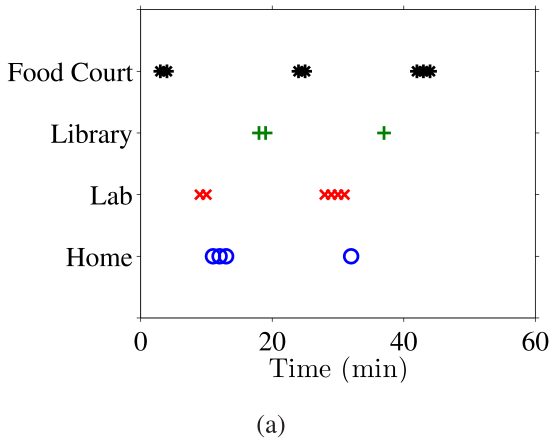


Figure 3.17: Sample inferred contexts: (a) locations of student 1 and (b) actions of student 2.

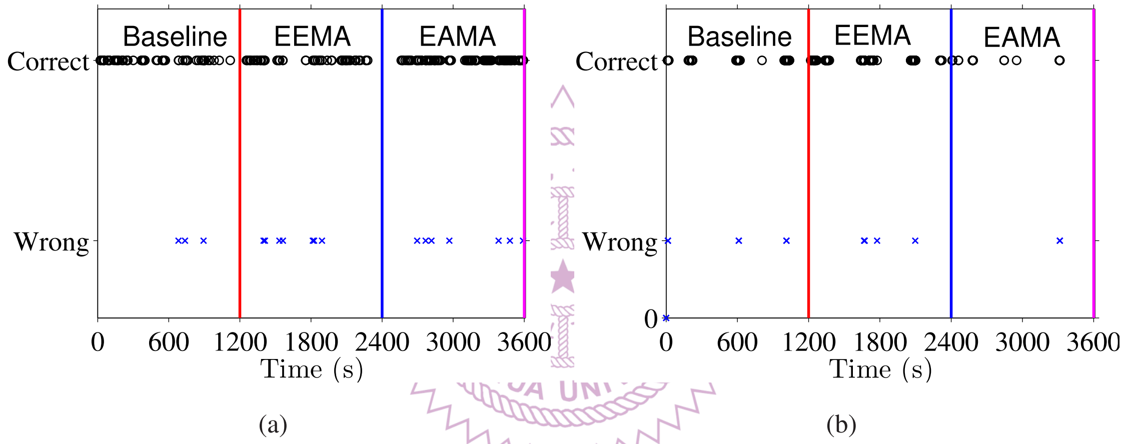


Figure 3.18: Sample correct/wrong contexts: (a) actions of student 1 and (b) locations of student 3.

window, which is in the order of minutes.

To better understand the scalability of our proposed algorithms, we generate synthetic, larger request traces by varying the number of contexts (combinations) between 7 and 15, while fixing the number of combinations (contexts) at 7. We then run an 1-hour experiment on a smartphone, and report the average running time in Fig. 3.20. This figure clearly shows that our EEMA and EAMA algorithms are efficient even on resource-constrained smartphones. Moreover, the running time increases linearly to the scheduling problem size. This shows that the EEMA and EAMA scale to more contexts/combinations, which will become a reality in the future.

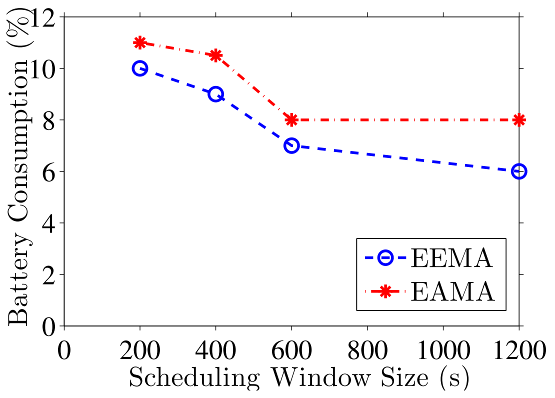
Inference algorithms energy consumption. We conduct measurements using the Agilent 66321D battery emulator [2] and a Samsung smartphone to quantify the sensor and algorithm energy consumption of three sample inference algorithms: (i) WiFi/location, (ii) cellular/location, and (iii) accelerometer/action. We invoke each inference algorithm

Table 3.5: Fraction of Correct Inference

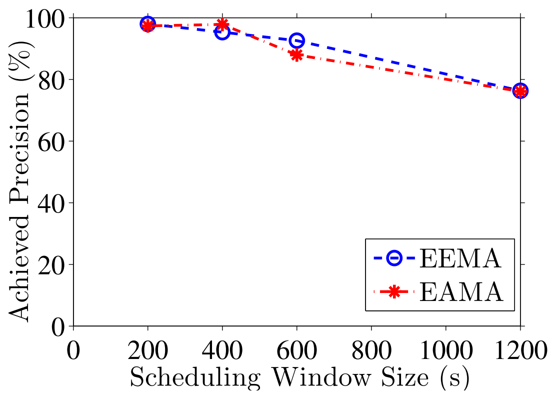
Student	1	2	3
Baseline (%)	94.63	85.34	89.61
EEMA (%)	97.99	84.98	90.46
EAMA (%)	83.09	86.12	89.84

Table 3.6: Consumed Battery Level

Student	1	2	3
Baseline (%)	12	11	7
EEMA (%)	6	6	6
EAMA (%)	8	8	7



(a)



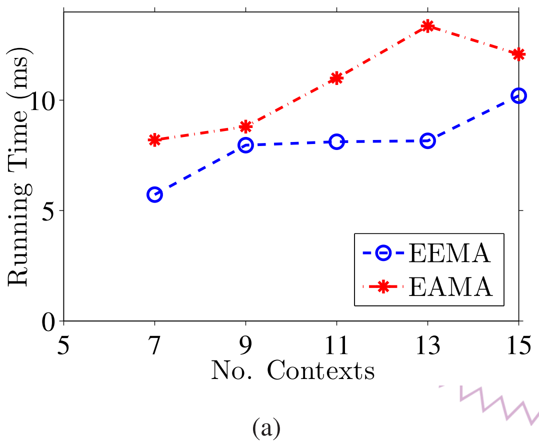
(b)

Figure 3.19: Implication of scheduling window sizes on: (a) battery consumption and (b) achieved precision.

once every minute, and measure the energy consumption. By repeating the measurements with/without sensor activations and algorithm invocations, we report the average energy consumption across 5 measurements in Table 3.7. This table shows the energy consumption of sensor, algorithm, and remaining smartphone components (other). We observe that the inference algorithms only account for a small portion (¡ 5%) of smartphone energy consumption. Although there are computationally-complex inference algorithms out there, they will gradually be implemented on special-purpose chips to meet the stringent energy budget of smartphones, as witnessed recently [42].

Table 3.7: Energy Consumption of Sensors and Algorithms (J)

Inference Algorithm	Sensor	Algorithm	Other
WiFi/Location (J)	3.73	3.48	404.13
Cellular/Location (J)	13.51	22.87	404.13
Accelerometer/Action (J)	4.97	0.23	404.13



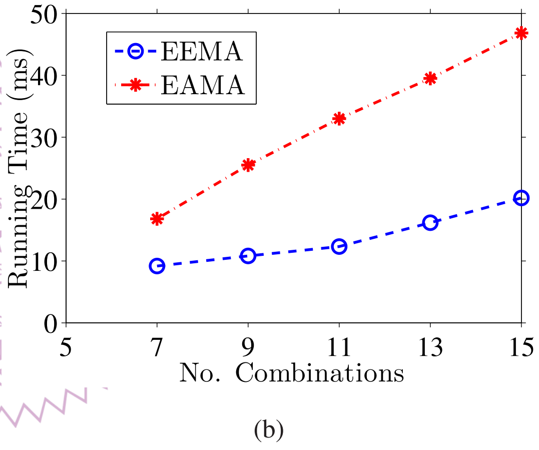


Figure 3.20: Running time of our proposed algorithms: (a) different number of contexts and (b) different number of combinations.

Chapter 4

Sensor Scheduling for Multiple Devices: CrowdSensing

4.1 Framework

Fig. 4.1 gives an overview of the proposed system, which consists of the *broker*, *servers*, and *users*. Users send queries to the broker for the events they are interested in, such as measuring traffic congestion on a specific street or the degree of crowdness at a popular fair. Both smartphone users and in-situ sensors are referred to as workers. The servers collect sensory data from workers and run analysis algorithms to derive the answers. Last, the servers return the results to users who issued queries via the broker. The smartphone users get to ask and answer queries, and each query specifies: (i) the interested event, (ii) the target accuracy, and (iii) the covered geolocation. The broker is a logically centralized service and composed of several management modules. Among them, the task assignment algorithm is responsible for solving the worker selection and sensor scheduling problem. In the rest of the chaper, we focus on the development of the task assignment algorithm.

4.2 Task Scheduling Problem

4.2.1 System Models

In this chapter, we assume each event is inferred by inference algorithms, and different inference algorithms employ different combinations of sensors. Let $\mathbf{C}$ and $\mathbf{S}$ be the sets of combinations and sensors, respectively. We discretize the map into *grids* and use $\mathbf{L}$ to denote the set of the locations. We let $d_{l,l'}$ be the distance between $l, l' \in \mathbf{L}$. $\mathbf{W}$ is the set of workers, and $\mathbf{W}^*$ is a subset of $\mathbf{W}$ consisting of all smartphones. $G^{w}_{s,l,l'}$ is a boolean indicator, which is 1 if sensor s ($s \in \mathbf{S}$) can gather the sensory data of location l' when

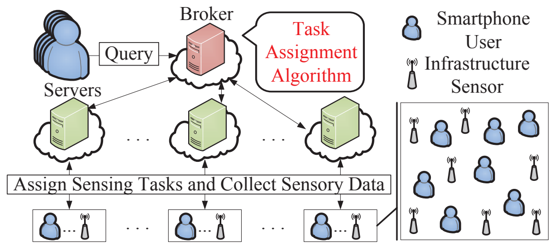

Broker
Query
Task Assignment Algorithm
Smartphone User
Infrastructure Sensor
Servers
Assign Sensing Tasks and Collect Sensory Data

Figure 4.1: The SAIS system overview. Our focus in on the task assignment algorithms.

worker w ($w \in \mathbf{W}$) is at location l; $G^w_{s,l,l'}$ is 0 otherwise.

Our problem is to determine the assignments of queries to workers, and the traverse routes of individual workers. We define two decision variables $x_{w,l,s}$ and $E^w_{i,j}$ for these two purposes, respectively. $x_{w,l,s}$ indicates how many times that worker w should perform sensing at location l with sensor s. $E^w_{i,j}$ is 1 if worker w moves from location i to location j. We notice that different requests may have diverse target accuracy levels, and some of them may dictate multiple workers to sense the same events. Therefore, we use $O_{l,s}$ to denote the number of workers that use their sensor s to collect the sensory data of location l. We write $O_{l,s}$ as a function of $G^w_{s,l,l'}$ and $x_{w,l,s}$:

$$O_{l,s} = \sum_{w \in \mathbf{W}} \sum_{l' \in \mathbf{L}} G^w_{s,l',l} x_{w,l',s}, \forall s \in \mathbf{S}, \forall l \in \mathbf{L}. \tag{4.1}$$

The sensing energy consumption of worker w while using sensor s is denoted as $e_{w,s}$, which includes both sensing and transmission energy. F_w is the residual energy of worker w, and each worker has his/her own energy threshold θ_w before he/she exits the crowd-sensing system. Our goal is to minimize the carbon footprint, which is the total energy consumption, including both sensing and user movement energy. The current location of worker w is A_w. $m_{c,s}$ is 1 if combination $c \in \mathbf{C}$ uses sensor $s \in \mathbf{S}$, and is 0 otherwise. $\mathbf{R}$ is the set of queries and f_r is the target accuracy for query $r \in \mathbf{R}$. The required location of query r is denoted as r_p, and r_c is the combination which be selected by EEMA. The accuracy of inferring query r with r_c is denoted as a_{r_c}. We let τ_r to be how many times query r be covered by workers :

$$\tau_r = min_{\forall s \in \mathbf{S}, m_{r_c,s}=1}\{O_{r_p,s}\}, \forall r \in \mathbf{R}. \tag{4.2}$$

$Q(\tau_r, a_{r_c})$ is a function returning the sensing quality of the result, which is a final answer to query r based on the results of combination c from τ_r nearby workers and the accuracy of the combination. $Q(\tau_r, a_{r_c})$ can be derived using approaches proposed in Liu et al. [28] and Xing et al. [44]. In fact, we can pre-compute a table using the experiment setup [28, 44] and obtain the accuracy by looking up the table for any locations and combination.

4.2.2 Problem Formulations

We formulate a problem of minimizing the overall carbon footprint in the following. We write the sensing energy consumption of worker w as $\sum_{l\in\mathbf{L}}\sum_{s\in\mathbf{S}} e_{w,s}x_{w,l,s}$, and the amount of generated carbon as $\alpha\sum_{l\in\mathbf{L}}\sum_{s\in\mathbf{S}} e_{w,s}x_{w,l,s}$, where α is a factor in g/J. We next give the total distance worker w travels as $\sum_{l,l'\in\mathbf{L}} d_{l,l'}E^{w}_{l,l'}$, and the amount of generated carbon is given by $\beta\sum_{l,l'\in\mathbf{L}} d_{l,l'}E^{w}_{l,l'}$, where β is a factor in g/km. Then, we can write the objective function as:

$$\min\{\alpha\sum_{w\in\mathbf{W}}\sum_{l\in\mathbf{L}}\sum_{s\in\mathbf{S}} e_{w,s}x_{w,l,s} + \beta\sum_{w\in\mathbf{W}^*}\sum_{l,l'\in\mathbf{L}} d_{l,l'}E^{w}_{l,l'}\}. \tag{4.3}$$

Moreover, we write all the constraints as:

$$Q(\tau_r, a_{r_c}) \geq f_r, \forall r \in \mathbf{R}; \tag{4.4}$$

$$F_w - \sum_{l\in\mathbf{L}}\sum_{s\in\mathbf{S}} e_{w,s}x_{w,l,s} \geq \theta_w, \forall w \in \mathbf{W}^*; \tag{4.5}$$

$$\sum_{j\in\mathbf{L}} E^{w}_{j,A_w} = 0, \forall w \in \mathbf{W}^*; \tag{4.6}$$

$$\sum_{j\in\mathbf{L}} E^{w}_{A_w,j} = \left\lceil \frac{\sum_{j\in\mathbf{L}}\sum_{s\in\mathbf{S}}\left\lceil \frac{x_{w,j,s}}{x_{w,j,s}+1}\right\rceil}{|\mathbf{L}||\mathbf{S}|+1} \right\rceil, \forall w \in \mathbf{W}^*; \tag{4.7}$$

$$\sum_{j\in\mathbf{L}\cup A_w, j\neq i} E^{w}_{j,i} = \left\lceil \frac{\sum_{s\in\mathbf{S}}\left\lceil \frac{x_{w,i,s}}{x_{w,i,s}+1}\right\rceil}{|\mathbf{S}|+1} \right\rceil, \quad \forall i \in \mathbf{L}, \forall w \in \mathbf{W}^*; \tag{4.8}$$

$$\sum_{j\in\mathbf{L}, j\neq i} E^{w}_{i,j} \leq \left\lceil \frac{\sum_{s\in\mathbf{S}}\left\lceil \frac{x_{w,i,s}}{x_{w,i,s}+1}\right\rceil}{|\mathbf{S}|+1} \right\rceil, \forall i \in \mathbf{L}, \forall w \in \mathbf{W}^*. \tag{4.9}$$

Eq. (4.4) states that the required accuracy f_r should be satisfied. The energy constraint is considered in Eq. (4.5). Eqs. (4.6)–(4.9) ensure the correctness of the paths for smartphone users to move to the query locations. Eq. (4.6) states that A_w is the start location of the path for worker w. Eq. (4.7) shows that worker w should start from A_w and go to a single next location if worker w is assigned any queries. Eqs. (4.8) and (4.9) ensure that the locations are dictated by assigned queries.

4.2.3 Optimal Task Scheduling Algorithm (OPT)

The optimization problem is solved by CPLEX [12]. However, the formulation does not prevent cycles, and we realize a common cycle elimination technique [18, 41], which iteratively removes a link from any cycle until the solution is cycle free. In particular, we

```
1: Input: $\Phi =< \mathbf{L}, \mathbf{R}, \mathbf{W}, \boldsymbol{\mu}, \widehat{\mathrm{R}} >$
2: Output: $\mathbf{X}, \mathbf{E}$
3: while $\widehat{\mathrm{R}} \neq \emptyset$ do
4:     $\mathbf{z}$ = Sensor_scheduling($\Phi$)
5:     Compute($\boldsymbol{\Lambda}$) using Eq. (4.11)
6:     Sort($\boldsymbol{\Lambda}$)
7:     $\lambda_{w,l}$ = pop($\boldsymbol{\Lambda}$)
8:     for $s \in \mathbf{S}$ do
9:         $x_{w,l,s} = x_{w,l,s} + z_{w,l,s}$
10:    $E^w_{A_w,l} = 1$
11:    update($\mathbf{E}$, $\mathbf{W}$, $\boldsymbol{\mu}$)
12:    for $r \in \mathbf{R}$ do
13:        if $Q(\mu_{r_p,r}, a_{r_c}) \geq f_r$ then
14:            remove $r$ from $\widehat{\mathrm{R}}$
15: return $\mathbf{X}, \mathbf{E}$
```

Figure 4.2: Efficient task assignment algorithm.

develop an optimal algorithm using CPLEX [12] and the cycle elimination technique [18, 41]. We refer to the algorithm as OPT.

4.2.4 Efficient Task Scheduling Algorithm (ETA)

The OPT algorithm gives the optimal task assignment, but may lead to long running time for larger problems. We next develop an efficient task assignment algorithm, called ETA. The algorithm first computes the sensor schedule for each worker to determine the sensors that should be turned on each location. We let $z_{w,l,s}$ be a boolean variable, which is 1 if and only if worker w has to turn on sensor s at location l. Once we have the sensor schedule, we can compute the number of queries that can be covered by worker w at location l, which is denoted as $T_{w,l}$:

$$T_{w,l} = \sum_{r \in \mathbf{R}} \left\lfloor \frac{\sum_{s \in \mathbf{S}} n_{r_c,s} z_{w,l,s} G^w_{s,l,r_p}}{\sum_{s \in \mathbf{S}} n_{r_c,s}} \right\rfloor, \forall w \in \mathbf{W}, l \in \mathbf{L}. \tag{4.10}$$

We iteratively assign a task, which consists of the target location and sensor schedule, to a worker. We define $\lambda_{w,l}$ as the utility of worker w performing tasks at location l:

$$\lambda_{w,l} = \frac{T_{w,l}}{\alpha \sum_{s \in \mathbf{S}} e_{w,s} z_{w,l,s} + \beta d_{A_w,l}}, \forall w \in \mathbf{W}, l \in \mathbf{L}, \tag{4.11}$$

Which is a ratio of the number of covered queries and the carbon footprint. We assign the tasks at location l to worker w who has the largest utility $\lambda_{w,l}$. ETA algorithm checks whether required accuracy levels of queries are satisfied in each run, and the iteration is terminated if all the queries are satisfied or the workers cannot perform any more task. Notice that if the moving cost is higher than the sensing cost, ETA assigns tasks that are close to workers. If the sensing cost becomes dominating due to advance of vehicle technologies, ETA may assign some further tasks to workers for higher sensing efficiency.

Fig. 4.2 presents our ETA algorithm. We use $\mu_{l,r}$ and $\widehat{\mathbf{R}}$ to denote the number of workers performing query r and the set of queries that are not satisfied, respectively. $\boldsymbol{\mu}$ is the set of $\mu_{l,r}$. The inputs of the algorithm are $\mathbf{L}$, $\mathbf{R}$, $\mathbf{W}$, $\boldsymbol{\mu}$, and $\widehat{\mathbf{R}}$. The algorithm outputs $\mathbf{X}$ and $\mathbf{E}$, which are the sets of $x_{w,l,s}$ and $E^{w}_{i,j}$, respectively. The algorithm works as follows. First, we compute sensor schedule $\mathbf{z}$, which is the set of $z_{w,l,s}$, for all the workers. Second, we compute $\boldsymbol{\Lambda}$, which is a heap contains $\lambda_{w,l}$. $\lambda_{w,l}$, which has the largest value, is popped out and we assign the task to worker w. We then update the status of worker w and $\boldsymbol{\mu}$ for all the queries covered by worker w. The for-loop checks whether the queries are satisfied, if so, remove the queries from $\widehat{\mathbf{R}}$. We repeat the process until all the queries are satisfied or workers cannot perform tasks anymore.

4.3 Trace-Driven Simulations for Multiple Devices

4.3.1 Simulation Setup

We implemented our task assignment algorithm (ETA) and three baseline algorithms: (i) *infrastructure sensors only (IS)*, (ii) *infrastructure sensors with opportunistic sensing (ISOS)*, and (iii) *optimal (OPT)*, in a simulator. IS uses infrastructure sensors to provide sensory data, and ISOS uses infrastructure sensors and opportunistic sensing, where smartphone users are moving following the random way point model. IS and ISOS are implemented in Java and OPT is implemented in CPLEX.

For the queries, we use real trace data collected from a popular Bulletin Board System (BBS) in Taiwan. We collect the traces of posts and use them as queries. The post time and IP address of the author are query's time and location. We connect the IP address into geolocation using IPInfoDB [1]. Since IPInfoDB claims the error range is 25 mile, we add noise to the transformed locations with normal distribution to make the location more realistic. The number of queries are varying from $\{500, 1000, 1500, 2000, 2500\}$, and if not specified, we set it to be 2000. We consider six types of events in our simulation, and each query is asking one of the events. The required accuracy of each query is uniformly distributed in [50%–100%]. The number of mobile users and infrastructure sensors are varying from $\{500, 1000, 1500, 2000, 2500\}$ and $\{500, 1000, 1500\}$. If not specified, the number of smartphone users and infrastructure sensors are 2000 and 1000, respectively.

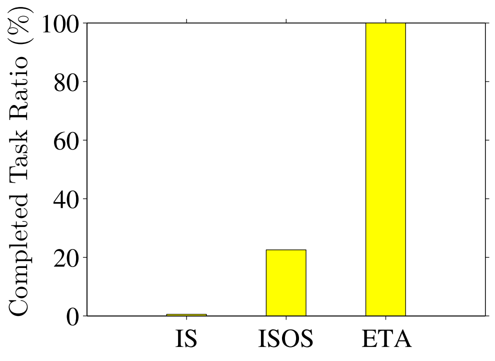


Figure 4.3: Completed task ratio of IS, ISOS, and ETA.

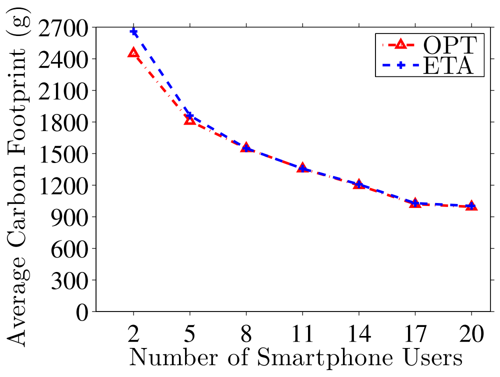
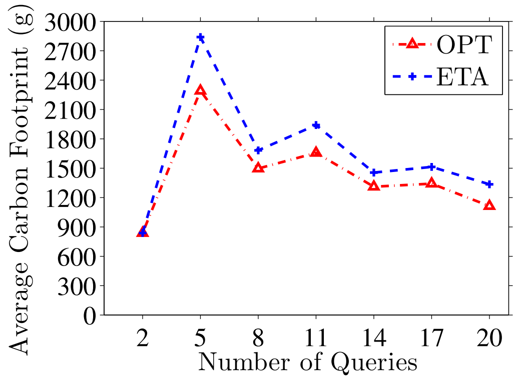


Figure 4.4: Carbon footprint of ETA and OPT while varying number of smartphone users.

Since we consider all the workers are consumers and producers in our system, we let the collected locations to be the locations of smartphone users as well. For the infrastructure sensors, we discretize Taipei City into grids and uniformly distribute infrastructure sensors in the map. For each smartphone user, we randomly determine the battery capacity. The current energy level and threshold are uniformly distributed in [20%–80%] and [10%–20%]. The sensor energy and inference algorithm accuracy is derived in Chap 3.

In order to understand the performance of our algorithm, we use (i) completed task ratio, (ii) carbon footprint, (iii) running time, and (iv) responding time to be our metrics. The responding time is defined as the time difference between the broker receives the query and the query is satisfied.

4.3.2 Simulation Results



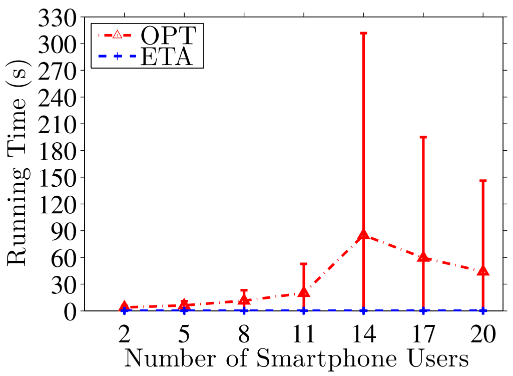
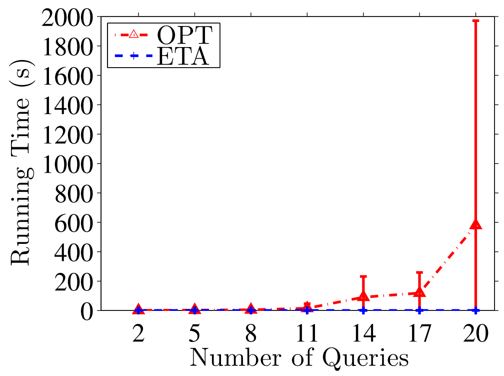


Figure 4.5: Carbon footprint of ETA and OPT while varying number of queries.

Figure 4.6: Running time of ETA and OPT while varying number of smartphone users.

Advantage of combining infrastructure sensing and Crowdsensing. Fig. 4.3 shows



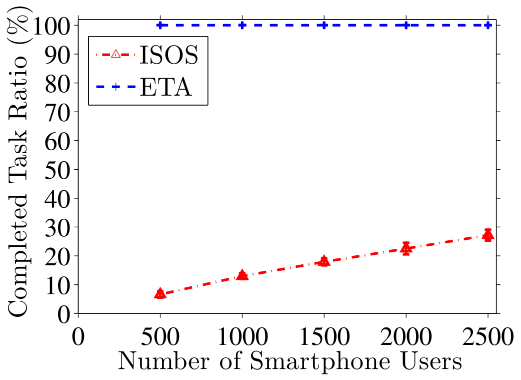


Figure 4.7: Running time of ETA and OPT while varying number of queries.

Figure 4.8: Completed task ratio while varying number of smartphone users.

the benefit of using both infrastructure sensors and smartphone sensors. It shows that infrastructure sensors can only cover 0.52% locations, and smartphone users can significantly improve the coverage of the system. Although ISOS improves the coverage to 22%, it still suffers from 78% incomplete queries. ETA instructs smartphone users to the required locations and thus covers more locations. Hence, we don't consider IS in the following experiments.

The performance gap between ETA and OPT is small. We compare the performance of ETA and OPT in two settings: (i) different number of workers with 10 queries and (ii) different number of queries with 10 workers. The carbon footprint gap is shown in Fig. 4.4 and Fig. 4.5. As the figures show, ETA consumes almost the same carbon footprint as OPT does, where the gaps are about 2% in Fig. 4.4 and 14% in Fig. 4.5. However, ETA has shorter running time, which is shown in Fig. 4.6 and Fig. 4.7. In particular, when the numbers of smartphone users or queries are huge, the running of time of ETA is 1333 times faster than OPT. Hence, we do not consider OPT in the following experiments.

ETA outperforms ISOS in many aspects. Fig. 4.8 reveals that more smartphone users indeed improve the completed task ratio of ISOS but still far behind 100%. In contrast, ETA achieves 100% completed task ratio with very few smartphone users. Fig. 4.9 shows that ETA achieves 95% when there are 4 smartphone users in the system, and achieves 99% with only 8 smartphone users. However, ISOS leads to a terrible completed task ratio almost 0%. This may be attributed to the fact that some queries are at remote locations, and opportunistic sensing cannot direct smartphone users to perform the tasks. Therefore, it is really hard for smartphone users opportunistically satisfy all queries. Fig. 4.10 gives the average carbon footprint of each query while varying the number of smartphone users. ISOS consumes up to 364 times higher energy than ETA, and ETA results in constant carbon footprint. This is because workers in ISOS are wandering and wasting large amount of energy on moving, which becomes more severe when the number of smartphone users increases. We show the average responding time of ISOS and ETA in Fig. 4.11. The average responding time of ISOS is up to 8 times higher than ETA. This is because smartphone users are moving whatever they want in ISOS, and ETA directs smartphone users to preferred locations. The responding time of ISOS is slightly decreasing when the number of smartphone users is growing. It is intuitive that more smartphone users in the system, more chance the tasks be covered by some of the users. We show the scalability of ETA is good when number of queries is growing in Fig. 4.12. As Fig. 4.12 shows, the completed task ratio of ISOS is lower than 23% and ETA still achieves 100%. The average responding time of varying the number of queries is shown in Fig. 4.13. ETA has smaller responding time compare to ISOS: up to 8 times faster than ISOS.

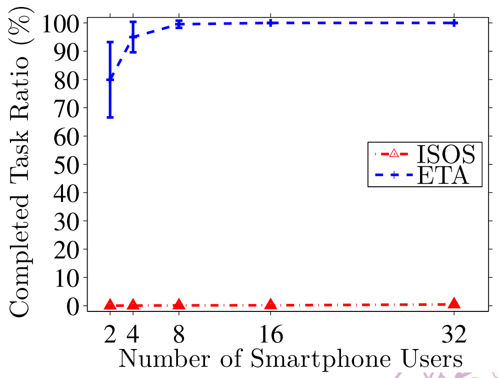


Figure 4.9: Completed task ratio when number of smartphone users is small.

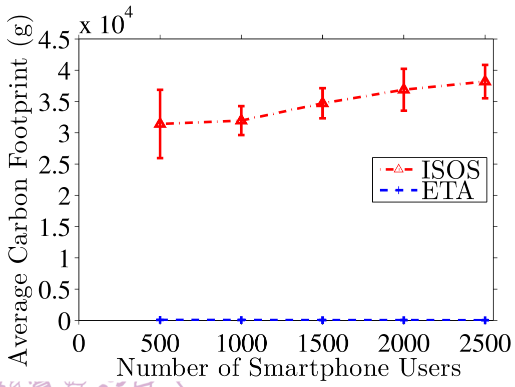


Figure 4.10: Average carbon footprint while varying number of smartphone users.

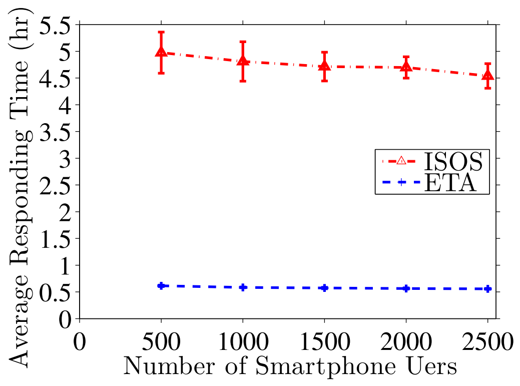


Figure 4.11: Average responding time while varying number of smartphone users.

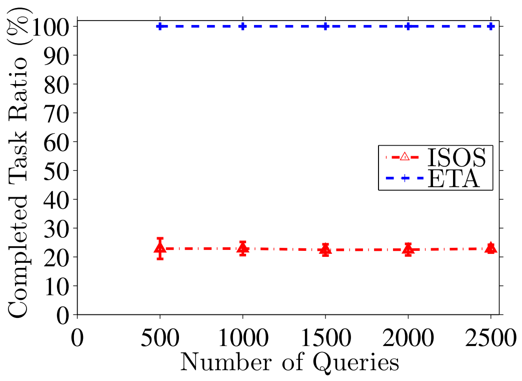


Figure 4.12: Completed task ratio while varying number of queries.

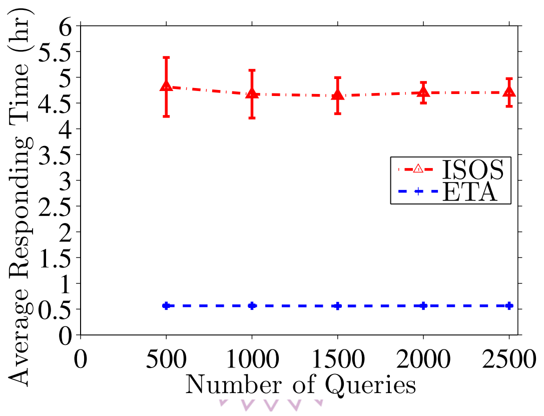


Figure 4.13: Average responding time while varying number of queries.

Chapter 5

Conclusion and Future Work

Context-aware apps are getting increasingly popular. Multiple apps may run at the same time on a smartphone and request for several overlapping contexts. In this thesis, we studied the problem of developing a green middleware for single smartphone to support efficient context inference in terms of energy consumption and accuracy. Moreover, we proposed a crowdsensing system which cooperating sensors on multiple smartphones. In Chap. 3, We presented the OSM middleware to selectively activate some sensors while considering the context sensing requirements from many context-aware apps. The OSM middleware is one of the first attempts to solve this problem in a coordinated and optimal manner. We also rigorously studied the scheduling problem–the core issue in the OSM middleware. We gave two optimization problem formulations: energy- and accuracy-optimization. We then proposed two optimal algorithms: EMA and AMA and two heuristic algorithms: EEMA and EAMA. We further extended the EEMA algorithm to EEMA* algorithm, which leverages the heterogeneous frequency and sampling rate for even higher energy saving. Our extensive trace-driven simulations show that: (i) EEMA/EAMA run in real-time, (ii) EEMA/EAMA achieve close-to-optimal performance, as small as $\sim 2\%$ and $\sim 1\%$ gap are observed, (iii) EEMA/EAMA/EEMA* lead to better performance with more contexts and combinations. We also implemented the OSM middleware and the EEMA/EAMA algorithms on Android phones. The real experiments show the practicality and efficiency of our solution.

We consider the extensive application in Chap. 4, we proposed the SAIS system. We mathematically formulated the task assignment problem and solve by an optimal algorithm (OPT) and an efficient task assignment algorithm (ETA). We implemented a simulator to evaluate the performance of ETA. Our simulation results show the proposed ETA algorithm achieves significant performance improvement: (i) up to 364 times improvement in carbon footprint, (ii) up to 8 times improvement in responding time, and (iii) only 2% gap in carbon footprint compare to optimal solution.

There are several possible future research directions. For example with single smartphone, large-scaled deployments on more diverse smartphone users will better quantify the potentials of the proposed green OSM middleware. Such performance improvement might not be huge at this point, but will certainly grow along with the increasing popularity of context-aware apps in next few years. Moreover, there are more extensive application about the usage of sensors such as data fusion [28], sharing context [3, 24], and *crowdsensing* [36].

Bibliography

[1] IPInfoDb. `http://www.ipinfodb.com/index.php`.

[2] User's guide, 66321B/D mobile communications dc source. `http://cp.literature.agilent.com/litweb/pdf/5964-8184.pdf`.

[3] V. Antila, J. Polet, A. Sarjanoja, P. Saarinen, and M. Isomursu. Contextcapture: Exploring the usage of context-based awareness cues in informal information sharing. In *Proc. of International Academic MindTrek Conference: Envisioning Future Media Environments(MindTrek '11)*, pages 269–275, Tampere, Finland, September 2011.

[4] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4), 2007.

[5] Y. Chon, N. Lane, Y. Kim, F. Zhao, and H. Cha. A large-scale study of mobile crowdsourcing with smartphones for urban sensing applications. In *Proc. of ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'13)*, Zurich, Switzerland, September 2013.

[6] Y. Chon, N. Lane, F. Li, H. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proc. of ACM Conference on Ubiquitous Computing (UbiComp'12)*, pages 481–490, Pittsburgh, Pennsylvania, September 2012.

[7] Y. Chon, E. Talipov, H. Shin, and H. Cha. Mobility prediction-based smartphone energy optimization for everyday location monitoring. In *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys'11)*, Seattle, WA, 2011.

[8] E. Chuvieco and R. Congalton. Application of remote sensing and geographic information systems to forest fire hazard mapping. *Remote Sensing of Environment*, 29(2):147–159, August 1989.

[9] Context-aware applications. what are they and how can they help your business? `http://tinyurl.com/kjqq2bo`.

[10] ContextMenus. http://developer.chrome.com/extensions/contextMenus.html.

[11] V. Coric and M. Gruteser. Crowdsensing maps of on-street parking spaces. In *Proc. of IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'13)*, pages 115–122, Cambridge, MA, May 2013.

[12] IBM ILOG CPLEX optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[13] R. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: Current state and future challenges. *IEEE Communication Magazine*, 49(11):32–39, November 2011.

[14] Gartner says context-aware technologies will affect $96 billion of annual consumer spending worldwide by 2015. http://www.gartner.com/newsroom/id/1827614.

[15] Gimbal. https://developer.qualcomm.com/mobile-development/mobile-technologies/context-aware-gimbal.

[16] GoogleMap. https://maps.google.com/.

[17] D. Hasenfratz, O. Saukh, S. Sturzenegger, and L. Thiele. Participatory air pollution monitoring using smartphones. In *Proc. of International Workshop on Mobile Sensing*, Beijing, China, April 2012.

[18] P. Holub, H. Rudová, and M. Liška. Data transfer planning with tree placement for collaborative environments. *Constraints*, 16(3), July 2011.

[19] J. Hsieh, S. Yu, Y. Chen, and W. Hu. Automatic traffic surveillance system for vehicle tracking and classification. *Intelligent Transportation Systems*, 7(2):175–187, June 2006.

[20] K. Kaer. A survey of context-aware middleware. In *Proc. of Conference on IASTED International Multi-Conference: Software Engineering*, Innsbruck, Austria, 2007.

[21] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. SeeMon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proc. of the International Conference on Mobile Systems, Applications, and Services (MobiSys'08)*, Breckenridge, CO, 2008.

[22] D. Kim, Y. Kim, D. Estrin, and M. Srivastava. SensLoc: sensing everyday places and paths using less energy. In *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*, Zurich, Switzerland, 2010.

[23] N. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha. Piggyback crowdsensing (pcs): energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities. In *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys'13)*, page 7, Rome, Italy, November 2013.

[24] J. Lee and U. Chandra. Mobile phone-to-phone personal context sharing. In *Proc. of IEEE International Symposium on Communications and Information Technology(ISCIT'09)*, pages 1034–1039, Incheon, Korea, September 2009.

[25] C. Liao and C. Hsu. A detour planning algorithm in crowdsourcing systems for multimedia content gathering. In *Proc. of Workshop on Mobile Video(MoVid'13)*, pages 55–60, Oslo, Norway, February 2013.

[26] C.-L. Lin. An Energy/Accuracy-Optimized Framework for Context Sensing on Smartphones. Master's thesis, National Tsing Hua University, Taiwan, 2013.

[27] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proc. of International Conference on Mobile systems, Applications, and Services (MobiSys'10)*, San Francisco, CA, 2010.

[28] J. Liu, E. Chu, and P. Tsai. Fusing human sensor and physical sensor data. In *Proc. of IEEE International Conference on Service-Oriented Computing and Applications (SOCA'12)*, pages 1–5, Taipei, Taiwan, December 2012.

[29] H. Lu, J. Yang, Z. Liu, N. Lane, T. Choudhury, and A. Campbell. The Jigsaw continuous sensing engine for mobile phone applications. In *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*, Zurich, Switzerland, 2010.

[30] Y. Ma, R. Hankins, and D. Racz. iLoc: a framework for incremental location-state acquisition and prediction based on mobile sensors. In *Proc. of ACM Conference on Information and Knowledge Management (CIKM'09)*, Hong Kong, China, 2009.

[31] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, H. Lu, S. Eisenman, X. Zheng, and A. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, Raleigh, NC, 2008.

[32] My Act. `http://xyo.net/android-app/my-act-cXMTrVs/`.

[33] S. Nath. ACE: exploiting correlation for energy-efficient and continuous context sensing. In *Proc. of International Conference on Mobile Systems, Applications, and Services (MobiSys'12)*, Low Wood Bay, UK, 2012.

[34] Nike. `https://play.google.com/store/apps/details?id=com.nike.plusgps`.

[35] V. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *Journal of ACM Computing Surveys*, 29(2), 1997.

[36] G. R.K., F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(32-39), November 2011.

[37] A. Saeed and T. Waheed. An extensive survey of context-aware middleware architectures. In *Proc. of IEEE International Conference on Electro/Information Technology (EIT'10)*, Normal, IL, 2010.

[38] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22(1), 1975.

[39] M. Schirmer and H. Höpfner. SENST*: approaches for reducing the energy consumption of smartphone-based context recognition. In *Proc. of International and Interdisciplinary Ionference on Modeling and Using Context (CONTEXT'11)*, Berlin, Heidelberg, 2011.

[40] M. Talasila, R. Curtmola, and C. Borcea. Improving location reliability in crowd sensed data with minimal efforts. In *Proc. of Joint IFIP Wireless and Mobile Networking Conference (WMNC'13)*, Dubai, United Arab Emirates, April 2013.

[41] P. Troubil and H. Rudová. Integer linear programming models for media streams planning. *Lecture Notes in Management Science*, 2011(3), August 2011.

[42] Ultra-low-power, context-aware motion-recognition platform to result from stmicroelectronics and movea cooperation. `http://www.st.com/web/en/press/t3468`.

[43] Y. Wang, J. Lin, M. Annavaram, Q. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proc. of International Conference on Mobile Systems, Applications, and Services (MobiSys'09)*, Kraków, Poland, 2009.

[44] G. Xing, R. Tan, B. Liu, J. Wang, X. Jia, and C. Yi. Data fusion improves the coverage of wireless sensor networks. In *Proc. of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom'09)*, pages 157–168, Beijing, China, 2009.

[45] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proc.of the International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, pages 77–90, San Francisco, CA, USA, June 2010.

[46] Z. Yan, H. Jeung, D. Chakraborty, A. Misra, and K. Aberer. SAMMPLE: Detecting semantic indoor activities in practical settings using locomotive signatures. In *Proc. of International Symposium on Wearable Computers (ISWC'12)*, Newcastle, UK, 2012.

[47] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In *Proc. of International Symposium on Wearable Computers (ISWC'12)*, Newcastle, UK, 2012.

[48] M. Yuen, I. King, and K. Leung. A survey of crowdsourcing systems. In *Proc. of IEEE International Conference on Social Computing (SocialCom'11)*, pages 766–773, Boston, MA, USA, October 2011.

[49] P. Zhou, Y. Zheng, and M. Li. How long to wait?: Predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'12)*, pages 379–392, Lake District, UK, June 2012.