University of California

Los Angeles

Sensing Across Mobiles and the Cloud: Architectural Styles and Software Mechanisms

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Haksoo Choi

2015

Abstract of the Dissertation

Sensing Across Mobiles and the Cloud: Architectural Styles and Software Mechanisms

by

Haksoo Choi

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2015

Professor Mani B. Srivastava, Chair

Recent changes in pervasive sensing applications require software systems that can address diverse technical, architectural, and human issues. Research on wireless sensor networks has served as technical fundamentals, mobile sensing research has helped solving many architectural problems, and now various human/cultural difficulties in pervasive sensing systems are revealed. We compare two contrasting architectural styles, i.e., the *cathedral* and the *bazaar*[1], and discuss the design of a system that *unifies* the both. Main challenges in designing such a system include: (1) a large amount of personal data; (2) privacy in sharing them; (3) energy-efficiency on mobile devices. We address them using a distributed network of virtually-private data stores featuring rule-based sharing control and flow-based execution of context inferences. Our performance benchmarks show that the rule processing delay is less than 25 ms in typical usage scenarios, and the flow-based execution saves 38.3% of CPU time as well as 54.3% of memory usage in comparison to a bus-based framework. Our twelve-person user study results indicate participants feel less privacy concerns using the rule-based sharing control. We also discuss an interesting tradeoff between usability and controllability, discovered from the user study. Finally, all source code for this research is readily available online[2].

[1]The analogy is inspired from a book authored by Eric S. Raymond, "The Cathedral and the Bazaar."

[2]`https://github.com/nesl/SensorSafe` and `https://github.com/nesl/FlowEngine`

The dissertation of Haksoo Choi is approved.

Mario Gerla

William J. Kaiser

Carlo Zaniolo

Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2015

little wings, her, and his guitar.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Prof. Srivastava and *NESL* members.

Prof. Cha and *Mobed* members.

Prof. Lee, Prof. Byun, and Dr. Rosa.

Friends at home.

Friends in Kwanjeong 8th and Ms. Kim.

Mr. Hwang, Daum Communication Corp., BK21 Foundation,
Korea Foundation for Advanced Studies, Yonsei International Foundation,
for helping me to keep studying.

Kwanjeong Educational Foundation for supporting a journey into myself.

Many musicians who helped me a lot in many ways during the journey.

My family for helping me to get out of the journey.

My wife for guiding me to finish the journey.

Little wings, for motivating me to start a new one.

Vita

2000-2007	*Bachelor of Science*, Computer Science, Yonsei University, Seoul, Korea.
2005,2007-2009	*Intern*, Samsung Electronics, Software Membership.
2006	*Exchange Student*, Computer Science, UC Davis.
2006	*Student Programmer*, Biological and Agricultural Engineering, UC Davis.
2007-2009	*Master of Science*, Computer Science, Yonsei University, Seoul, Korea.
2007-2008	*Intern*, Korea Electronics Technology Institute, Seoul, Korea.
2009	*Teaching Assistant*, Operating Systems, Yonsei University, Seoul, Korea.
2009-2014	*Ph.D. Student*, Computer Science, UCLA.
2011	*Teaching Assistant*, Embedded Systems, UCLA.
2012	*Intern*, IBM T.J. Watson Research Center, New York.
2012	*Advanced to doctoral candidacy*, UCLA.
2012, 2013	*Intern*, Qualcomm Corporate Research and Development, San Diego.

Publications

Haksoo Choi, Kwanghee Lee, Taehyun Kim, Changkyun Lee, Ho Seok Ahn, Minwoo, Lee, Won Il Choi, Kyu Chul Kyung, Joongkil Shin, Minhyuck Kim, *"Home Network System Based on Wireless Sensor Networks for Intelligent Service Robot,"* Korean Conference on Power Electronics, July 2007.

Kwanghee Lee, Kyu Chul Kyung, Haksoo Choi, Taehyun Kim, Changkyun Lee, Ho Seok Ahn, Minwoo Lee, Won Il Choi, Joongkil Shin, Minhyuck Kim, *"Chair Robot System Using PSP,"* Korean Conference on Power Electronics, July 2007.

Changkyun Lee, WonIl Choi, Minwoo Lee, TaeHyun Kim, Ho Seok Ahn, JoongKil Shin, KwangHee Lee, KyuChul Kyung, Haksoo Choi, MinHyuck Kim, *"Design of Intelligent Head Robot Using Rotating LED Array,"* Korean Conference on Power Electronics, July 2007.

Ho Seok Ahn, In-Kyu Sa, Young Min Beak, Kwang Hee Lee, Haksoo Choi, Joong Kil Shin, Kyu Chul Kyung, Minhyuck Kim, Byung Soo Lim, Jin Young Choi, *"Modular System*

Architecture of Intelligent Service Robots Focused on Smart Adapted Services," The 10th IEEE International Conference on Advanced Robotics (ICAR), August 2007.

Sukwon Choi, Hyojung Shin, Chanmin Yoon, Haksoo Choi, Hojung Cha, *"A Survey on Operating Systems for Wireless Sensor Networks,"* Communications of the Korea Information Science Society, December 2007.

Byunghun Song, Haksoo Choi, Hyung Su Lee, *"Surveillance Tracking System Using Passive Infrared Motion Sensors in Wireless Sensor Networks,"* The 22nd IEEE International Conference on Information Networking (ICOIN), January 2008.

Haksoo Choi, Chanmin Yoon, Hojung Cha, *"Device Driver Abstraction for Multi-threaded Sensor Network Operating Systems,"* The 5th European Conference on Wireless Sensor Networks (EWSN), February 2008.

Haksoo Choi, Sukwon Choi, Hojung Cha, *"Structural Health Monitoring System based on Strain Gauge Enabled Wireless Sensor Nodes,"* The 5th IEEE International Conference on Networked Sensing Systems (INSS), June 2008.

Kwanghee Lee, Byunghun Song, Haksoo Choi, *"Location Control Technique for Industrial Robots Based on RTLS,"* Korea Computer Congress (KCC), July 2008.

Chanmin Yoon, Haksoo Choi, Seungwoo Lee, Hojung Cha, Byunghun Song, Hyungsu Lee, *"Demo Abstract: IEEE 802.15.4a-Based Anchor-free Mobile Node Localization System,"* The 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Demo Presentation, April 2009.

Haksoo Choi, Nakyoung Kim, Hojung Cha, *"6LoWPAN-SNMP: Simple Network Management Protocol for 6LoWPAN,"* The 11th IEEE International Conference on High Performance Computing and Communications (HPCC), June 2009.

Uriel A. Rosa, Todd S. Rosenstock, Haksoo Choi, David Pursell, Christopher J. Gliever, Patrick H. Brown, Shrini K. Upadhyaya, *"Design and Evaluation of a Yield Monitoring System for Pistachios,"* Transaction of the American Society of Agricultural and Biological Engineers (ASABE), 2011.

Supriyo Chakraborty, Haksoo Choi, Mani B Srivastava, *"Demystifying Privacy In Sensory Data: A QoI based approach,"* The 3rd IEEE International Workshop on Information Quality and Quality of Service for Pervasive Computing (PerCom-IQ2S), March 2011.

Haksoo Choi, Supriyo Chakraborty, Zainul M Charbiwala, Mani B Srivastava, *"SensorSafe: a Framework for Privacy-Preserving Management of Personal Sensory Information,"* The 8th VLDB Workshop on Secure Data Management (VLDB-SDM), June 2011.

Supriyo Chakraborty, Zainul M Charbiwala, Haksoo Choi, Kasturi Rangan Raghavan, Mani B Srivastava, *"Balancing Behavioral Privacy and Information Utility in Sensory Data Flows,"* The Journal of Pervasive and Mobile Computing, Elsevier, 2012.

Haksoo Choi, Supriyo Chakraborty, Mani B Srivastava, *"Design and Evaluation of SensorSafe: a Framework for Achieving Behavioral Privacy in Sharing Personal Sensory Information,"* The 2nd IEEE International Symposium on Security and Privacy in Internet of Things (TrustCom-SPIoT), June 2012.

Haksoo Choi, Raghu Ganti, Mudhakar Srivatsa, Ramya Raghavendra, *"Content Dissemination Protocols in Hybrid Wireless Networks,"* The Annual Conference of U.S. Army Information Technology Agency (ACITA), September 2012.

Pandarasamy Arjunan, Nipun Batra, Haksoo Choi, Amarjeet Singh, Mani B Srivastava, *"SensorAct: A Privacy and Security Aware Federated Middleware for Building Management,"* The 4th ACM Workshop On Embedded Systems for Energy-Efficiency In Buildings (BuildSys), November 2012.

Chenguang Shen, Supriyo Chakraborty, Kasturi Rangan Raghavan, Haksoo Choi, Mani B Srivastava, *"Exploiting Processor Heterogeneity for Energy Efficient Context Inference On Mobile Phones,"* The 5th USENIX Workshop on Power-Aware Computing and Systems (HotPower), November 2013.

Chenguang Shen, Haksoo Choi, Supriyo Chakraborty, Mani Srivastava, *"Towards a Rich Sensing Stack for IoT Devices,"* The 33rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Special Session, November 2014.

CHAPTER 1

Introduction

Collaboration of mobile devices and cloud systems are popular in modern pervasive sensing applications. Mobile devices are ubiquitous, connected to the Internet, and used as platforms for personal data collection. Cloud systems have made it possible for such data to be almost permanently stored, shared across many entities, and used to generate more sophisticated information about individuals. Common use of terms such as Internet of Things (IoT) [AIM10], Cyber-Physical-Systems (CPS) [Lee08], and Machine-to-Machine (M2M[1]) reflects how pervasive such technologies are in our lives.

Systems for such applications have technical roots in Wireless Sensor Networks (WSNs) [Lew04], which can be considered as a micro-scale version of pervasive computing systems. From dust-sized wireless sensor nodes [WLL01] to structural health monitoring systems for Golden Gate Bridge [KPC07], WSNs started as autonomous networks collecting various sensor data through wireless channels. Then, it became large-scale systems such as environmental, seismic, and oceanic sensing/actuation systems. Now, the sensors are replaced by mobile devices, and the low-power wireless medium are replaced by the Internet.

Although diverse research has been conducted in WSNs, modern pervasive systems have their own problems, especially in software architectures and human-related problems. It is because: (1) the scale of the systems has been expanded; (2) the sensing target has changed from things to humans; (3) the nature of the collected data has become privacy-sensitive. Among many problems, we focus on data privacy, energy-efficiency, and large data volume.

First, one human-related issue is privacy of individuals sensed by machines, i.e., mobile devices and sensors embedded in infrastructures. There has been much news on privacy,

[1]`https://en.wikipedia.org/wiki/Machine_to_machine`

ranging from facetious to serious ones. An Android application, "Do you trust me?", transmitted precise location coordinates between romantic couples without any consent from each other. Later, the developers were indicted without physical detention due to violation of location information protection laws in South Korea [Lee11]. Moreover, if you use so-called "hacking" tools such as Tor [DMS04] and Tails[2], or merely visit *Linux Journal* website, your identity might be on a watch list maintained by U.S. National Security Agency (NSA) [GKO14]. Further, researchers were able to reidentify credit card users from anonymous transaction history of 1.1 million people in 10,000 stores over three-month period. Other major privacy incidents include reidentification of the Netflix dataset, patient health records from Washington State, taxi ride records from New York City, and so on [Sin15].

Second, energy efficiency while processing sensor data is also important because the most popular end-devices at the sensing side are typically battery-operated. Nowadays, there are growing demands on capabilities of mobile devices, but the battery technology is not keeping up. To overcome this problem, industries have designed flexible mobile CPUs using heterogeneous multiple processors and cores (HMPCores) on a single chip. Although the hardware seems ready, core operating system services are not fully optimized for such CPUs. In addition, energy efficiency in mobile devices better preserves privacy because more data can be processed within a user's own devices rather than someone else's cloud. In this way, a less amount of information instead of risky raw data needs to be shared.

Third, the amount of sensor data can become very large so architectures for pervasive computing systems should be carefully designed. Apparently, collecting data from many mobile devices and storing them in a centralized place is straightforward, but it will require a highly-scalable server infrastructure. However, we argue that if privacy is architecturally handled, the data size that a single host machine needs to handle is not large enough, so the typical performance of a home PC will suffice.

We discuss two broad architectural styles, i.e., a centralized *cathedral* and a distributed *bazaar*[3], and then introduce a design called *kazaar* that unifies the both. Briefly speaking,

[2]`https://tails.boum.org`

[3]The analogy is inspired from a book authored by Eric S. Raymond, "The Cathedral and the Bazaar."

the cathedral has problems with data privacy, but the bazaar is hard to manage. Our unified architecture remedies the manageability problem with virtual machines, and provides individuals with virtually-private data stores to preserve data privacy.

The main advantage of the cathedral style is manageability because an owner of a system has full access to user data and infrastructures hosting the data. Good manageability also helps earning trust from users because a well-managed system leads to better quality of services. However, it makes users give away their privacy easily in return for such services. Another disadvantage of the cathedral approach is scalability. Although modern data centers are highly scalable, they can face limitations once the amount of data reaches the capacity of addressable entities in the IPv6 environment.

Better privacy can be achieved by the bazaar because personal data are stored in private devices so users can have better control. Once data leave personal data stores, we can maintain traces of data movement and investigate if needed. Self-destruct data would be ideal, but without help from the core operating system services, it seems impractical. As a last resort, we also need help from privacy laws. On the other hand, the distributed architecture might scale in unmanageable way, but it could be the only choice for handling the amount of data produced in the IoT environment based on IPv6.

As mentioned before, if we just focus on privacy, the distributed style would be ideal, but the manageability is a problem. For this reason, many researchers have investigated if virtual machine infrastructures can help achieving both the manageability and the privacy. In this paper, we leverage virtualization techniques to provide users with *tangible* privacy experience through our *kazaar* architecture, which unifies the cathedral and the bazaar.

The kazaar architecture remedies the data scalability problem by distributing individual's data into their own storage. Ideally, the storage should be physically owned by data owners but it is impractical due to typical upstream limitations of residential Internet services. Thus, kazaar stores the data as close to owners as possible (e.g., regional data centers at each zip code). This will geographically distribute the data and give physical access to owners if needed. For this, we present *SensorSafe*, an implementation of virtually-private data stores, which is a core component of the kazaar architecture.

On the other hand, in order for mobile devices to distill raw sensor data into more abstract information, energy-efficiency is an important issue. Our design of *FlowEngine* employs flow-based programming model to manage data processing applications easily. This obtained manageability can be used by an operating system scheduler to execute them in energy-efficient manner, especially using the recent heterogeneous CPUs. We propose such a technique and discuss its feasibility.

The kazaar architecture was inspired by many existing works. Shilton et al. discussed social, political, and legal aspects of computer systems with personal data, and suggested expanding Codes of Fair Information Practice [SBE09]. Cranor et al. contributed numerous research works in *Usable Privacy and Security* area [CG05]. Closest to our work, Montjoye et al. presented a software framework called openPDS, where users can share only privacy-safe information instead of risky raw data [MSW14].

Key contributions of this work are as follows:

- We propose an architecture that gives tangible privacy experience, in that users can physically unplug and stay private.
- We provide a practical implementation of SensorSafe, a virtually-private data store in the cloud, and FlowEngine, an energy-efficient dataflow execution in mobile devices.
- We discuss our observations on usability vs. controllability tradeoffs, discovered from our user study.

Our performance benchmarks show that SensorSafe has an average latency of less than 25 ms on processing privacy rules in typical usage scenarios, and FlowEngine reduces 38.3% of CPU time as well as 54.3% of memory usage in comparison to a bus-based framework. Such results demonstrate that the proposed mechanisms are feasible. Moreover, our user study results indicate that users have less privacy concerns when they have more control over their data, which is provided by SensorSafe. We also discuss an interesting tradeoff between usability and controllability, discovered in the user study. Lastly, all software source code for this research is readily available online[4].

Rest of this paper is structured as follows. Chapter 2 compares two contrasting architectural styles, and propose an unified architecture called *kazaar*. Chapter 3 presents example applications used in this paper. Chapter 4 discusses the cloud component of our architecture, SensorSafe. Chapter 5 discusses the mobile component of our architecture, FlowEngine. Chapter 6 describes performance benchmark results of our implementation. Chapter 7 discusses results and findings from our user study. Chapter 8 provides an overview of the existing literature. Finally, Chapter 9 concludes this paper, discusses privacy principles that we have discovered, and provides a few ideas on implementating the principles.

[4]`https://github.com/nesl/SensorSafe` and `https://github.com/nesl/FlowEngine`

CHAPTER 2

Architectural Styles

In this chapter, we discuss several architectural styles for a computer system that flows sensory information among many users. We first discuss two contrasting architectures, i.e., centralized vs. distributed, and then propose an approach that unifies them. We also point out when either approach fails and why the unification is necessary.

2.1 A Centralized *Cathedral*

As shown in Figure 2.1, a centralized architecture consists of a single server and many clients (or users). Such a server is owned by a single entity, usually a commercial company. Sensory information from users is stored in the central server and can be retrieved by users or other entities from the server. Although we state there is *a* server, data centers usually operate many server nodes in coordination, giving users a perspective of interacting with a single server.

One strong advantage of the centralized architecture is simplicity. Making an architecture simple is especially important when capability requirements of a system are very high. It is because a system can scale easily (in terms of throughput, bandwidth, storage size, etc.) by simply adding more server nodes. In addition, an owner of the server infrastructure has a full access to every aspect of the system. This manageability leads to better service availability because technical issues can be well-handled with the full access. More importantly, the server owner also has an access to user data and auxiliary information such as system logs. Such data are useful in many ways, for example, providing personalized services or improving service qualities. From a user's perspective, good service quality often results in better

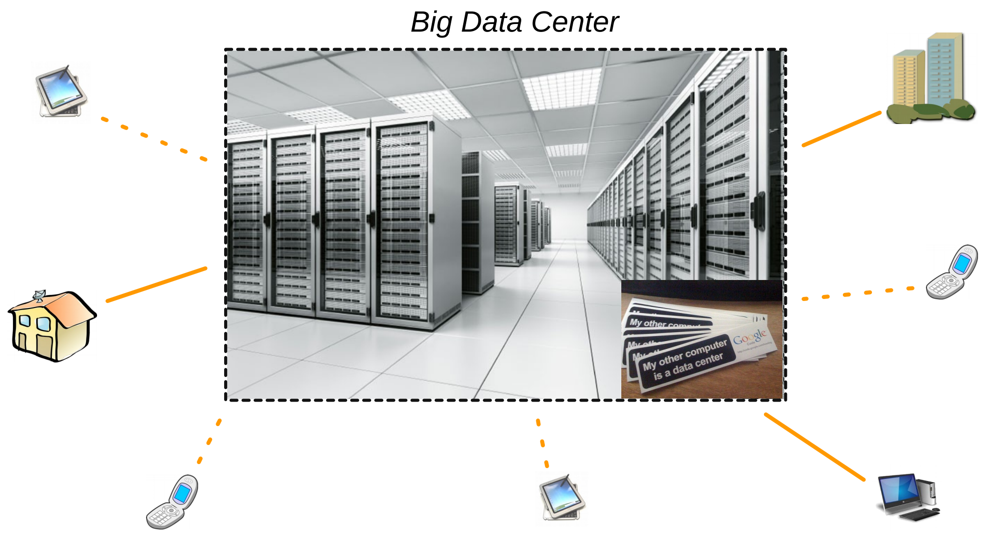


Figure 2.1: A centralized *cathedral* architecture.

trustworthiness, and most users are not much concerned about privacy once they trust someone.

However, it is interesting that users become indifferent to privacy due to the trustworthiness built upon their private information. We point out several problems related to privacy in the centralized architecture. First, ownership of user data is ambiguous because they are physically stored in a server owner's hardware while the data are about users. Second, the server owner potentially has unlimited access to private data unless proper security mechanisms are employed. Although the server owner could claim that user data are secure, users do not have a choice but to believe if the software is not open-sourced. In the worst-case scenario, even open-sourced software is not sufficient because physical ownership of data entails many side channels for illegitimate data access. Besides, data privacy is especially important in sensing applications because raw sensor data not only are sensitive but also can be used to infer more privacy-invasive information.

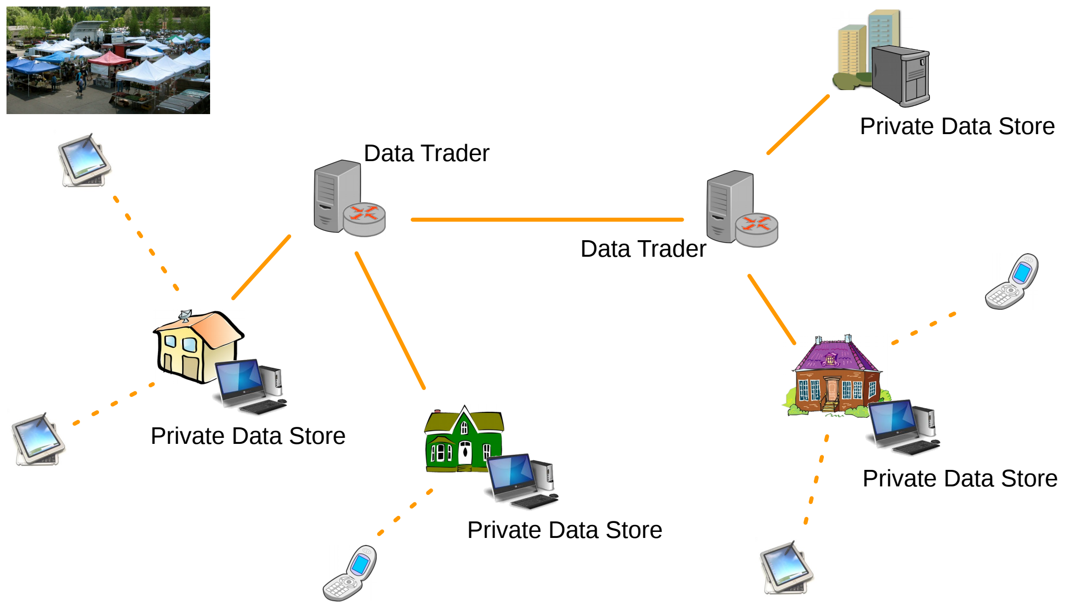


Figure 2.2: A distributed *bazaar* architecture.

2.2 A Distributed *Bazaar*

As the KISS[1] principle implies, we should always avoid a system becoming *unnecessarily* complex. However, we claim that it is reasonable to add *necessary* complexity for privacy because it is one of the basic human rights. In this section, we discuss whether a fully decentralized approach is indeed necessary, and see if it is feasible to build applications on top of such an architecture.

In a decentralized architecture shown in Figure 2.2, each user physically owns their data and hardware storing them. Commercial companies request user data every time they need them, and are not allowed to store them outside the personal storages. This restriction can be enforced by legal and/or technical measures. Data traders act as mediators, providing a kind of directory service for connecting data producers (e.g., users) and consumers (e.g., commercial companies). In this kind of architecture, it is clear who owns data and the data owners have a complete control over their data.

However, this architecture is impractical due to several reasons. First, typical upstream speed of residential Internet services is slow. It will cause intolerable service delay because

[1]Keep It Simple and Stupid (`https://en.wikipedia.org/wiki/KISS_principle`)

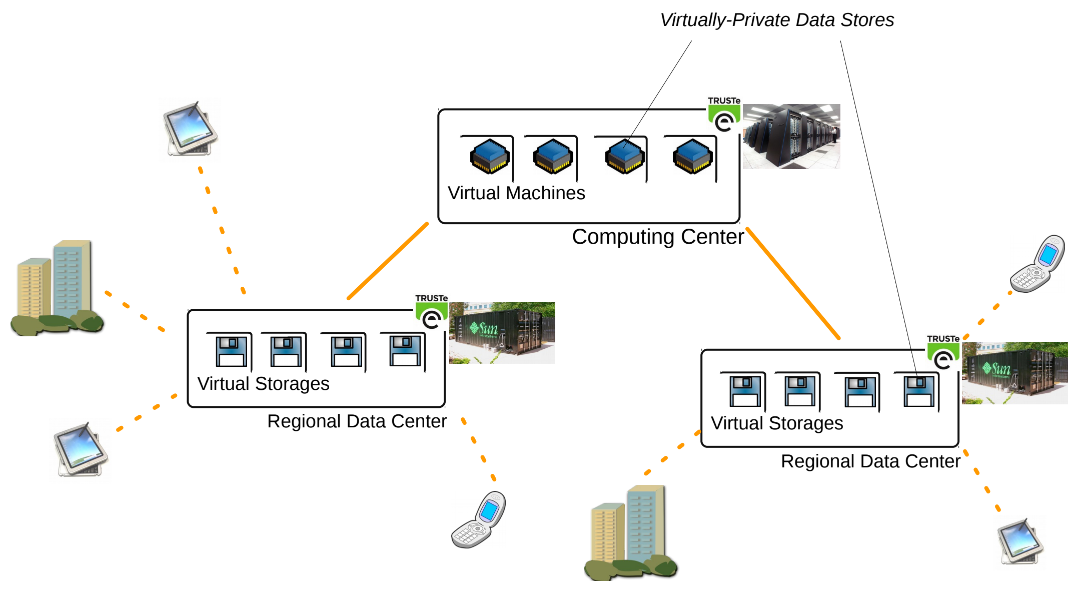


Figure 2.3: The unified *kazaar* architecture.

companies fetch user data only from the personal storages. Second, it might be impossible for non-technical users to run a PC at home all the time. Even if they manged to set up one, it would be challenging to keep the system up and running. Third, we cannot entirely rely on companies to exercise the no-store policy. Although privacy laws can help, technically enforcing the policy is more desirable. Lastly, distributed computation over wide area networks (WANs) is not trivial. In order to analyze data spread over the large networks (e.g., machine-learning over WANs), distributed computations are unavoidable because companies are not allowed to store personal data in their hardware. However, such distributed computations are not trivial in general.

2.3 A Unified *Kazaar*

We remedy the problems mentioned above by proposing a unified architecture that distributes user data over personal data stores, but still makes the centralized access feasible. An overview of the architecture is shown in Figure 2.3. The key idea is a network of virtual machines (VM) hosting virtually-private data stores, and a single VM instance contains its owner's data only. Data storages are purchased by the owners (e.g., portable solid-state drives), but the storages are commercially-operated at locations close to the owners. The close operation allows users to *unplug* their storage physically and go offline in case they are concerned. This architecture is a technically safer way to share personal data and also gives users tangible experience about their privacy. In addition, storing data in personally-purchased media gives users better legal rights, even though the data are used by third-party companies.

SensorSafe is a core component of the kazaar architecture. It is a framework for virtually-private data stores, which are distributed over the Internet. VM instances running the SensorSafe framework are hosted in regional data centers, and each SensorSafe framework stores a single user's private data. The regional data centers are connected to commercial companies' computing centers where they fetch personal data and process them. Because communication speed between the regional data centers and the computing centers are much faster than that of the residential Internet services, data are more accessible for central processing. For additional privacy guarantees, commercial companies operating the regional data centers can be certified by third-party companies such as TRUSTe [Ben99].

While SensorSafe is a backend framework in the cloud, FlowEngine runs on mobile devices to process personal sensor data energy-efficiently. Note that processing more data in a user's private device is also important for privacy because it reduces the risk of raw sensor data leakage. Figure 2.4 shows core components in SensorSafe and FlowEngine, along with how they are related to each other. We describe details of SensorSafe and FlowEngine in Chapter 4 and Chapter 5, respectively.

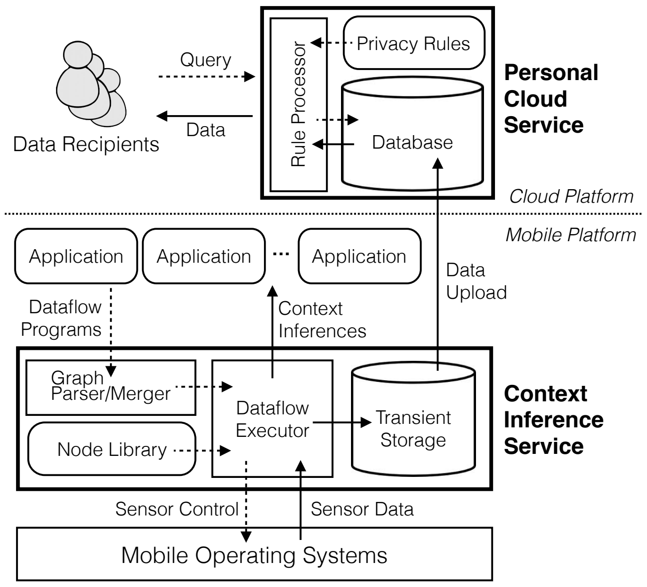


Figure 2.4: FlowEngine (Context Inference Service) and SensorSafe (Personal Cloud Service)

CHAPTER 3

Example Applications

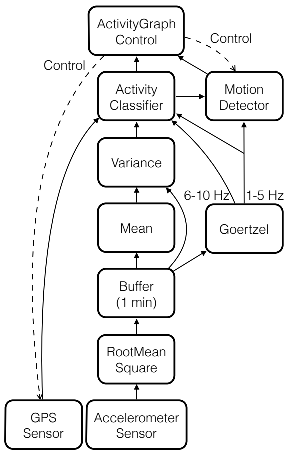
We introduce three context inference applications that help the discussion of our system throughout this paper. The applications include activity, stress, and conversation inferences, which have been presented in existing works. We have reimplemented them on FlowEngine to demonstrate energy-efficiency achieved by our the data-flow execution framework. The reimplemented dataflow programs are shown in Figure 3.1. We also briefly discuss several important characteristics that have guided the design of FlowEngine.

3.1 Activity

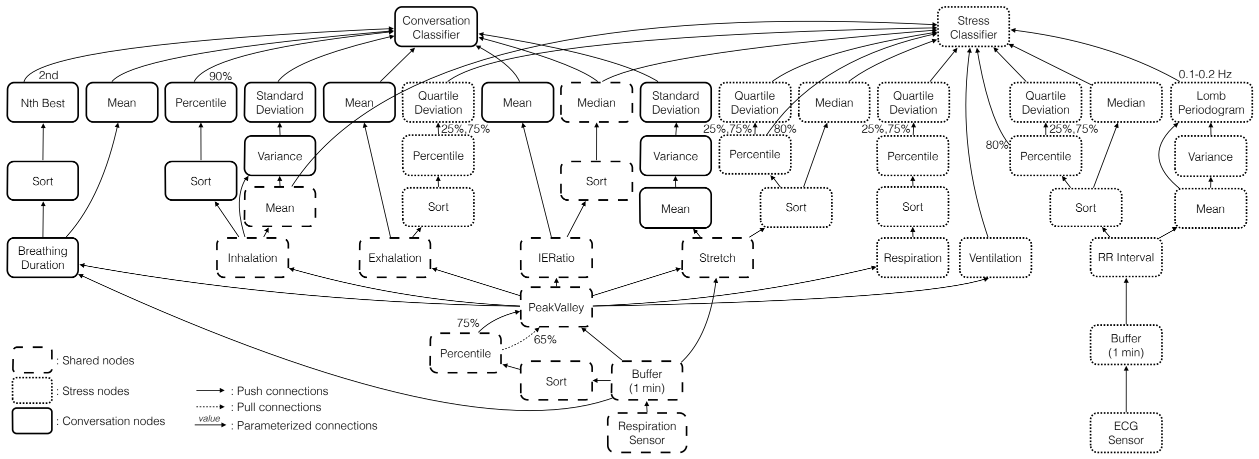
This application infers user's activities in five states, i.e., stationary, walking, running, biking, and driving [RMB10]. It uses accelerometer and GPS sensors on a smartphone. One interesting characteristic of this application is that it internally uses movement detection based on accelerometer to reduce usage of an energy-hungry GPS sensor. Depending on the result of the movement detection, subsets of its dataflow graph are activated or deactivated.

3.2 Stress and Conversation

These applications use ECG and respiration sensors from a body-worn chest-band [RBA10], and infer whether a user is stressed or not and speaking or not [RAP11]. The machine-learning algorithms for the two classifications have many common features computed from the respiration sensor. Such shared computations would have caused a waste of resources if there was no system support from FlowEngine. In addition, several features have common



(a) Activity



(b) Stress and Conversation

Figure 3.1: Dataflow programs of three example applications.

primitive computation stages, and they are also exploited by FlowEngine to reduce resource usage.

CHAPTER 4

SensorSafe: Virtually-Private Data Stores

This chapter introduces a concept of virtually-private data stores, SensorSafe, which is a foundation of the *kazaar* architecture. SensorSafe enables users to control how their data will be shared using expressive rules, and release data with differential-privacy in mind. We also talk about usability of rule-based sharing by comparing simple-but-easy and powerful-but-hard user interfaces. The database layer is also an important component of SensorSafe implementation, so we discuss how it has been evolved from the first version. SensorSafe has been field-tested, and the user study results indicate that participants feel more comfortable sharing their personal sensory information using SensorSafe as discussed in Chapter 7

4.1 Need for Privacy

Nowadays, mobile devices can collect personal and sensitive information about various aspects of our lives. They usually have GPS, WiFi/3G, and sensors such as an accelerometer, a gyroscope, a photodiode, a proximity sensor, etc. Such capabilities enable continuous collection of personal sensory information. Not only they can provide location and sensor data but also infer a user's context such as activities [KWM11], transportation modes [RMB10], and identities of speakers [LPL09]. Moreover, we can even obtain information about psychosocial status [PRH11] or social contexts [RAP11] if body-worn sensors (e.g., Autosense [ESK11], etc.) are used. Typically, machine learning algorithms are used by the context inferences, and classification models are trained using diverse features extracted from raw sensor data [JKM03].

This technology advancement is now pushing a new class of applications that involves *sharing* of personal sensory information. One example is Mobile Health [ES10], where medical sensors continuously monitor physiological signals on a user's body, and share the data with doctors, researchers, family members, or insurance companies [KAB09]. In commercial industries, Xively[1] is a popular web service for collecting, storing, and sharing sensor data streams. Foursquare[2] is also a prevalent web application for users to share location information with friends, and benefit from various location-based services.

Although these applications are promising, privacy implication behind sharing personal sensory information is a hindrance toward wide adoption. It is because even simple personal information can be easily correlated with privacy-violating facts [FWC10]. For example, UCLA's Center for Health Policy Research has found that there is a high correlation between where people live and certain diseases such as obesity and diabetes [Dri08]. World Tracker is a web service for locating a mobile phone just using phone numbers, and its popular application is tracking employees [wor]. An employee has been fired due to frequent breakaways from his workplace, discovered by using World Tracker without the employee's consent [Sei07]. Moreover, privacy concerns can be a hindrance to conducting pure research. For example, 200 people from a tribe had consented to provide DNA samples to "the study of the causes of behavioral and medical disorders". However, the researchers were sued later by the tribe because they used the DNA samples for a research related to schizophrenia [Har10].

We categorize privacy into several types: maintaining secrecy of information itself (confidentiality), hiding identity from information (anonymity), and protecting behavioral facts that can be inferred from sensory information (behavioral privacy). Existing security techniques can be used to provide the confidentiality, and various research efforts have been conducted to achieve anonymity of relational data (as opposed to sensory data) [FWC10]. Anonymity of sensory data is less applicable than that of relational data because many applications involving sensory information need users' identities to provide proper services. For example, mHealth services should be provided to a correct patient. Therefore, we claim that protecting the behavioral privacy is a key to success in such applications, and it become even

[1]`https://xively.com`
[2]`https://foursquare.com`

more important as users start to share their data with others. We attempt to achieve the behavioral privacy through SensorSafe.

Among many ways to preserve the behavioral privacy in sensory data, the following two approaches are promising: sharing the right amount of data (adequate sharing), and modifying data to make it hard to infer further information (data obfuscation). One of the mechanisms for the former approach is rule-based sharing where users define their own sharing rules, and the system enforces them when it disseminates data. The latter approach includes several classes of computational techniques such as generalization, perturbation, and transformation [GPT08, APG10, CRT06, CCC12]. Our work provides the rule-based sharing mechanism and a playground for experimenting the data obfuscation techniques. One of the challenges of SensorSafe is to process privacy rules efficiently because the amount of sensor data easily becomes large due to the continuous collection for a long duration. In addition, designing a flexible framework that can cover broad range of applications is also important.

There are many research efforts in systems for the behavioral privacy. Lockr [TSG09] provides an access control based on digitally signed social relationships. Persona [BBS09] also provides an access control with attribute-based encryption and out-of-the-band key exchange. Although the access control mechanisms are based on identity of data requester, but more fine-grained way of access control (i.e., rule-based sharing) is needed to achieve adequate sharing. Locaccino [TCH10] and Personal Data Vault (PDV) [MHM10] support defining fine-grained time and location conditions in sharing rules. PDV further provides automated recommendation of the rules and auditing of data dissemination. However, when sharing context information inferred from sensor data, rule-based sharing should be designed to have reasonable response time because an underlying database stores a large amount of sensor data.

SensorSafe supports adequate sharing through privacy rules with fine-grained conditions, which are applied to a large amount of sensor data. Our implementation of the rule processing has practical performance as discussed in Section 6.1. SensorSafe also provides differentially-private computation of statistical metrics (e.g., min, max, average, median, etc.) by adding the Laplacian noise [Dwo08]. We also discuss the effect of rule-authoring user interfaces and provide details in how the implementation of the SensorSafe design has been evolved.

4.2 Rule-based Sharing

SensorSafe is a personal cloud service for a user to store private data such as raw data samples from sensors or context inferences drawn from the sensor data. One of the main features is a *rule-based sharing*, which enables an owner to control how data are shared with other entities. In this section, we discuss the feature in details.

4.2.1 Privacy Rules

Rule-based sharing is a suitable mechanism for achieving adequate sharing. It is because each person has a very different sense of privacy, and rule-based sharing can provide personalized control of the amount of data that a user wants to share. In our system, users define their privacy rules specifying whether they want to allow or deny sharing based on conditions such as current contexts, locations, timestamps, data consumers, and sensor data themselves. Users can also choose to share but modify data to a certain degree (data obfuscation) so they can feel comfortable in sharing. As shown in Table 4.1, our privacy rule processor provides abstraction of location and timestamps to more general semantics, and raw sensor data to context labels. Figure 4.1 shows examples of privacy rules represented in JavaScript Object Notation (JSON) [Bra14].

Our privacy rules enable users to specify whether they want to allow or deny sharing their data based on various conditions: timestamp, location, sensor data values, and identities of data recipients. The condition can also include values across different data streams, which allow users to create rules based on context inferences. For example, if a user has an activity data stream that has a series of activity labels such as stationary, walking, running, biking, and driving, the user can specify a condition such as `activity = ‘driving’`. This condition can be applied to different data streams, for example, location or accelerometer data, so users can allow or deny sharing data collected when they are driving.

Table 4.1: Options supported by the rule-based sharing.

(a) Conditions and actions

Options		Attributes
Conditions	Consumer	User Name, Group Name
	Location	Label, Region
	Time	Range, Repeat (e.g., weekdays)
	Sensor	Sensor Name
	Context	(e.g., Moving, Not Moving, Still, Walk, Run, Bike, Drive, Stress, Conversation, Smoke)
Actions		Allow, Deny, Modify

(b) Examples of obfuscation options

Context	Options
Location	Coordinates, Street Address, Zip-code, City, State, Country
Time	Hour, Day, Month, Year
Activity	Accelerometer Data, Still/Walk/Run/Bike/Drive, Move/No Move
Stress	ECG/Respiration Data, Stressed/Not Stressed,
Smoking	Respiration Data, Smoking/No Smoking
Conversation	Respiration Data, Conversation/No Conversation,

```
{ target_users: [ researchers ]
  target_streams: [ Stress, ECG, Respiration ]
  condition: activity = 'driving' AND
             ( weekday(timestamp) = 0 OR
               weekday(timestamp) = 6 )
  action: deny
}
```

(a) "Do not share stress, ECG, and respiration data with researchers when I drive on weekends."

```
{ target_users: [ family ]
  condition: [ * * 9-18 * * 1-5 ]
  action: allow
}
```

(b) "Share all data collected in 9am-6pm on weekdays with family."

Figure 4.1: Examples of the privacy rules in JSON-like syntax. The repeating time syntax in Figure 4.1(b) is similar to the Linux *cron* [Nem10] time expression.

4.2.2 Rule Priorities

The rule processor translates rules to a database query before processing them. Therefore, rule processing is essentially merging query conditions of multiple rules. There is an interesting observation in this merging process in consideration of priorities. For example, let's say we have two rules: "Allow all data on my campus." and "Deny all data at a certain building on my campus". It is safer to interpret the rules as "Share all data on campus but not at the building", which is equivalent with assuming that the latter rule has a higher priority. If the former rule has a higher priority, then the interpretation will be: "Share all data on campus". We provide algorithms for merging rules with and without priorities.

Rules without priorities: We claim that more intuitive and safer interpretation of rules without priorities is taking union of data permitted by *allow*-rules and subtracting union of data prohibited by *deny*-rules from the union. That is,

$$\bigcup_{\forall i, D_i \in D_{allowed}} D_i - \bigcup_{\forall j, D_j \in D_{denied}} D_j$$

Rules with priorities: Interpretation of rules with priorities can be obtained by Algorithm 1. Let D_i a set of data affected by R_i, a rule with priority i. (Higher i means higher priority) Also, let K the result set after applying all rules. Note that, at line number 4, the deny-rule with priority 0 is ignored because higher allow-rules will override the rule making it has no effect on the result set K.

4.3 Differentially-Private Data Release

Although the rules with allow or deny actions enable users to control sharing based on various conditions, they only provide binary decisions: Share the data as they are or do not share at all. It is possible that users do not want to share raw data samples but feel comfortable sharing aggregate statistics over a certain duration. For example, let's say a user is collecting her daily step count data and wants to share the information with her friends or coworkers

Algorithm 1 Applying rules with priorities.

```
1: if $R_0$ is *allow* then
2:     $K \leftarrow D_0$
3: else if $R_0$ is *deny* then
4:     $K \leftarrow \emptyset$
5: end if
6:
7: for all $R_i$, increasing $i$ starting from 1 do
8:     if $R_i$ is *allow* then
9:         $K \leftarrow K \cup R_i$
10:    else if $R_i$ is *deny* then
11:        $K \leftarrow K \cap R_i^C$
12:    end if
13: end for
```

Table 4.2: Differentially-private aggregates supported by SensorSafe.

Aggregate Operator	Δf
average	$\lvert x_{min} - x_{max} \rvert / n$
sum	$\max(\lvert x_{min} \rvert, \lvert x_{max} \rvert)$
median	$\lvert x_{min} - x_{max} \rvert / 2$
minimum, maximum	$\lvert x_{min} - x_{max} \rvert$
first, last, n-th	$\lvert x_{min} - x_{max} \rvert$

as a motivation for them to do more exercise. However, she is a little concerned with sharing at each day level because she feels there is too much information. Instead, she would be happy to share the average step count over a week or a longer period.

To support such a user scenario, SensorSafe also allows users to create rules that share only aggregate statistics instead of raw data samples. As shown in Table 4.2, we currently support aggregate operators such as average, sum, median, minimum, maximum, first, last, and n-th over durations ranging from one minute to one year. Users can use the aggregate operators as actions for their privacy rules.

However, individual data samples can be easily revealed even with the aggregate values. To give a simple example, let's consider the above scenario that the user shares the average step count over a week or longer. An adversary can query her SensorSafe and obtain two average step counts, a_1 and a_2, for n days ($n \geq 7$) and $n + 1$ days (the same n days with one more contiguous day), respectively. The adversary can easily figure out the step count for the one additional day by calculating,

$$a_2(n+1) - a_1 n = \frac{\sum_1^{n+1} s_i}{n+1}(n+1) - \frac{\sum_1^{n} s_i}{n} n = s_{n+1}$$

, where s_i is a daily step count for a specific day. By querying as many times as needed, the adversary can potentially reconstruct the entire daily step count data at each day level.

To address the above problem, we have adopted a *differential privacy* [Dwo08] mechanism that adds random noise drawn from a Laplace distribution to aggregate values before releasing them. According to the formal definition of differential privacy in [Dwo08], for a certain privacy parameter ϵ, a randomized function Y satisfies ϵ-differential privacy if,

$$\Pr[Y(D) \in S] \leq \exp(\epsilon) \cdot \Pr[Y(D') \in S]$$

for all data sets D and D' that differ only in a single data sample, and all $S \subseteq \mathit{Range}(\mathcal{K})$. One such randomized function is to add Laplacian noise to the original query function $f(D)$. That is, $Y(D) = f(D) + Z$ where Z has the Laplace distribution with mean 0 and scale parameter $\Delta f/\epsilon$. In the scale parameter, Δf is the maximum difference in the values that f can generate on two data sets that differ only in a single data sample. Clearly, Δf depends on the type and the range of the function f, so we determine Δf for each aggregate operator we provide as follows.

Consider the data sets, D and D', and let x_{min} and x_{max} be a minimum and a maximum that the data sets can take, respectively. For the average operator f_{avg}, consider a case that D contains $n-1$ samples of x_{min}, and D' contains all samples in D and x_{max}, or vice versa. This is the case for Δf_{avg}, which is calculated by $|f_{avg}(D) - f_{avg}(D') = |x_{min} - ((n-1)x_{min} + x_{max})/n| = |x_{min} - x_{max}|/n$. For the sum operator f_{sum}, when D' has x_{min} or x_{max} in addition to D, $f_{sum}(D)$ and $f_{sum}(D')$ have the maximum difference. Thus, $\Delta f_{sum} = \max(|x_{min}|, |x_{max}|)$. For the median operator f_{med}, the maximum difference occurs when D has only one sample, either x_{min} or x_{max}, and D' has two samples, x_{min} and x_{max}. When there are the even number of samples, the median is defined to be the mean of the two middle values. Thus, in this case, $\Delta f_{med} = |x_{min} - x_{max}|/2$. For the minimum and the maximum operators (f_{min} and f_{max}), the maximum difference for Δf_{min} is caused when D has any number of x_{max}, and D' has all samples in D and x_{min}. Similarly, the maximum difference for Δf_{max} is caused when D has any number of x_{min}, and D' has all samples in D and x_{max}. In both cases, $\Delta f_{min} = \Delta f_{max} = |x_{min} - x_{max}|$. Lastly, the first, the last, and the n-th operators (f_{first}, f_{last}, and f_{nth}) behave similarly. The maximum difference occurs when D and D' differ in a single sample, so the samples at corresponding positions in D is x_{min} and D' is x_{max} or vice versa. In any case, $\Delta f_{first} = \Delta f_{last} = \Delta f_{nth} = |x_{min} - x_{max}|$.

As it is shown above, Δf is affected by minimum and maximum values that a certain data set can take. Theoretically, the minimum and the maximum values can be infinite, but practically they are bounded by the type of data sets and a user's characteristics. For example, a study shows that an average of daily steps taken by adults ranges from 5,000 to 10,000 [Par10]. However, the range would vary a lot depending on a user's occupation, for example, an office worker vs. a marathoner. Therefore, we believe it is reasonable to find the minimum and the maximum values from the personal data that has been collected for a certain user. Although the bounds would not be accurate at the initial data collection stage, they will gradually become more accurate as a user collects data.

We note that choosing the privacy parameter ϵ is not trivial because it affects privacy vs. utility tradeoff. In addition, the notion of a *privacy budget* [DMN06] should be also considered because every ϵ-differentially private query answer costs ϵ to the budget. In our implementation, we let users choose from several default values that are generally acceptable such as 0.01 or 0.1 [Dwo08].

4.4 Sensor Database and Privacy Rules

One approach to realize the privacy rule processing is placing a *filtering* layer on top of an underlying database system, similar to a network firewall. In this approach, unfiltered data are first retrieved from the database, and the filtering layer evaluates privacy rules on each data sample to determine whether each sample has to be blocked. Although there are techniques such as pre-compiled binary rules [EK96] for faster evaluation, a better approach is entirely avoiding the per-sample basis evaluations.

Therefore, we directly retrieve filtered data from the underlying database by including rule conditions in database queries (e.g., using the SQL WHERE clause). In this approach, the performance of applying privacy rules is affected by how sensor data are structured in the underlying database. In general, sensor data have timestamps associated with each data sample, and storing all the timestamps is a waste of storage space, especially when the data are sampled at regular intervals. Therefore, in the initial versions of SensorSafe [CCC11, CCS12], we adopted a concept of storing continuous sensor data in a series of *segments*. Each segment typically contains hundreds or thousands of data samples with a start timestamp and a sampling interval as metadata. The segment is the smallest unit of data that can be retrieved from the database.

The major limitation caused by the concept of segmenting was that the per-sample rule evaluation could not be avoided entirely. Specifically, consider a rule with conditions based on data values. Although we can filter out many segments through queries using per-segment statistics such as minimum and maximum, we still have to determine which individual samples *within* a segment are subject to such rules. It means that there are still rule evaluations on a per-sample basis.

Due to the problem, we redesigned our database layer using Informix TimeSeries[3], which is a relational database with extensions for time series data. Informix TimeSeries stores all samples in the same table row to achieve storage efficiency by avoiding duplication of redundant information, i.e., timestamps. One important feature provided by Informix TimeSeries is *virtual tables*, which provide a relational view of time series data. In other words, we can use standard SQL queries over the sensor data as if we have a relational table that contains individual sensor readings at each row, while timestamps and values are contained in distinct columns. Using the virtual tables, we can apply privacy rules by directly using the SQL WHERE clause. All rule conditions are carefully joined together by the algorithms presented in [CCS12], and inserted into a WHERE clause of a SQL query generated for a

[3]`http://www.ibm.com/software/data/informix/timeseries`

data requester. In this way, we can avoid evaluating rules for every data sample.

In the following section, we further discuss the database layer with implementation details. We also provide benchmark results of the rule processing using the time-series database in Section 6.1.

4.5 Implementation

In the first version of SensorSafe, we used a NoSQL database [HHL11], i.e., MongoDB [Cho13]. The main reason for choosing MongoDB was its faster performance in write operations than that of read operations. This feature was desirable because applications involving sensor data collection have much more write operations than read operations. Moreover, the write operations in NoSQL databases are typically faster than relational databases such as MySQL.

However, the concept of a single data record in MongoDB is a *document*, which is not suitable for storing continuous stream of sensor data samples. It is because the concept of a document is designed for storing a single large entity such as a blog article while a sensor sample is a small entity containing a timestamp and a value only. A stream of sensor data samples can be structured in multiple segments to fit the document concept as discussed before. However, such data model is not suitable for privacy rule processing because rule evaluations on a per-sample basis are unavoidable.

We have learned a lesson that it is important to choose a database that has the right data model, considering specific operations that will be performed on data stored using the model. In our case, Informix Timeseries has the better model than MongoDB because the nature of sensor readings are time-series data and the rule-processing is naturally supported by the SQL WHERE clause. Besides, the advantage of faster write operations in MongoDB is not useful because write operations are not frequent enough as SensorSafe only handles a single individual's data.

As mentioned before, SensorSafe was built on top of the Informix TimeSeries database[4]. We used a Java servlet container, Jetty[5], as a web service engine. We implemented RESTful APIs using Jersey[6], and used HTTPS to secure the API communications. Several authentication mechanisms were implemented such as OAuth [Ham10], API-Key[7], and Basic HTTP Authentication [FR14].

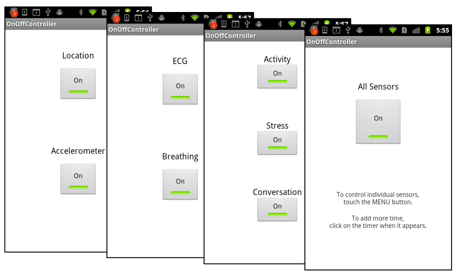
4.6 Rule-Authoring User Interfaces

When designing a user interface for managing privacy rules, it is important to consider usability vs. expressiveness tradeoff. Letting users directly write rules in the JSON format (shown in Figure 4.1) has the most expressive power because one can create complex conditions using arbitrary combinations of Boolean operators. However, it is obvious that general users without any programming experience would find it unusable. On the other hand, a user interface with a too simple design might not be able to expose the full capabilities of the underlying rule mechanisms. Therefore, we provide two kinds of GUI tools, On/Off Controller and Rule Manager, which have different usability and expressiveness tradeoffs as shown in Figure 4.2.

On/Off Controller has a set of simple switches. There is one switch for controlling all data at once, and several switches for individual sensors and context inferences. Whenever users feel uncomfortable sharing their data, they can simply press the appropriate switches. To prevent users from forgetting to switch back on, On/Off Controller asks users for an off-duration when they first press the switches. Users will receive reminders at five and ten minutes before the duration expires, and they can also extend the duration at any time.

Rule Manager lets users create privacy rules using various GUI elements. Users can choose what actions to take, which sensor data or context inferences to control, and whom to share with. Users can also create conditions based on time and location labels. A time

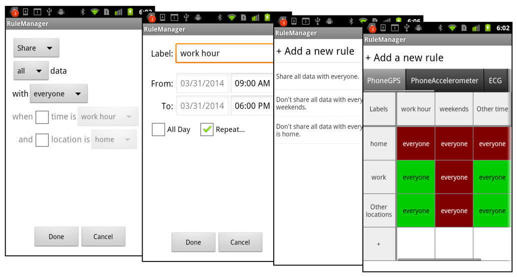
[4]http://www.ibm.com/software/data/informix/timeseries
[5]https://en.wikipedia.org/wiki/Jetty_(web_server)
[6]https://jersey.java.net
[7]https://en.wikipedia.org/wiki/Application_programming_interface_key



To control individual sensors, touch the MENU button.

To add more time, click on the timer when it appears.

(a) On/Off Controller



Share all data with everyone.

Don't share all data with every weekends.

Don't share all data with every is home.

Labels	work hour	weekends	Other time
home	🟥 everyone	🟥 everyone	🟥 everyone
work	🟩 everyone	🟥 everyone	🟩 everyone
Other locations	🟩 everyone	🟥 everyone	🟩 everyone
+			

(b) Rule Manager

Figure 4.2: Graphical User Interfaces for the sharing controls.

label can be a one-time duration or a repeating time, for example, every weekday from 9 AM to 6 PM. Users can define a location label by drawing a circle on a map. Both the On/Off Controller and the Rule Manager internally generate corresponding rules in the JSON format and store them in a user's SensorSafe.

On/Off Controller is very simple to use, but users might forget to press the buttons, leading to undesirable sharing of sensitive data. In addition, it only allows users to control sharing based on specific time durations. To solve this problem, we have developed Rule Manager. It has more expressive power than On/Off Controller. Users can create rules such as "Do not share my activity data at home during weekends." However, when a user has many rules, it could be hard to figure out under what conditions his/her data are shared or not. To help users understand their rules more easily, Rule Manager displays a table that summarizes how a user's data are shared based on time and location labels in a grid format as shown in Figure 4.2(b).

We have conducted a user study to understand how the rule authoring user interfaces affect users' privacy concerns in sharing personal sensory information. We have found that different GUIs change how much users feel comfortable in sharing such data. More details are discussed in Chapter 7.

CHAPTER 5

FlowEngine: Energy-Efficient Dataflows

In this chapter, we describe a mobile-side component of the *kazaar* architecture, called FlowEngine. It is a dataflow execution runtime and a programming framework for context-classifications on battery-operated devices. FlowEngine is useful not only for energy-efficiency but also for data privacy because personal data can leave private devices in more abstract forms. FlowEngine has been field-tested in our user study along with SensorSafe. We show how it reduces resource usage in comparison to a data-bus framework in Section 6.2. In order to further save energy, we propose a technique that exploits heterogeneous mobile CPUs by scheduling dataflow tasks and dynamically controlling voltages and frequencies of CPU cores.

5.1 Contexts by Systems

In modern pervasive applications, *sensing* of personal sensory information is an important characteristic. Mobile devices can continuously perform sensing, and the sensed data are used to infer a rich set of user contexts through machine learning algorithms. The sensory information is used in many applications, for example, social location sharing[1], mHealth [ES10], participatory sensing [BEH06], and medical behavioral studies [PRH11]. However, many challenges in the sensing aspect are still hindrances to wide adoption of such applications.

A key challenge in sensing on mobile devices is that they are resource-constrained in terms of CPUs, memories, and batteries. Current mobile applications utilizing context inferences

[1] `https://foursquare.com`, `http://www.getsaga.com`

involve heavy computation due to complex machine learning algorithms. Typically, individual applications directly process raw sensor data and make their own context inferences. However, if multiple applications make the same inferences, scarce resources are wasted due to redundant computation. Even if the multiple applications make different context inferences, they could still waste resources because it is possible to share common data processing stages.

To tackle the above challenge in sensing on mobile devices, we claim that context inferences should be provided as an operating system service. The system-managed context inferences have several advantages over application-managed inferences. A system-level service has a global view of how sensor data are processed, so it can optimize the computation needed to perform context inferences across multiple applications. Individual applications can also benefit from the service because they can now simply subscribe to the system-provided context data, which will free application developers from implementing complex data processing algorithms. Moreover, if core operating system services use the context information, there are opportunities for optimizing memory management, process scheduling, I/O, and security on mobile devices [CKL11].

One important consideration in designing the system-managed context inference service is that it should not limit an application's ability to use algorithms that are best suited for the application's requirements. It means that we need a flexible programming model for applications to specify their own context requirements. We have chosen the dataflow programming model [Mor10] because the typical context inference workload consists of multiple series of computations for data preprocessing, feature extractions, and classifications. As shown in Figure 3.1, dataflow nodes output their results to other nodes and take inputs from other nodes, forming a graph structure. This can be well captured by the dataflow programming model.

In several prior works, the dataflow programming model has been employed for execution of context inference algorithms [CKL11, CLL11, JLY12]. Compared to the existing works, FlowEngine further increases efficiency in resource usage through the following features: push/pull connection mechanisms, parameterized connections, subgraph activation/deactivation, and graph merging. While the existing works also provide a *push* mechanism for connecting dataflow nodes, but as we will discuss in later sections, our experience in developing several applications has revealed that not only a push but also a *pull* mechanism, and parameterized connections are useful for optimizing computation loads.

We summarize key contributions as follows. First, FlowEngine is a dataflow exeuction framework that features intuitive language, rich connections (i.e., push/pull and parameterized connections), subgraph controls, and graph merging. The features enable scarce resources on mobile devices to be efficiently used by multiple applications. Second, through the benchmark results, we show that FlowEngine can reduce 38.3% of CPU time and 54.3% of memory usage compared to an existing work based on a bus architecture. Third, FlowEngine has been field-tested in our user study with twelve participants for six days as discussed in Chapter 7. Finally, FlowEngine is open source and readily available online[2].

5.1.1 Dataflow Graph Language

It is important for a dataflow framework to provide flexible and intuitive programming language for developing dataflow graphs. Without such support, application developers have to write quite a bit of *plumbing code*[3] that might be too dependent on underlying details of a dataflow framework. Moreover, the language has to be simple and easy to learn so application developers can quickly prototype their own programs.

Therefore, we have designed a language that allows application developers to simply *declare* and *connect* dataflow nodes and develop their applications easily. Developers can use parameters at both *declaration-time* and *connection-time* to customize the ways nodes are instantiated and connected. Table 5.1 describes the syntax of our graph language for various types of connections.

Figure 5.1 shows examples of our language used in the example applications shown in Figure 3.1. Left columns are declaration of dataflow nodes, and right columns are connection

[2]`http://github.com/nesl/FlowEngine`

[3]Under-the-hood, low-level code that bridges applications and lower layers, which needs much effort of programmers.

Table 5.1: Dataflow language syntax.

Connection type	Syntax	Description
Push	`a -> b`	*a* pushes to *b*.
Pull	`a <- b`	*a* pulls from *b*.
Parameterized Push	`a -(value)-> b`	*a* pushes a parameterized result to *b*.
Parameterized Pull	`a <-(value)- b`	*b* pulls a parameterized result from *b*.

of the declared nodes. The Buffer node takes a *declaration-time* parameter for its size in terms of seconds. The Goertzel node connection in Figure 5.1(a) takes three *connection-time* parameters in the following format: `(start, end, step)`.

5.1.2 Push/Pull Connections

Basic behaviors of a dataflow node are taking input data, processing them, and producing outputs to other nodes. In our example applications shown in Figure 3.1, data originating from sensors go through many nodes as they *push* their outputs to other nodes. This basic behavior is supported by the push-connection mechanism.

However, it is possible that a dataflow node needs some of its inputs based on certain conditions. If we only have the push-connections, such conditional data should be always calculated by previous dataflow nodes because they do not have knowledge about when the data are needed. The lack of information causes a waste of resources because the conditional data are not always necessary. For example, the PeakValley node used in the stress/conversation application finds peaks and valleys in respiration sensor data. It dynamically adjusts a peak threshold value based on a default percentile value, but when it fails to find the proper number of peaks, it adjusts its threshold with a more relaxed percentile value. With the push-only model, both the default and the relaxed percentile values have to be always calculated by the Percentile node and pushed to the PeakValley node, causing an unnecessary computation.

To avoid such a situation, FlowEngine also provides a *pull*-connection mechanism. While the push connection is initiated by a node that produces data, the pull connection is initiated by a node that consumes data. The consumer node conditionally calls the pull interface so

```
Accelerometer acc       acc —> rms
RootMeanSquare rms      rms —> buf
Buffer buf(60)          buf —> goertzel
Goertzel goertzel       goertzel -(1.0,5.0,1.0)-> \\
GPS gps                   motion,activity
MotionDetector motion   goertzel -(6.0,10.0,1.0)-> \\
Activity activity         activity
ActivityGraphControl \\ activity -> control
  control(gps,motion)   motion -> control
.                       .
.                       .
```

(a) Activity

```
RespirationSensor res   res -> buf
Buffer buf(60)          buf -> sort
Sort sort               sort -> percentile
PeakValley pv           buf -> pv
Percentile percentile   percentile -(0.75)-> pv
BreathingDuration bd    pv <-(0.65)- percentile
Sort bdSort             pv -> bd
NthBest nth             bd -> bdSort
Conversation conv       bdSort -> nth
.                       nth -(2)-> conv
.                       .
                        .
```

(b) Conversation

```
ECGSensor ecg           ecg -> ecgBuf
Buffer ecgBuf(60)       res -> resBuf
RespirationSensor res   ecgBuf -> rrInt
Buffer resBuf(60)       rrInt -> rrIntSort
RRInterval rrInt        rrIntSort -> rrIntMedian
Sort rrIntSort          rrIntMedian -> stress
Median rrIntMedian      pv -> vent
PeakValley pv           vent -> stress
Ventilation vent        .
Stress stress           .
.
.
```

(c) Stress

Figure 5.1: Dataflow programs of the three applications.

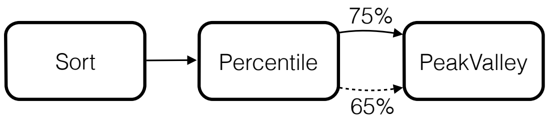


Figure 5.2: Push- (solid lines) and Pull-Connections (dotted lines).

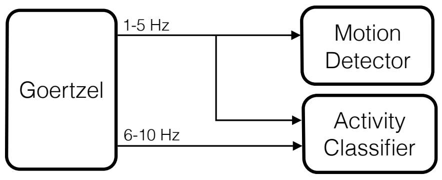


Figure 5.3: Parameterized Connections

the producer node can calculate required data only when they are necessary. Figure 5.2 shows how the PeakValley node is connected to other nodes using both the push- and pull-connection mechanisms.

5.1.3 Parameterized Connections

While typical dataflow nodes simply take inputs and generate a single result, multiple results can be generated based on different parameters. For example, results of a node that finds a certain percentile of its input data depend on specific percentage values. In the activity application, a node based on the Goertzel algorithm [Goe58] that calculates a power of a certain frequency from a discrete signal can generate multiple results based on selected frequencies.

A naive approach to such parameterized results is to create multiple instances of a node and let each instance generates a result based on a specific parameter. However, instantiation of multiple nodes that perform the same operation is a waste of the memory. A better way is to let a single node generate multiple results depending on parameters so that the unnecessary instantiation of nodes can be avoided. In FlowEngine, we support this through optional connection-time parameters in the push- and the pull-connections. For example, the Goertzel node shown in Figure 5.3 generates results based on ranges of frequencies so the connections take three parameters: start, end, and step frequencies.

5.1.4 Subgraph Activation/Deactivation

A dataflow program can conditionally choose different data-paths depending on current execution circumstances to conserve resources. For example, the activity application shown in Figure 3.1(a) uses GPS and accelerometer sensor data. However, the application tries to minimize the GPS usage because GPS consumes too much power [PKG10]. It is obvious that the GPS sensor is only useful when a user is moving. On that account, the activity application has an additional classifier that detects motion solely based on accelerometer, so the GPS can be turned off as much as possible.

To support such a behavior, dataflow nodes in FlowEngine can activate or deactivate portions of an entire dataflow graph. Such *control* nodes take input data from other nodes, decide which nodes to control, and call enable or disable interfaces on the target nodes. Once the control is initiated, FlowEngine traverses the graph toward source nodes to make sure all nodes and connections affected by the target node are appropriately controlled. For example, the ActivityGraphControl node in Figure 3.1(a) controls two target nodes: GPS and MotionDetector. Initially, the GPS node is disabled, and the MotionDetector node is enabled. When a user's motion is detected by the MotionDetector node, the ActivityGraph-Control node enables the GPS node and disables the MotionDetector node. After disabling the MotionDetector node, FlowEngine goes to the Goertzel node and disables all connections to the MotionDetector node. No further disabling is performed because the Goertzel node is still required by the ActivityClassifier node.

5.1.5 Graph Merging

Even though multiple dataflow programs perform different context inferences, they can have common nodes if they share the same sensors. For example, the stress and the conversation inferences shown in Figure 3.1(b) have several common nodes originating from the respiration sensor data such as Inhalation, Exhalation, IERatio, and Stretch. If we execute such dataflow programs individually, it would be a waste of CPU and memory because the common dataflow nodes will cause redundant computations generating the same outputs.

Therefore, FlowEngine checks whether multiple programs have common nodes and merges them to avoid redundant computations. Actually, Figure 3.1(b) is a result of merging the two individual programs shown in Figure 5.1(b) and 5.1(c).

One important consideration in merging common nodes is that merging decisions on individual nodes cannot be simply based on node types. Dataflow nodes with not only the same type but also the same data-path preceding the nodes should be merged together. It is because different instances of nodes with the same type can be used for different data-paths. For example, there are several instances of the Mean node in the stress and the conversation programs, but they calculate different mean values based on their input.

Therefore, each node instance in FlowEngine carries *preceding data-path information* that contains all parameters and connections between nodes on the data-path starting from source nodes. This information is used when FlowEngine receives a new dataflow program and examines each existing and new node to determine whether they can be merged.

5.1.6 Node Library

FlowEngine provides a library of dataflow nodes that can be used by application developers when they design their own dataflow programs. Currently, the library includes every node used in the three motivating applications so developers can quickly prototype their own applications using the library. However, it is hard to cover all dataflow nodes that can be potentially used by any applications, so we also allow developers to extend the existing library. All nodes in FlowEngine have a common parent class called DataFlowNode. Application developers can create a new dataflow node by extending the parent class, and then FlowEngine is able to recognize the new node as a valid dataflow node.

5.1.7 Implementation

FlowEngine is implemented using multiple Android services. First, we abstract sensor-specific communication details within *device driver* services. For example, a device driver service can simply call sensor APIs provided by Android SDK, or implement a complex

Bluetooth SPP[4] protocol for wireless sensors. Second, the main FlowEngine service executes dataflow programs submitted by applications. It obtains raw sensor data from the device driver services, processes them through dataflow graphs, and outputs final context classifications or intermediate results. Third, a separate service archives all data into a local storage and continuously uploads them to SensorSafe. Finally, applications subscribe to the context inference results provided by the FlowEngine service. All communication between the services and applications are implemented using Binder, and the interfaces are defined using Android Interface Definition Language (AIDL[5]).

5.2 Scheduling Dataflows using Heterogeneous Multiple Processors and Cores

As mentioned in Section 5.1, recent context-aware applications have high energy demand due to heavy CPU load caused by machine learning algorithms. Besides, typical mobile operating systems still ask users to charge on a daily basis. In order to remedy this, CPU industries have invented an architecture fabricated with heterogeneous multiple processors and cores (HMPCores) such as ARM big.LITTLE [Gre11]. However, software is yet to utilize its potential in full.

The HMPCore architecture essentially gives operating systems one more control *knob*, compared to existing CPU architectures. When the Dynamic Voltage and Frequency Scaling (DVFS) technique first emerged, it made the first knob available: core frequencies. Then, as Symmetric Multiple Processors (SMP) became popular, it enabled the second knob: the number of active cores. Lastly, the HMPCore architecture enabled the third knob: core types, which is a choice among cores with different power vs. performance tradeoffs. Figure 5.4 and 5.5 show an architecture diagram and the power vs. performance tradeoffs of two core types composing an HMPCore CPU.

The power vs. performance tradeoffs come from differences in core architectures. Specifi-

[4]Serial Port Profile

[5]`https://developer.android.com/guide/components/aidl.html`

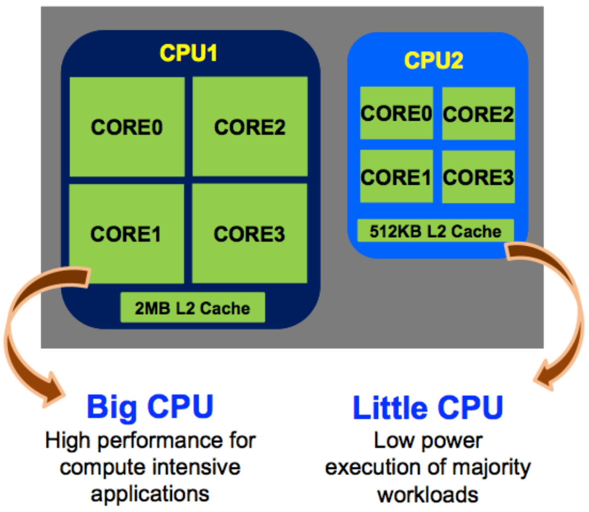

CPU1
CORE0
CORE2
CORE1
CORE3
2MB L2 Cache
CPU2
CORE0
CORE2
CORE1
CORE3
512KB L2 Cache
Big CPU
High performance for compute intensive applications
Little CPU
Low power execution of majority workloads

Figure 5.4: The SoC architecture of a HMPCore CPU [SLS13].

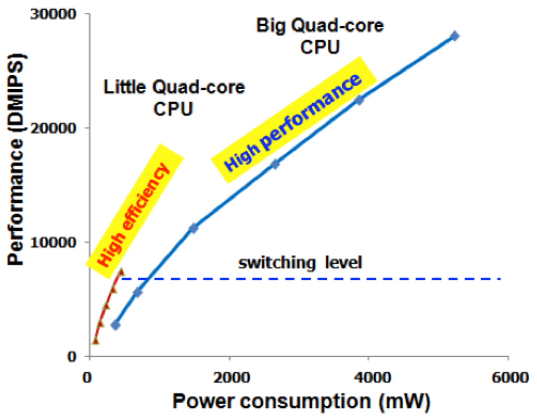


Figure 5.5: The power and performance characteristic of a HMPCore CPU [SLS13].

cally, the *big* cores employ a more complex architecture, i.e., more registers, richer instruction sets, deeper pipelines, bigger caches, etc., while the *little* cores employ a simpler architecture. Although there are many types of CPUs with such differences, the idea of using a system on a chip (SoC) consisting of dramatically different cores is a recent idea. First generations of such SoCs (e.g., TI OMAP 5 [Cum03]) coupled CPUs with different instruction set architectures (ISA) for more dynamics. However, such *loosely-coupled* SoCs suffer from binary executable incompatibility and cache inefficiencies while migrating tasks between cores. It also has limitations on shared memory performance while communicating between different CPUs. Although an OS dealing with such restrictions has been proposed [LWZ14], we should be careful about the benefits gained by having more complex OS designs.

Due to the disadvantages of the *loosely-coupled* architecture, more recent mobile SoCs are employing *tightly-coupled* designs (i.e., ARM big.LITTLE [Gre11]). These SoCs consist of cores with the same ISA but different pipeline depths, cache sizes, etc. The main advantage of the tight-coupling is minimal overhead in migrating tasks between different core types. The same binary executables can be used, and special features such as Cache Coherent Interface (CCI) preserve cache data during migration. One disadvantage is narrow dynamics in power vs. performance tradeoffs, but it enables simple operating system designs. In addition, the performance gain from simple design can exceed the penalty from using less *little* CPU.

While the hardware seems to be ready, software is still trying to fully utilize the three knobs mentioned above. The most recent industry practice is Global Task Scheduling (GTS) [Jef13], which is largely based on heuristic algorithms in task migration and scheduling decisions. Knowing the importance of keeping the core operating system services simple and fast, we explore potential applicability of an analytical approach for task scheduling and DVFS decisions.

One challenge is understanding of application execution patterns and accurately predicting application behaviors. It can be impractical to predict application behaviors precisely because mobile applications involve frequent interactions with humans. However, we observe that more applications are relying on background services with no user interventions, and such services tend to consume more energy than interactive tasks due to the high resource demand of machine-learning algorithms.

An example showing the importance of context data in mobile operating systems is the Apple M7 co-processor[6]. Although it is not as much flexible as the tightly-coupled architectures, it shows that system's support for context inferences is an important aspect of mobile operating systems. One interesting observation is that we have seen such co-processor integration in the history of CPU architectures. A Floating Point Unit (FPU) in addition to a main CPU was introduced to accelerate specialized computations. Now, we are witnessing similar changes in CPU architectures, except it is for energy efficiency. Although motivations are different (i.e., computation vs. energy efficiency), FPU is a successful example of a special-purpose hardware, which is a de facto standard for more than three decades[7].

In the following sections, we describe an analytical approach to the HMPCore scheduling problem. As mentioned before, we focus on context inference tasks, which are more tractable execution patterns than those of general user-interactive applications. We propose a scheme that works in following three steps: (1) it *learns* about timing and power characteristics of tasks and HMPCores, respectively; (2) it *determines* the optimal HMPCore parameters by numerically calculating them; (3) finally, it *acts* upon the decisions by proactively applying the parameters before executing the tasks.

5.2.1 Learning Dataflows and Cores

In order to find an optimal HMPCore configuration, we first obtain characteristics of dataflow tasks and cores as follows.

Timing Model Parameters

Typical dataflow tasks have repetitive execution patterns as shown in Figure 5.7. They periodically wait for a chunk of sensor data, process them, and wait for an another chunk. In addition, a context classification workload typically involves three stages: (1) data sampling;

[6] `https://en.wikipedia.org/wiki/Apple_M7`
[7] `https://en.wikipedia.org/wiki/Floating-point_unit`

(2) feature extraction; (3) classification. These tasks have different timing characteristics in terms of time taken for a one-shot task execution, T_{task}, and an interval between the executions, $T_{interval}$. Generally, the data sampling stage has short T_{task} and $T_{interval}$, while the classification stage has very long ones. Following list shows types of information needed for characterizing such tasks.

- $T_{task} = g(f, c)$: g is a function that maps an operating frequency of a core (f) and a core type (c, e.g., a big or a little core) to a one-shot task execution time, T_{task}.
- $T_{interval}$: an interval time between task executions.
- d: a total execution duration, e.g., a week.

The function g is introduced because T_{task} can vary depending on many factors. The most affecting ones are operating frequencies and core types as we can see in Figure 5.5. Other factors include input data characteristics and existing CPU load at the time of execution. We believe timing differences resulting from the input data characteristics are negligible and expect they are dominated by the operating frequencies and the core types. On the other hand, the existing CPU load at the time of execution will affect the timing characteristics a lot. We assume dataflow tasks will be executed on dedicated cores, so timing characteristics become more deterministic and analyzable. Although a scheduling algorithm with no such limitations is desirable, we claim that a few dedicated cores for dataflow tasks are practical considering the increasing number of cores in recent mobile SoCs (e.g., octa cores in Exynos 5 [SLS13]) and industry practices of coupling dedicated co-processors (e.g., Apple M7, TI OMAP 5, etc.).

The most precise way to implement the function g is direct measurements with real input data because the timing characteristics can be affected by the data. This approach can be cumbersome because it requires measuring all possible combinations of frequencies and core types. However, it can be automated easily and required only once. Obtaining CPU time information of threads is readily available through the *proc* filesystem in case of Linux-based mobile operating systems. In addition, $T_{interval}$ can be also easily obtained from the dataflow programs because they specify buffer sizes.

Power Model Parameters

Once we know characteristics of dataflow tasks, we need information about underlying CPU cores. Following is a list of parameters needed to calculate an optimal CPU configuration.

- $P_{active} = h(f, c)$: h is a function that maps an operating frequency of a core (f) and a core type (c, e.g., a big or a little core) to a power consumed when a core is active, which is P_{active}.
- P_{sleep}: A power consumed when a core is deactivated.
- T_{trans}: A time required to perform a power mode transition.
- E_{trans}: A total energy required to perform a power mode transition.

The function h could be constructed using information available in datasheets of SoCs, but it is better to measure the function directly due to the process variabilities[8] in silicon chips, which make power consumption different even in the exactly same models. Although the direct measurement is not feasible in most commercial off-the-shelf (COTS) products without using external equipments, more devices are becoming equipped with such capabilities [Lim13] (e.g., ODROID-XU[9], Qualcomm MDP[10], etc.)

The other parameters are usually available in datasheets. Otherwise, it should be measured beforehand. For example, according to measurements done by Ra et al. [RPK12] shown in Figure 5.6, a COTS smartphone (Samsung Focus SGH-i917) takes about 12 ms to wake up, and about 8 ms to sleep. The reported transition energy is $E_{trans} = E_{wakeup} + E_{sleep} = 3.065 \text{ mJ} + 3.101 \text{ mJ} = 6.166 \text{ mJ}$.

[8]`https://en.wikipedia.org/wiki/Process_variation_(semiconductor)`

[9]`http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127`

[10]`https://developer.qualcomm.com/mobile-development/development-devices/mobile-development-platform-mdp`

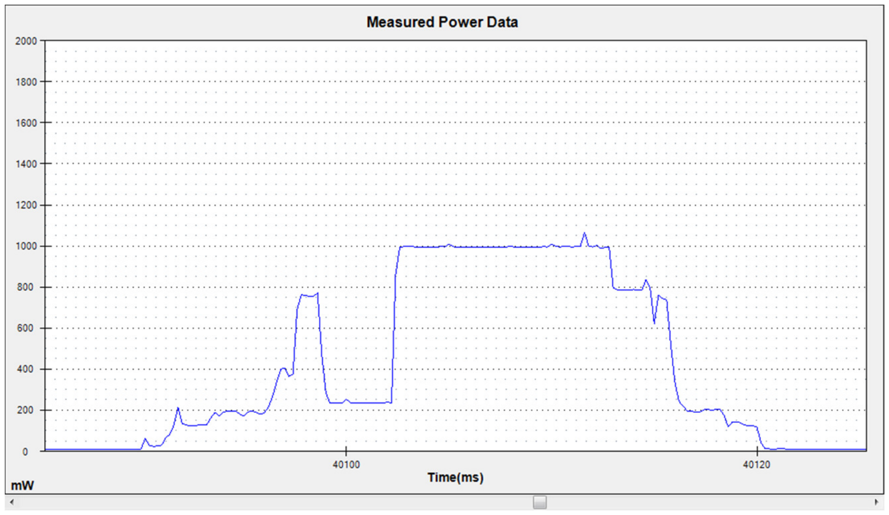


Figure 5.6: An example power trace of a COTS smartphone waking up, staying active, and going back to sleep. The measurements are reported by Ra et al. [RPK12].

5.2.2 Determining an Optimal Configuration

After learning the dataflow and underlying core characteristics, we can numerically determine the least power-consuming configurations for HMPCores. It is possible due to the deterministic nature of dataflow execution patterns. We predict energy required to execute the tasks in every possible way to assign tasks to multiple combinations of HMPCores, and pick an optimal one. Outputs of our algorithm include: (1) a selection of cores, which can be the same or different types; (2) mappings of multiple tasks (or threads) to the selected cores; (3) operating frequencies of the selected cores.

One interesting aspect of our algorithm is that it involves predicting a core's active/sleep pattern when multiple threads are running at the same time. It could have been inaccurate or infeasible to predict due to the complex nature of OS scheduling behavior. However, the behavioral simplicity of Completely Fair Scheduler (CFS) in Linux, which is just trying to mimic an *ideal* CPU[11], enabled the prediction feasible. As a result, execution times of multiple threads on a single core can be predicted by simply multiplying them by the number of threads. An overview of the proposed scheme is following.

1. Generate all possible ways to assign threads to HMPCores. It is done by an algorithm

[11]An *ideal* CPU: a CPU with no context-switch overhead.

that distributes l distinct tasks to m distinct processors, where each distinct processor p_i has its own n_i indistinct cores.

2. Given the dataflow/power model parameters and the task assignments, numerically calculate power consumption.

    - Decide whether each task assignment is actually schedulable considering the dataflow timing characteristics.
    - In case of multiple threads on a single core: expand overlapping executions by $T_{task} \times N_{task}$.

3. Pick an optimal HMPCore configuration, which is the least power-consuming task assignment, a selection of cores, and their operating frequencies.

5.2.2.1 A Task Assignment Algorithm

The goal of this algorithm is to find all possible ways to assign l distinct tasks to m distinct processors, where each processor has n_i identical cores. Note that one could have multiple identical tasks because different applications could execute the same dataflows, which is not desirable due to a waste of resources. However, a single system service such as FlowEngine can detect the redundant tasks and prevent them from being instantiated. On the other hand, one should be careful when deciding distinct tasks because the same tasks operating on different buffer sizes should be considered as indistinct tasks because their timing characteristics are different.

We define several terms as follows.

- Let T a set of l distinct tasks. $|T| = l$, where $T = \{t_1, t_2, .., t_l\}$
- Let P a set of m distinct processors. $|P| = m$, where $P = \{p_1, p_2, .., p_m\}$
- Let C_i a set of n_i indistinct cores in p_i, where $1 \leq i \leq m$. $|C_i| = n_i$, where $C_i = \{c_1, c_2, .., c_{n_i}\}$

Because we distribute tasks to processors and then cores inside the processors, the algorithm works in two steps. First, let's define a function $f : (T, P) \rightarrow F$ that distributes l distinct tasks to m distinct processors. Let F_P^T be a range set of f that takes two arguments. Each argument is picked from a processor set P and a task set T, respectively. F_P^T can be defined by a set of ordered pairs, i.e., tuples. Each position in a tuple represents a processor, and each element is a set containing tasks assigned to the processor. For example, $F_P^T = \{(\{\}, \{t_1, t_2\}, \{t_3, t_4\}), ..\}$. The first element in F_P^T means that no tasks are assigned on p_1, $\{t_1, t_2\}$ on p_2, and $\{t_3, t_4\}$ on p_3.

Second, for each tuple in F_P^T and each task subset T_i' in a tuple, we define another function $g : (T_i', C_i) \rightarrow G_i$, which distributes l' distinct tasks to n_i indistinct cores in a processor p_i. An example of a range set is $G_{C_i}^{T_i'} = \{\langle\{\}, \{\}, \{t_1, t_2\}, \{t_3, t_4\}\rangle, ...\}$, where $\langle ... \rangle$ means an $\emptyset$-*multiset*, a multiset that allows multiple appearances of empty sets but *not others*. The first element in $G_{C_i}^{T_i'}$ means that in a processor with four cores, two cores do not run any tasks, a core runs t_1 and t_2, and another core runs t_3 and t_4. Note the use of ordered pairs and $\emptyset$-*multiset* in the range sets of the two functions, f and g, respectively. It is because each position in an ordered pair has meanings while positions are not meaningful in a set. An ordered pair represents a task assignment on m distinct processors (which has different power vs. performance tradeoffs). An $\emptyset$-*multiset* represents a task assignment on n_i indistinct cores (which has the same power vs. performance tradeoff).

At this point, we have F_P^T, which is a set of all possible processor-level assignments. Let k_j be each tuple in F_P^T, and remember a position in the tuple represents a specific processor. For each element in a tuple k_j (in other words, for each task subset of each processor-level assignment), we have multiple sets representing core-level assignments, i.e., $G_1, G_2, ...G_m$, where G_i is a short for $G_{C_i}^{T_i'}$. Therefore, all assignments for a k_j (a single processor-level assignment) can be obtained by performing m-ary Cartesian product, i.e., $K_j = G_1 \times G_2 \times ... \times G_m$. Finally, we can obtain complete task assignments spanning from processor- to core-level as a set containing all K_j for each tuple k_j, i.e., $S = \{K_1, K_2, ...\}$.

A recursive algorithm for $f : (T, P) \rightarrow F$ is described in Algorithm 2. Again, F_m^T is a range, which is a set of tuples describing all possible ways to distribute l distinct tasks to

m distinct processors, where $|P| = m$. The $\oplus$ operator performs adding the left operand to the right operand while a new element in a tuple preserves corresponding positions. For example, $\{t_1\} \oplus (\{t_2, t_3\}, \{t_4, t_5\}) = (\{t_1\}, \{t_2, t_3\}, \{t_4, t_5\})$. C_i^T is a set of sets containing all possible ways to select the i number of tasks from T.

Algorithm 2 A recursive algorithm for $f : (T, P) \to F$, finding F_m^T. Note $F_1^T = \{(T)\}$

```
$S \leftarrow \{\}$
**for** $i : 0 \to l$ **do**
    **for** $C \in C_i^T$ **do**
        $S \leftarrow \left(\bigcup_{j=1}^{|C_i^T|} C \oplus a_j\right) \cup S, \quad \forall a_j \in F_{m-1}^{T-C}$
    **end for**
**end for**
$F_m^T \leftarrow S$
```

The above algorithm can be used without modification for obtaining all possible ways to distribute l distinct tasks in a task subset T' to n_i indistinct cores in C_i. However, redundant assignments will be generated as we distribute tasks to identical cores. One way to remove the redundant assignments is checking redundancy when a new element is added to the $\emptyset$-*multiset*. A more desirable way is to prevent the redundant assignments from generated in the first place by tweaking several parameters as follows.

- Let i starts from k to $\lfloor n/2 \rfloor$.
- If $c_1 < q$, then pass the inner for-loop, where q is the task with the smallest index number from the first upper level recursion.

The modified algorithm using the tweaks mentioned above is shown in Algorithm 3 as follows.

Implementation of $\emptyset$-*multiset* Data Structure

$\emptyset$-*multiset* is useful because it is a natural data structure to store distinct combinations of task distribution on same cores in a single processor. We provide a simple way to implement

Algorithm 3 Recursive algorithm for $g : T' \rightarrow C_i$, finding $G^{T'}_{n,k,q}$, the solution set. Note $G^{T'}_{1,\cdot,\cdot} = \{(T')\}$, and $s(C)$ is the task with smallest index number in $C \in C^{T'}_i$, where $C^{T'}_i$ is a set of $\emptyset$-*multiset*.

```
$S \leftarrow \{\}$
for $i : k \rightarrow \lfloor n/2 \rfloor$ do
  for $C \in C^{T'}_i$ do
    if $s(C) < q$ then continue with next $C$
    else $S \leftarrow \left( \bigcup_{j=1}^{|C^{T'}_i|} C \oplus a_j \right) \cup S, \quad \forall a_j \in G^{T'-C}_{n-1,i,s(C)}$
  end for
end for
$G^{T'}_{n,k,q} \leftarrow S$
```

the data structure using Java as shown in Appendix A. The `MultiEmptySet` class extends the standard Java `Set` class with the following details.

- Keeping track of the number of $\emptyset$ insertions in a variable `n`.
- Keeping track of the smallest number for task indices so the function $s(C)$ can perform in $O(1)$.
- A method `size()` returns `super.size()` $+ (n - 1)$
- Iterator extension: using the variable `n` above, it repeats $\emptyset$ for `n` times when the iterator encounters it.

Time complexity of $s(C)$ is $O(1)$ because the implementation of $\emptyset$-*multiset* internally maintains the smallest task index whenever a new element is inserted. Getting an element is also $O(1)$ because the underlying data structure is a hash map.

5.2.2.2 Checking Schedulability

Given the possible task assignments, not all of them can be actually schedulable because a core might be too slow to run many threads and meet the timing constraints. Therefore, we remove such assignments from further considerations. The schedulability check works as follows.

If there is only one thread on a core, the schedulability condition is simply $T_p \geq T_t$. When a core runs multiple threads, execution time T_{t_i} is extended in proportion to the number of simultaneous threads running. This is due to the multi-thread behavior of the CFS scheduler mentioned in Section 5.2.2. Note that not all tasks run at the exactly same time because they are periodic. A thread is schedulable if its extended execution time T'_{t_i} is shorter than its period.

We describe a way to calculate T'_{t_i}. Let $A = \lfloor T_{t_i}/T_{p_j} \rfloor \times T_{t_j}$, $B = min(T_{t_j}, T_{t_i} - \lfloor T_{t_i}/T_{p_j} \rfloor \times T_{p_j})$, and R_j as follows.

$$R_j = \begin{cases} \dfrac{A+B}{T_{t_i}}, & \text{if } T_{t_j} < T_{t_i} \\ 1, & \text{if } T_{t_j} \geq T_{t_i} \end{cases} \tag{5.1}$$

The extended task execution time T'_{t_i} in the worst case is following.

$$T'_{t_i} = \left(\sum_{i=1}^{n} T_{t_i} \cdot (R_i - R_{i-1}) \cdot (n+1-i) + \delta_{CFS} \right) + \delta_{CC} \tag{5.2}$$

Now we can determine the schedulability. Namely, if $T'_{t_i} \leq T_{p_i}$, then the thread is schedulable on this core with other threads.

Although CFS tries to mimic an ideal CPU, the context switch overhead cannot be avoided in practice. Therefore, we introduce variables δ_{CFS} and δ_{CC}, which are overhead caused by the CFS scheduler and cache coherency, respectively. The overhead variable δ_{CFS} depends on the number of concurrent threads because the more number of threads, the more context overhead. On the other hand, the δ_{CC} variable depends on whether a task's parent is assigned on the same core and the size of data it processes. The overhead is increased in the following order: (1) if its parent is on the same core, there is no overhead because they are using the same L1-cache; (2) if its parent is on the same processor but on a different core, there is an overhead caused by L2-cache coherency; (3) if its parent is on a different processor, there is an overhead from L3-cache coherency.

5.2.2.3 Calculating Power Consumption

After preparing a schedulable set of task assignments, we calculate anticipated power consumption for each assignment. There are two factors to consider: duty-cyclability of a task along with power-mode transition energy of a CPU core, and the number of threads running on a core. We first describe how to calculate the energy in case of a single thread.

Let $T_{interval}$ be time between two consecutive task executions, which can be calculated by $T_{period} - T_{task}$. If $T_{interval} \geq T_{trans}$, the thread can be duty cycled. However, it should be done only if there is energy saving. Therefore, we perform duty cycling only if the following equation holds true.

$$E_{trans} < T_{interval} \times P_{active} \tag{5.3}$$

If there is no benefit of duty-cycling or a task is not duty-cyclable (i.e., $T_{interval} < T_{trans}$), then the energy can be calculated as follows.

$$E = P_{active} \times d \tag{5.4}$$

A single thread's energy consumption when duty cycling is possible is following.

$$E = [E_{trans} + T_{task} \cdot P_{active} + (T_{period} - T_{task} - T_{trans}) \cdot P_{sleep}] \cdot d/T_{period} \tag{5.5}$$

When multiple threads are running on the same core, a core's activity pattern becomes irregular as shown in Figure 5.7. However, it is feasible to predict the activity pattern numerically because the CFS scheduler tries to mimic an ideal CPU. There is no context switch overhead in the ideal CPU so a thread's execution time can be simply multiplied by the total number of threads running on a core simultaneously. For example, let's say two tasks t_1 and t_2 have the same execution time of 10 ms when they run individually. An ideal CPU will finish both the tasks in 20 ms no matter how many times it switches between the two tasks.

Now we describe how to predict the irregular activity pattern. Figure 5.7 shows an

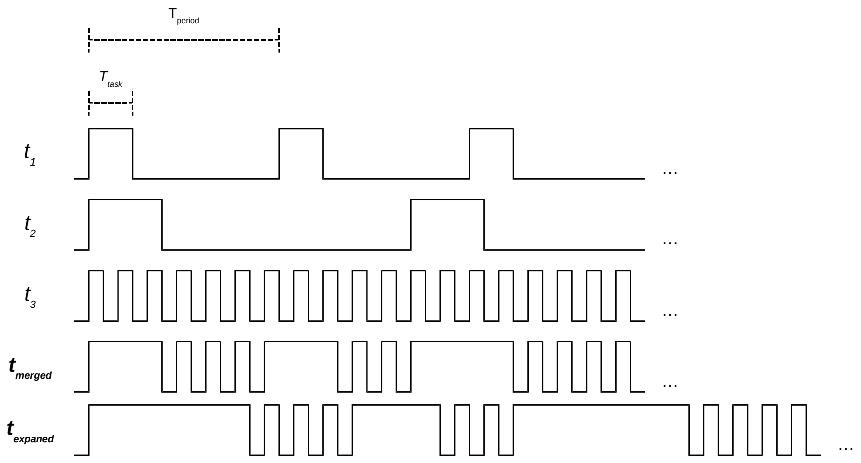


Figure 5.7: An example activity pattern of a single core running multiple dataflows.

example of constructing a merged activity pattern from three different threads. The pattern t_{merged} is a result of simply overlapping the activity patterns of t_1, t_2, and t_3. The overlapped regions need to be expanded by the number of threads active at the same time. The pattern t_{merged} is a result of the expansion.

To calculate the t_{merged} pattern, we use the worst-case extended execution time of tasks T'_{t_i} from the equation (5.2). The merged activity pattern should repeat every $P_{merged} = LCM(T^{t_1}_{period}, ..., T^{t_n}_{period})$, and the smallest time granularity of the merged activity pattern is $T_G = GCD(T^{t_1}_{period}, ..., T^{t_n}_{period})$. Each thread's execution pattern can be represented as a bit-vector, where each bit is the smallest time granularity T_G. Then, we can perform bitwise-OR of all bit-vectors, which will produce the final merged activity pattern.

Once we have the merged activity patterns, we can determine if each execution interval is worth switching the core into sleep mode by considering E_{trans} as mentioned in Section 5.2.2.3. Now we have the final activity pattern where unnecessary sleep intervals are removed. Using the final activity pattern, we calculate a power consumption for all combinations of operating frequencies and task assignments (Section 5.2.2.2) filtered by the schedulability check (Section 5.2.2.1). Finally, we simply pick an assignment with the least energy consumption.

5.2.3 Acting Upon the Decisions

Once we know the best task assignment and processor frequencies, the next step is acting upon the decision. In order to maximize the energy savings, we should ensure all tasks are executed with the optimal configuration. In our investigation to implement, we found that current Linux DVFS[12] mechanism is not in favor of granular control. According to the Kernel Documentation[13], changes of Operating Power Point (OPP) typically happens every 100 ms on general PC platform. In addition, DVFS control decision is mostly in reaction to various CPU load metrics. Main problems of the *reactive* mechanism include: (1) it can miss short bursts of task executions; (2) it can react too late. Our scheme needs to know when a task starts to execute precisely and *proactively* configure processors with the optimal settings. Therefore, it is necessary to implement this scheme in coordination with the scheduler.

In this paper, we discuss an implementation direction in details, and remain evaluation of the proposed scheme as a future work. We do not recommend modifying the scheduler to include code for CPU power control due to the following reasons. First, it mixes up orthogonal functionalities into a single software module. Second, the scheduler code has to be fast and efficient because every line of code in the scheduler constitutes to the context switch overhead. However, our scheme needs to know when a task starts precisely, to prepare a processor with an optimal status before it starts executing the task. Therefore, we recommend using *kprobes* [Kri05] to obtain a callback from scheduler code before it wakes up dataflow tasks. Inside the callback function, we can configure corresponding processors with an optimal setting. Using kprobes is especially useful because the callbacks run in the user space that is easier to debug, and it minimizes the amount of code added to the scheduler.

[12]Dynamic Voltage and Frequency Scaling

[13]`https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`

CHAPTER 6

Evaluation

We evaluate the implementation of the *kazaar* architecture in various aspects. First, performance benchmark results of the database layer in SensorSafe show that the rule-based access control over a large amount of sensor data is feasible. The rule processing has an average latency of less than 25 ms when SensorSafe has ten rules and serves a single data recipient. Second, we demonstrate resource-saving performance of FlowEngine in comparison to a data-bus framework, and explain why it is important to reduce computation redundancy in finer granularity. Our benchmark results show that 38.3% of CPU time and 54.3% of memory usage have been reduced by adopting FlowEngine. Third, SensorSafe and FlowEngine have been field-tested in our user study with twelve people for six days. We discuss how users' comfort levels have been increased by using our system.

6.1 SensorSafe Benchmarks

As described in Section 4.2, SensorSafe applies associated rules created by its owner whenever it performs a read operation for stored data. Therefore, it is important for the rule processing overhead to be reasonable because every read request is affected by the overhead. We conducted several experiments to demonstrate whether the overhead of the rule processing is reasonable by applying various numbers of rules. We measured latency and throughput across three dimensions: 1) the rule type (time- and value-based), 2) the number of rules, and 3) the number of simultaneous users requesting data.

All measurements were performed on a virtual machine running SensorSafe. The virtual machine stored personal data from a participant in the user study described in Chapter 7.

The virtual machine was hosted on our lab server with an Intel Xeon 2.7 GHz CPU and 64 GB memory. Each participant collected an average of 1.4 GB ($\sigma = 0.3$) of sensor data and context inferences, which included an average of 56.6 million samples ($\sigma = 13.6$ million) from the data collection for six days.

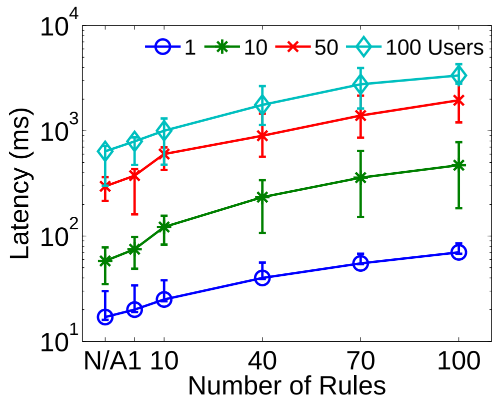
On a separate machine, we configured Apache JMeter [Hal08] to simulate requests from multiple simultaneous users. All physical machines in our experiments were connected through 100 Mbits/s LAN, and all requests generated by JMeter were authenticated using API-Key and secured with HTTPS. Because the number of rules created by each participant in the user study was not enough for measuring the rule processing overhead, we generated different numbers of time-based and value-based rules with allow and deny actions. JMeter created 1, 10, 50, and 100 threads simulating different number of simultaneous users. The number of rules was varied from 0 to 100.

In order to bring out the performance overhead caused by the rule processing layer, we deliberately queried only a small amount of data (i.e., less than 1 KB of sensor readings). It is because the latency and throughput measurement is dominated by network and disk access delays if we retrieve a large amount of data. Note that the overhead of rule processing layer becomes more negligible as the amount of requested data grows larger.

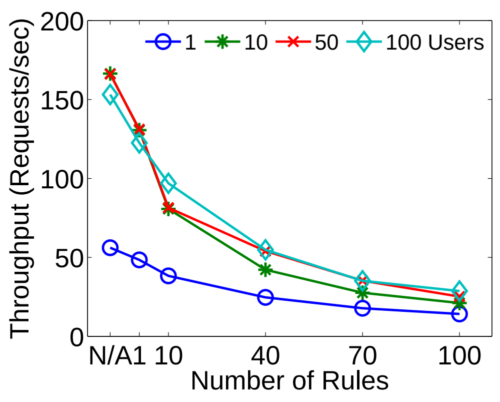
Figure 6.1 illustrates the benchmark results. Error bars in (a) and (c) depict minimum and maximum values. Mean, minimum, and maximum values were obtained from approximately 2,000 measurements for each experimental setting. The Y-axes in (a) and (c) are on logarithmic scales.

N/A in X-axes denotes that the rule processing layer does not exist at all in SensorSafe. By comparing the N/A case with only one (allow-everything) rule case, we can see the pure overhead of the rule processing layer itself, which is about 3 ms increase in the average latency and 7.7 requests/sec drop in the throughput.

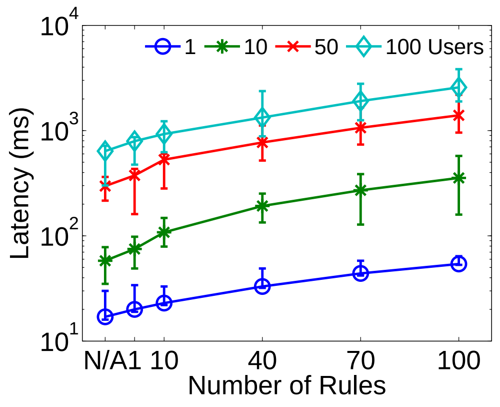
We can also observe that the query latency degrades as the number of rules and the number of simultaneous users are increased. Each time- and value-based rule adds about 0.5 ms and 0.34 ms, respectively, and each user adds about 6.2 ms of latency. The throughput is



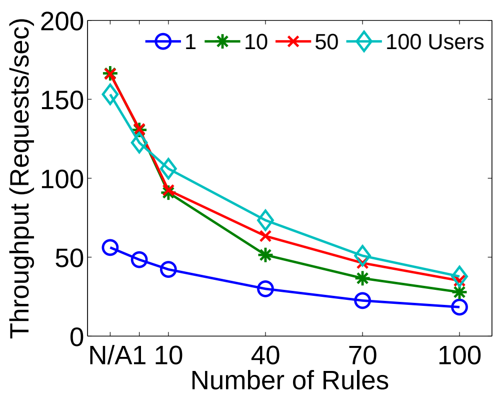
(a) Latency of time-based rules



(b) Throughput of time-based rules



(c) Latency of value-based rules



(d) Throughput of value-based rules

Figure 6.1: Benchmark results of SensorSafe.

saturated when the number of simultaneous users reaches 10, and the more number of rules reduce the system throughput. However, even with 100 users and 100 rules, SensorSafe can still support about 28.68 and 37.79 requests per second with time- and value-based rules, respectively.

6.2 FlowEngine Benchmarks

To evaluate how the dataflow-based context service achieves resource efficiency, we have reimplemented the stress and conversation applications based on the existing framework [ESK11], called mStress, by applying our flow-based context framework.

Similar to FlowEngine, mStress also reduces redundant computation through *buses*, but the level of computation sharing is coarser. The mStress framework is built using four buses, i.e., Packet Bus, Window Bus, Feature Bus, and Context Bus. Any computation nodes can publish data and subscribe to a bus. Computation sharing is achieved by these shared buses because once a node publishes a computed result to a bus, the result is sent to multiple nodes subscribing to the bus. However, the level of sharing is fundamentally limited by the number of the buses. To overcome this limitation, the mStress framework also allows nodes to publish data to the same bus the nodes subscribe to, essentially increasing the level of computation sharing. However, one cannot arbitrarily increase the level of sharing by republishing new results on the same bus, because the bus will become too busy, causing overhead for subscribed nodes to filter out unwanted data.

By using FlowEngine, we were able to achieve more fine-grained sharing of computations. Figure 6.2 shows CPU time measured for each dataflow node. FlowEngine separated out several primitive computation stages, i.e., Sort, Variance, and PeakValley. Specifically, the Sort operation was redundantly performed by the Median and the Percentile nodes, but using FlowEngine, the result from the Sort node is now shared between the two nodes. The Mean and the StandardDeviation nodes were computed independently, but we separated out the Variance node so the intermediate results from the Mean and the Variance nodes can be shared among the StandardDeviation and the classifiers nodes (Stress and Conversation). Separating out the PeakValley node was especially beneficial because it was consuming relatively large CPU time and was redundantly computed in many respiration feature nodes, i.e., Ventilation, Inhalation, IERatio, Stretch, BreathingDuration, Respiration, and Exhalation. After separating out the PeakValley node, CPU times consumed by the respiration feature nodes were reduced significantly. Consequently, we were able to reduce the total CPU time needed for one stress and one conversation context inference by 38.3% and the memory usage by 54.3% as shown in Table 6.1. Although several nodes in Figure 6.2 show a little overhead incurred by FlowEngine, they are largely compensated by savings achieved by reducing redundant computations. Note that, in Figure 6.2, the primitive computation stages (i.e., Sort, Variance, and PeakValley) do not have measurements in case of mStress because they are mixed-up with other computations.

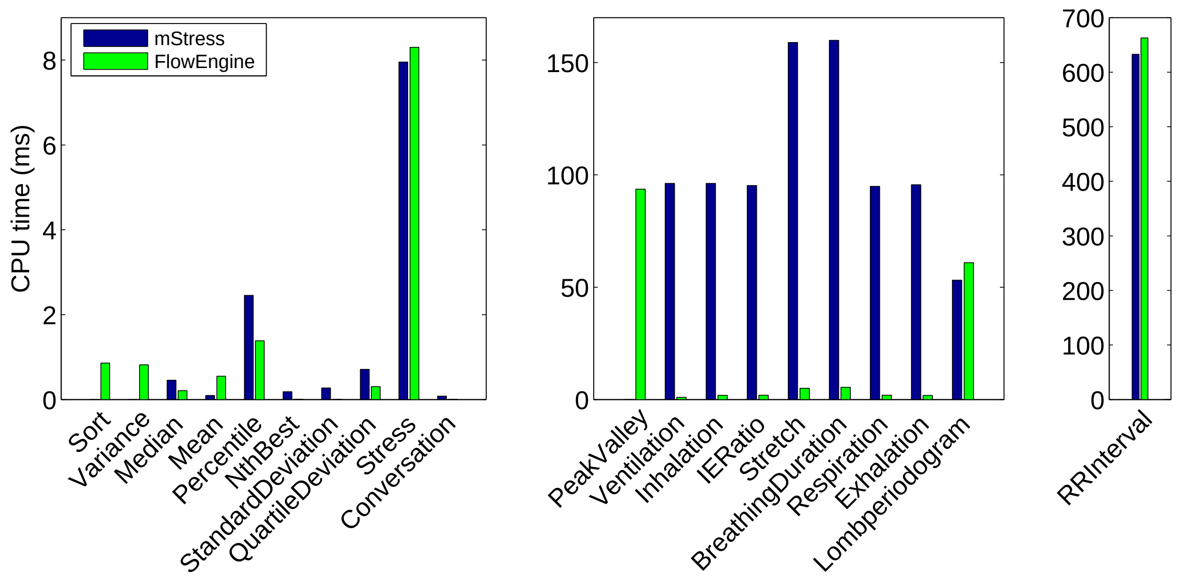


Figure 6.2: CPU times of individual dataflow nodes in the stress and conversation program.

Table 6.1: Total CPU time and memory usage of FlowEngine.

	mStress	FlowEngine	% reduced
CPU Time	1503.9 ms	927.8 ms	38.3 %
Memory Usage	36.8 MB	16.8 MB	54.3 %

In the mStress framework, it might have been possible to achieve the similar degree of computation sharing by republishing intermediate results on the same bus. However, it would have caused non-negligible overhead on the bus communication as mentioned before. More importantly, we claim that the dataflow framework is a more natural way to support fine-grained computation sharing than the bus-based framework because computations can be shared arbitrarily in dataflow graphs without such an overhead. Besides, the dataflow framework enables the intuitive programming language discussed in Section 5.1.1

CHAPTER 7

User Study

Through our user study[1], we aim to answer the following questions: (1) Do users feel more comfortable with sharing when they can control how much to share? (2) Do users gradually feel more comfortable as they have more controllability? We first describe the study design and discuss the results in the following sections.

7.1 Design

We recruited twelve participants for the user study, of which ten were graduate students and two were housewives. Their ages varied from mid-20s to mid-30s. We gave them smartphones and sensor chest bands to collect sensor data for six days. The participants were asked to carry the phones and wear the chest bands during their normal hours. They were instructed to wear the equipments in the mornings and take it off at nights.

The study phones had the three context applications mentioned in Section 3, which were developed using FlowEngine. The phone continuously collected accelerometer and location data, which were used to infer activities such as stationary, walking, running, biking, and driving. The phone also collected ECG and respiration data from the chest band, which were used to infer stress and conversation events.

The context service collected the raw sensor data and the inferences in a SD card, and opportunistically uploaded them to a SensorSafe VM. We set up one SensorSafe VM for each participant, twelve VMs in total.

[1]All user study protocols have been approved by the Institutional Review Board at University of California, Los Angeles (IRB#14-000124).

7.1.1 Study Phases

The study duration of six days was divided into three phases (two days per a phase) to compare how privacy concerns change with different degrees of controllability.

- Phase 1: The participants did not have any control over sharing, so they had to share all data collected.
- Phase 2: We installed the On/Off Controller in the study phones and instructed them to use the application whenever they felt uncomfortable with sharing their data.
- Phase 3: We additionally installed the Rule Manager on the study phones. The participants were able to control sharing with expressive privacy rules with various time and location conditions.

We made sure that the participants understood how to use RuleManger by giving a 10 minute instructional session and letting them create example rules.

7.1.2 Data Reviews

To help the participants understand what kind of sensory information were being collected, and how their sharing control was working, SensorSafe automatically generated web pages every day as shown in Figure 7.1. The red location points on the map and the red background in the plots indicate not-shared data. The blue location points and white background indicate shared data. (In this paper, we intentionally blurred the map image to protect the privacy of the participants).

7.1.3 Surveys

At the end of each phase, we requested participants to take our online privacy surveys. The questionnaire asked the participants to rate how much comfortable they felt with sharing their data on a seven-point Likert scale: very uncomfortable (1); neutral (4); and very comfortable (7). Each questionnaire consisted of seven sections. Section 1 through 6 asked

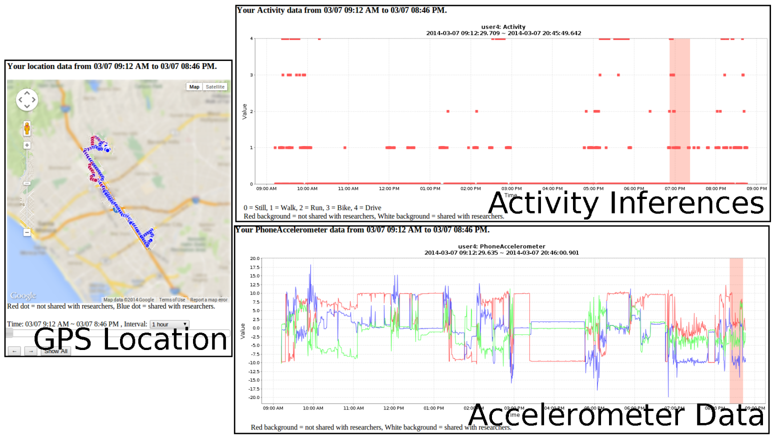

Your location data from 03/07 09:12 AM to 03/07 08:46 PM.
Red dot = not shared with researchers, Blue dot = shared with researchers.
Time: 03/07 9:12 AM ~ 03/07 8:46 PM , Interval: 1 hour
GPS Location
Your Activity data from 03/07 09:12 AM to 03/07 08:46 PM.
user4: Activity
2014-03-07 09:12:29.709 ~ 2014-03-07 20:45:49.642
0 = Still, 1 = Walk, 2 = Run, 3 = Bike, 4 = Drive
Red background = not shared with researchers, White background = shared with researchers.
Activity Inferences
Your PhoneAccelerometer data from 03/07 09:12 AM to 03/07 08:46 PM.
user4: PhoneAccelerometer
2014-03-07 09:12:29.635 ~ 2014-03-07 20:46:00.901
Red background = not shared with researchers, White background = shared with researchers.
Accelerometer Data

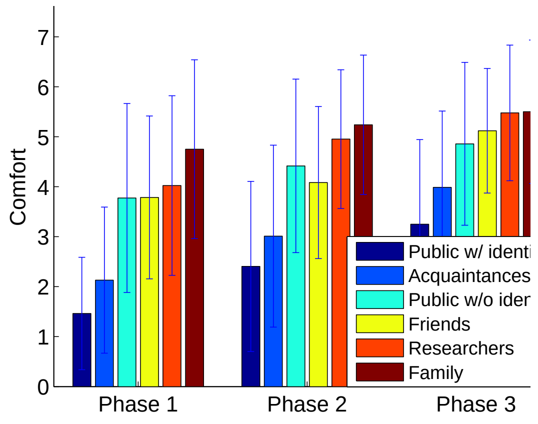
Figure 7.1: The data visualizations on a participant's daily report.

the participants to rate their comfort levels of sharing with six types of data recipients, i.e., researchers, family members, friends, acquaintances, the general public with a participant's identity, and the general public without a participant's identity. For data recipients except the general public, we instructed the participants to assume they were sharing with their identity because personal sensing applications typically require a user's identity. Within each section, there are seven questions for each sensor and context inference, i.e., GPS, accelerometer, activity status, ECG, respiration, stress, and conversation. Section 7 asked the participants to provide their general feedback in free-text form regarding their privacy concerns and our user study.

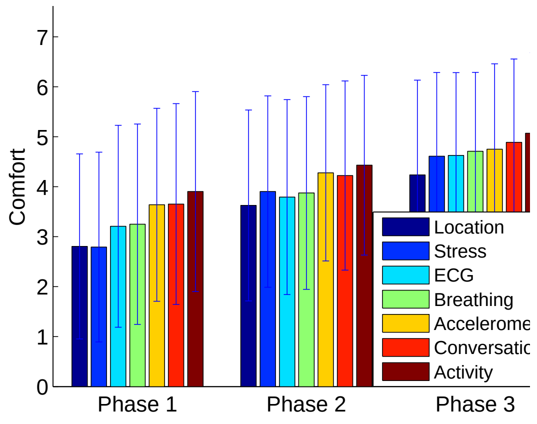
7.2 Results

In this section, we discuss our analysis of the survey questionnaires. We first discuss general findings from the analysis, and then verify if the hypotheses mentioned above hold true by discussing test results for statistically significant differences. Unless otherwise noted, we used paired one-tailed t-tests with a threshold of $p = 0.05$.

In Appendix B, we provide a figure showing a complete result of the survey, which include plots of mean comfort levels of the twelve participants, regarding all combinations of the distinct types of the data recipients and the sensory data. To facilitate interpretation of the plots, we summarize the results into a more concise form as shown in Figure 7.2. It



(a) Recipient Types



(b) Data Types

Figure 7.2: Summarized mean comfort levels of the twelve participants, with respect to the distinct types of (a) the data recipients and (b) the sensory data.

depicts the means and standard deviations of the comfort levels, and the error bars show ± one standard deviation from the mean.

7.2.1 Intuitive Findings

The participants were more concerned about with whom they shared than what they shared. The fact that the differences in the means among the recipient types in Figure 7.2(a) were greater than those among the data types in Figure 7.2(b) supports this. The order of comfortableness from the least to the most was: (1) the general public with identity; (2) acquaintances; (3) the general public without identity, and friends; (4) researchers; (5) family members. It was interesting that sharing with the general public is more comfortable than that with acquaintances, if one's identity was removed. In other words, participants were willing to make their private data publicly available after anonymization. On the other hand, the participants were not much concerned about the data types, as supported by the differences in the mean comfort levels that were not statistically significant (Figure 7.2(b))

7.2.2 T-Test Methodology

We performed three kinds of t-tests. The first t-test checks whether the mean comfort levels in Phase 2 (binary control) are greater than those in Phase 1 (no control), which is denoted as `2>1?` in the legend of Figure 7.3. The second one checks whether those in Phase 3 (expressive control) are greater than those in Phase 1 (no control), denoted as `3>1?`. The third one checks whether those in Phase 3 (expressive control) are greater than those in Phase 2 (binary control), denoted as `3>2?`. The three kinds of t-tests were performed on all tuples in a two-fold Cartesian product of the recipient types and the data types, hence there were 126 t-tests (3 kinds of t-tests $\times$ 6 recipient types $\times$ 7 data types).

Let's consider the results of t-tests as a three-dimensional (3D) array where each element contains true or false (i.e., a corresponding t-test was succeeded or failed). As mentioned before, within each phase, there were no significant differences in mean comfort levels across the data types (Figure 7.2(b)). Therefore, we collapsed the 3D-array along the data type dimension and counted the number of true elements. Each element of the resulting 2D-array contained the number of successful t-tests on each combination of the three t-test types and the six recipient types. Figure 7.3 shows the content of the 2D-array in a more intuitive form. On the Y-axis, we showed how many t-tests were successful for each combination of the recipient types and the three kinds of t-tests.

Intuitively speaking, each count on the Y-axis of Figure 7.3 can be interpreted as an indicator of how strongly a corresponding t-test holds true. For example, the left-most bar in Figure 7.3, means that the first t-test (checking whether the binary control gives more comfort than no control) is true at a degree of 5 out of 7 in case of researchers.

7.2.3 Hypothesis Verification

We describe the conclusions drawn from the t-test results shown in Figure 7.3 as follows.

- From the 1st t-test (`2>1?`), we found that the binary control gave weak comfort (0–3 out of 7) compared to the no control. However, it gave moderate comfort (5) for researchers.
- From the 2nd t-test (`3>1?`), we concluded that the expressive control gives much more comfort than the no control for all recipient types (7 out of 7) except family (3) and

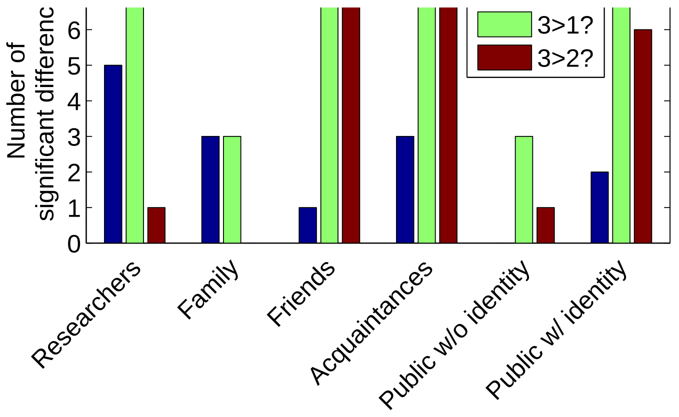


Figure 7.3: Statistically significant differences in mean comfort levels.

public without identity (3).

- From the 3rd t-test (`3>2?`), in comparison to the binary control, we found that the expressive control was effective for friends, acquaintances, and public with identity (6–7 out of 7) but ineffective for researchers, family, and public without identity (0–1).

We emphasize that the 1st and 2nd t-tests (`2>1?` and `3>1?`) support the first hypothesis (i.e., users feel more comfortable when they can control), and the 1st and 3rd t-tests (`2>1?` and `3>2?`) supports the second hypothesis (i.e., users gradually feel more comfortable as they can control more). However, how strongly the t-tests support the two hypotheses depends on recipient types. Specifically, let's categorize the recipients into two: (1) the sensitive group (friends, acquaintances, public with identity); (2) the safe group (family, researchers, public without identity). In general, the hypotheses hold true more strongly with the sensitive group than the safe group.

7.2.4 Feedbacks

In the end-of-study feedbacks, users generally recognized the power of the rule-based sharing mechanism, but they also complained about usability of the graphical user interface (GUI) of the rule-based control. As shown in Figure 4.2, the on/off control GUI consisted of buttons, while the expressive control GUI included tables, lists, check boxes, drop-down boxes, text-inputs, time-picker, etc. Therefore, the controllability came at a price of an effort to learn how to use such a control. Although one could come up with a more usable GUI for the expressive control, we claim that it is hard to become as convenient as the simple buttons, which involve almost no efforts to learn.

Regarding the tradeoff between usability and controllability, it is important to provide a privacy control with right usability based on an application's use cases. From our user study results, we conclude that if an application involves sharing with the safe group, privacy control does not need to be sophisticated because the binary control would suffice. If an application involves sharing with the sensitive group, it may have to provide a sophisticated privacy control even though it requires a user's effort to learn how to use it.

From other interesting feedbacks, we found that more considerate mechanisms for a user's subtle privacy needs are required. A user stated, "Sharing with researchers is a bit vague. My comfort level would depend on the purpose of the study, on the utility to me, and on whether my identity is known to the researchers." Another user mentioned, "I feel, for family, it's a more complex question whether to share data. I want to share all the data with them, but at the same time, I don't want to let them overly worry about how I'm doing. It is very complicated to describe such feelings in terms of the rules."

The former concern indicates that other factors such as purposes, utilities, etc. can affect data sharing decisions. In other words, user data can be subject to sophisticated privacy policies defined by users. The recent ideas on *policy-carrying data* [SWA15] would be useful for addressing such concerns. The latter concern shows that privacy is not simply a question of share or not. The data obfuscation techniques (mentioned in Section 8.2) that can hide privacy-sensitive information in sensor data could be used to remedy such concerns. However, it is worth to think about if deceiving his family is what he really wants.

CHAPTER 8

Related Work

In this chapter, we provide a summary of existing literature in pervasive sensing/actuation systems, mobile context inferences, and various works in data privacy. We cover related works in two broad dimensions, i.e., academic vs. practical, and personal vs. infrastructural sensing systems. We point out that the *kazaar* architecture stands as an academic but more towards a practical system because many realistic issues are considered in a design of the architecture. It also stands between personal and infrastructural sensing systems because the architecture itself is agnostic to types of sensing targets, so it is applicable to systems streaming personal sensory information flows.

8.1 Pervasive Systems

We introduce pervasive sensing/actuation systems in personal area (using mobile devices) and infrastructural area (using sensors/actuators embedded in buildings). In addition, commercial Home Automation systems are presented because they are infrastructural systems but deployed in personal buildings.

8.1.1 Personal Sensing

Early systems for sensor data collection include SenseWeb [SNL06], SensorWeb [CDB06], and GSN [AHS07]. The initial systems are focused on data collection and visualization to gain certain insights. Therefore, they have limited support for sharing data. More recently, several research systems [CYH06, MRS09, MLF08] and commercial cloud-based services[1]

[1]`https://xively.com`, `http://www.nimbits.com`, `https://www.thingspeak.com`, etc.

emerged. They facilitate sharing or distributing collected data through web APIs, but store users' data in a centralized server. As we discussed in Chapter 2, privacy of data owners are not well supported in a centralized architecture.

8.1.2 Building Management

Building Management Systems[2] (BMS) are designed for common building operations such as Heating Ventilation and Air Conditioning (HVAC) controls. The primary target of such systems is a centralized control, so their architecture inherently lacks support for data privacy. Several research projects for building management have been proposed in [DJT10, DMA12, RBB11]. They are primarily focused on designing usable and extensible frameworks, because the commercial building management systems are hard to program and limited in extending sensor/actuator interfaces. Sensor Andrew [RBB11] takes privacy into consideration, but it provides access controls in coarse-grained way.

8.1.3 Home Automation

In commercial industries, Home Automation systems[3] provide integration of sensors and actuators into a single system at individual home. They provide services for better user comfort, increased security, etc. They also support programming scripts for automated control of various actuators. However, their application target is personal buildings, so they lack support for sharing of sensor data with privacy in mind.

8.2 Data Privacy

An increased awareness of the privacy risks involved in sharing of personal information, coupled with a series of high-profile privacy breaches in the recent past has spurred interest in privacy research. To protect identity privacy, several metrics such as k-anonymity [Swe02],

[2]http://www.trane.com, http://www.johnsoncontrols.com, http://www.buildingtechnologies.siemens.com
[3]http://micasaverde.com,http://www.homeseer.com,http://www.control4.com/residential

l-diversity [MKG07], and t-closeness [LLV07] have been proposed. In addition, obfuscation techniques such as perturbation, suppression, and generalization [FWC10] are used for achieving these metrics. The above efforts are mainly directed towards protection against de-anonymization attacks on personal data sets. However, de-anonymization attacks exploiting the released quasi-identifiers (e.g., zip code, age, gender, etc.) have been performed, which indicates the apparent weakness of these schemes [NS08, Han06, NS09]. Moreover, while the above privacy metrics and algorithms are useful in context of relational data, they cannot be directly applied to sensory information because it is hard to perform anonymization without degrading much of its utility.

Several user studies have been conducted to investigate privacy concerns in sensory information. Researchers have found that privacy concerns increase as users better understand what kind of information can be inferred from sensor data [RGK11], but privacy profiles can help users to share more data without changing their comfort levels [WCS13]. Our user study complements the prior works and investigates how different sharing controls affect user's privacy concerns.

Moreover, the mere fact that a user does not share portions of their data could be also privacy intrusive because it means the user is trying to hide something by not sharing it. In this case, a data transformations or probabilistic techniques can be useful to mitigate such privacy concerns [EBF13, GNG12]. Automated rule-learning mechanisms proposed in [CMS11, BHA13] can be also adopted to help users manage privacy rules more effectively.

Ahmadi et al. [APG10] proposed a data transformation scheme that preserved a regression model of the original data. There are also several techniques for protecting identity that can be inferred from location information. Hoh et al. [HGH08] achieved k-anonymous location updates using a temporal cloaking scheme. Krumm et al. [Kru07] introduced techniques such as deleting, rounding, and addition of noise for obfuscating home location. Although the above schemes protect against identity privacy, they do not deal with the behavioral privacy.

To protect identity when sharing sensory information, a different set of techniques have been proposed. A popular strategy is to perturb the original sensor data while preserving its aggregate characteristics. AnonySense [CKK08] uses a mix-network in their architecture that anonymizes using sensor data from multiple users. PoolView [GPT08] is an architecture with a perturbation scheme that adds noise to original sensor data but maintains a community average and distribution.

8.3 Personal Data Stores

Researchers have widely adopted the concept of personal data stores to provide privacy when personal information is shared. Several system architectures in online social networks [CCL09, SSN10, TSG09, BBS09] provide a simple access control mechanism, which manages who has access to what data. Loccacino [TCH10] and PDV [MHM10] provide rule-based sharing control mechanisms. In comparison to SensorSafe, our rule-processing has practical performance even with a large amount of sensor data, and we further provides the differentially private data aggregates. Montjoye et al. proposed openPDS [MSW14] that executes third-party computations on personal data stores, which is complementary to our system. In addition, differential privacy has been adopted by several works [McS09, RSK10, CRF12, LWG13], which protects against revealing an individual in data sets. The main difference in our work is that we prevent a single user's data samples from being revealed by a series of aggregate queries.

In our earlier works [CCC11, CCS12], we extended the idea of personal data stores by having a broker coordinating many data stores, and proposed a rule-based sharing control mechanism with its initial implementation. In this paper, we improve the performance of rule processing by redesigning the underlying database layer, and further provide the differentially private aggregates. In addition, we conducted a user study to evaluate how rule-based sharing control affects users' privacy concerns.

8.4 Contexts on Mobile Devices

Several early systems [LYL10, RBA10] that provides context inferences from sensor data collected on mobile devices. The initial systems are based on stovepipe architectures and focused on optimizing individual data-processing pipeline. Such architectures suffer from energy inefficiencies due to the lack of global knowledge on context inference workloads. The energy efficiencies are important not only for longer battery life but also for data privacy because users can process more raw sensor data on their private devices.

According to a principle proposed by Chu et al. [CKL11], mobile operating systems should provide context data directly. Our system also follows the principle because it has several benefits over application-managed context. First, limited resources can be managed better by reducing redundant computations using global knowledge of context workloads. Second, sharing only intended inferences provides better data privacy. Third, operating systems themselves can benefit from the context information by adapting core OS services (e.g., scheduling, memory management, power management, etc.) to the current context.

The design principle of OS-provided context data has inspired several research systems. Funf [API11] is a framework that allows applications to simply subscribe to various sensors and context inferences on mobile devices. Kobe [CKL11] focuses on balancing energy, latency, and accuracy tradeoff of inference algorithms for better management of the limited resources. Orchestrator [KLM10] reduces resource usage by primarily conducting runtime off-loading of context computation. SymPhoney [JLY12] considers coordination of context inferences among multiple applications by scheduling. Our work on FlowEngine is especially focused on reducing redundant computations by providing a fine-grained dataflow programming framework.

The idea of using dataflow model for context inference algorithms has also been discussed in other works. Kobe [CLL11] provides a programming interface for designing context inferences in terms of feature and classifier modules. SymPhoney [JLY12] also includes a general dataflow framework based on DataBank [JML12]. However, Kobe is focused on balancing energy, latency, and accuracy of inference algorithms, and SymPhoney is focused on coordinating resource usage among concurrent sensing applications. Such techniques are complementary, and we elaborate the dataflow execution framework to enable more fine-grained sharing of computations through rich node connection and control mechanisms. Auditeur [NDA13] is a recent work that also employs dataflow programming, but it specifically focuses on processing acoustic data while we aim for a general-purpose framework.

CHAPTER 9

Conclusion

This paper presents *kazaar*, a software architecture that embraces pervasive sensing applications with tangible privacy. The key components of the kazaar architecture, i.e., SensorSafe and FlowEngine, improves data privacy by letting users actually own their data, retain better control, and process more within private devices. Besides, we free developers' mind from complex data processing flows and executing applications energy-efficiently. The evaluation based on the performance benchmarks and the user study demonstrates that our architecture proposal is feasible.

We also discuss contrasting styles of software architectures for pervasive sensing applications. The centralized architecture provides good manageability, while the distributed one respects individual's privacy. We propose *kazaar*, which architecturally unifies the both to achieve manageability and privacy at the same time, and provides users with tangible privacy experience. The key idea is to leverage virtualization techniques and operate personal data stores as physically close to users as possible. As a result, users get freedom to unplug when they want.

The reality of pervasive computing systems is still far from respecting individual's privacy. We envision pervasive computer systems should be designed with the following data privacy principles in mind: (1) *cognitive* end-to-end obfuscation (cf. *network terminal* end-to-end); (2) no traces on irrelevant devices; (3) self-destructible data; (4) traceable data inferences. Future research directions in *technically* implementing such principles will be meaningful, so people can use computer systems with better peace of mind.

We briefly describe a few ideas on how to implement the data privacy principles mentioned above. Note that many of following ideas can be only realized in the core of operating systems. The principle of *traceable data inferences* means that OS should provide a mechanism that can identify whether certain data originate from privacy-sensitive data. Once this tool is available, privacy-aware systems can determine whether to store certain data locally or back to a private storage. The taint-tracking technique [EGH14] is a promising candidate for such a mechanism. It can also help realize the *self-destructible data*. In addition, mobile application auditing and software-defined networking can be used to leave *no traces on irrelevant devices*. Finally, Lau et al. recently proposed to extend OSI 7 Layer for privacy [LCS14], which is a good example of the *cognitive end-to-end obfuscation*.

APPENDIX A

A Java Implementation of $\emptyset$-*multiset* Data Structure

```
import java.util.*;

public class HashMultiEmptySet<E>
  extends HashSet<E>
  implements MultiEmptySet<E>
{

  private int totalNumEmpty = 0;
  private int curNumEmpty = 0;

  private boolean isEmpty(E elem) {
    return ((Set)elem).size() == 0;
  }

  private Object emptySet() {
    return new HashSet<E>();
  }

  @Override
  public int size() {
    return super.size() + totalNumEmpty - 1;
  }

  @Override
  public boolean add(E elem) {
    if (isEmpty(elem)) {
      totalNumEmpty++;
      if (totalNumEmpty == 1)
        return super.add(elem);
      else
        return true;
    } else {
      return super.add(elem);
    }
  }

  @Override
  public Object clone() {
    throw new UnsupportedOperationException(
        "clone() not supported.");
  }

  @Override
  public boolean remove(Object o) {
    E elem = (E)o;

    if (!isEmpty(elem)) {
      return super.remove(elem);
    } else /* if (isEmpty(elem)) */ {
      if (totalNumEmpty <= 1) {
        return super.remove(elem);
      } else {
        totalNumEmpty--;
        return true;
      }
    }
  }

  @Override
  public Iterator<E> iterator() {
    return new Iterator<E>() {
      private Iterator<E> i = getSuperIterator();
    private boolean isEmptyHit = false;

    public boolean hasNext() {
      if (isEmptyHit) {
        if (curNumEmpty == totalNumEmpty)
          return i.hasNext();
        else if (curNumEmpty < totalNumEmpty)
          return true;
        else {
          assert false;
          return false;
        }
      } else {
        return i.hasNext();
      }
    }

    public E next() {
      E curElem;
      if (!isEmptyHit) {
        curElem = i.next();
        if (isEmpty(curElem)) {
         isEmptyHit = true;
         curNumEmpty++;
        }
      } else /*if (isEmptyHit)*/ {
        if (curNumEmpty++ < totalNumEmpty) {
          return (E)emptySet();
        } else {
          curNumEmpty = 0;
          isEmptyHit = false;
          curElem = i.next();
        }
      }
      return curElem;
      }
    };
  }

  private Iterator<E> getSuperIterator() {
    return super.iterator();
  }
}
```

APPENDIX B

Detailed Plots for the User Study Results

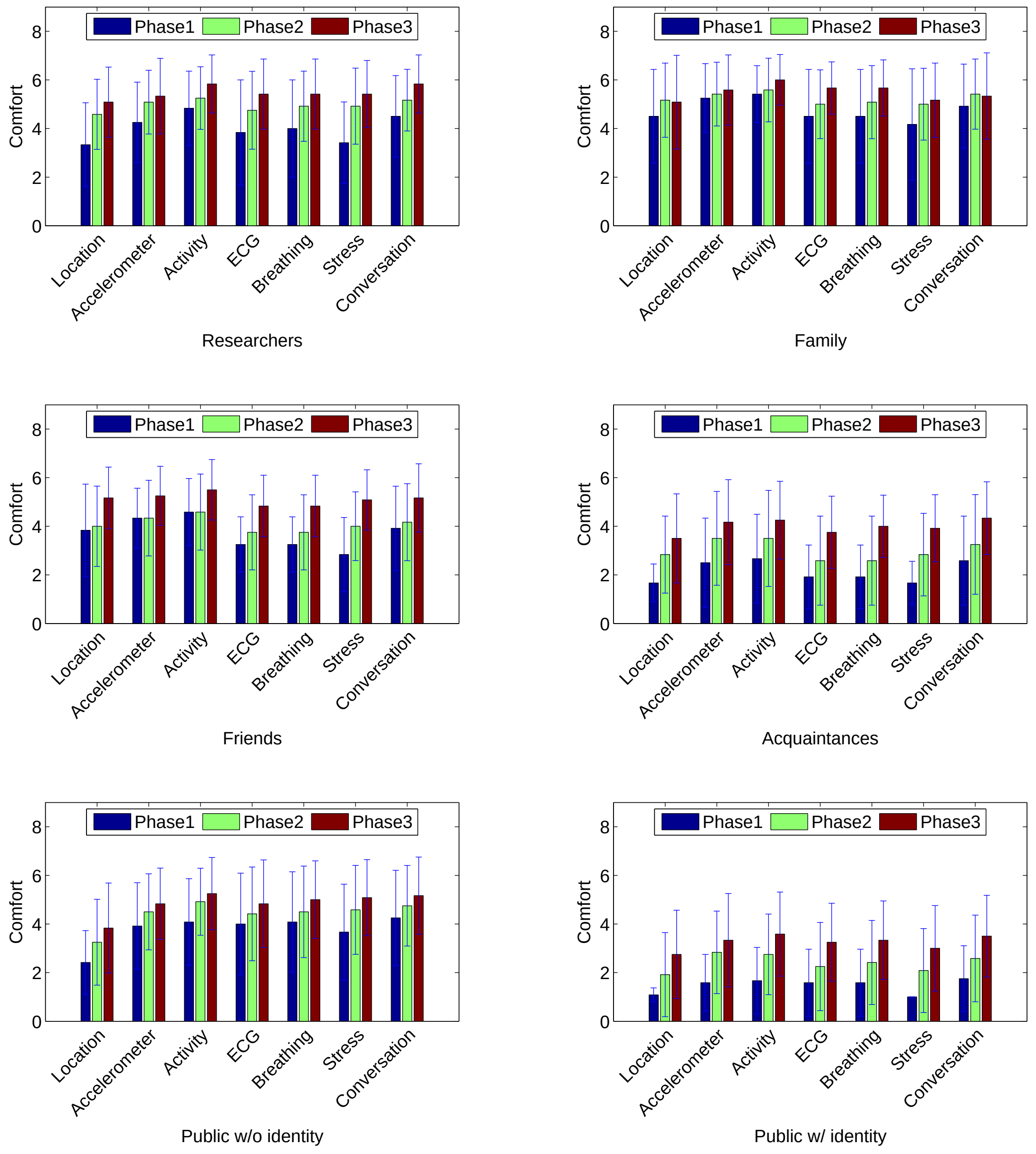


Figure B.1: Mean comfort levels of the twelve participants, with respect to all combinations of the distinct types of the data recipients and the sensory data.

References

[AHS07] Karl Aberer, Manfred Hauswirth, and Ali Salehi. "Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks." In *Proceedings of the International Conference on Mobile Data Management*. IEEE, 2007.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A Survey." *Computer Networks*, Elsevier, 2010.

[APG10] Hossein Ahmadi, Nam Pham, Ranti Ganti, Tarek Abdelzaher, Suman Nath, and Jiawei Han. "Privacy-Aware Regression Modeling of Participatory Sensing Data." In *Proceedings of the Conference on Embedded Networked Sensor Systems*. ACM, 2010.

[API11] Nadav Aharony, Wei Pan, Cory Ip, Inas Khayal, and Alex Pentland. "Social fMRI: Investigating and Shaping Social Mechanisms in the Real World." *Pervasive and Mobile Computing*, Elsevier, 2011.

[BBS09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. "Persona: An Online Social Network with User-Defined Privacy." *Computer Communication Review*, ACM SIGCOMM, 2009.

[BEH06] Jeffrey A. Burke, Deborah Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy, and Mani B. Srivastava. "Participatory Sensing." In *Proceedings of the Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications*. ACM, 2006.

[Ben99] Paola Benassi. "TRUSTe: An Online Privacy Seal Program." *Communications of the ACM*, ACM, 1999.

[BHA13] Igor Bilogrevic, Kévin Huguenin, Berker Agir, Murtuza Jadliwala, and Jean-Pierre Hubaux. "Adaptive Information-Sharing for Privacy-Aware Mobile Social Networks." In *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2013.

[Bra14] Tim Bray. "The JavaScript Object Notation Data Interchange Format." *RFC 7159*, Internet Engineering Task Force, 2014.

[CCC11] Haksoo Choi, Supriyo Chakraborty, Zainul M. Charbiwala, and Mani B. Srivastava. "Sensorsafe: A Framework for Privacy-Preserving Management of Personal Sensory Information." In *Secure Data Management*. Springer, 2011.

[CCC12] Supriyo Chakraborty, Zainul Charbiwala, Haksoo Choi, Kasturi R. Raghavan, and Mani B. Srivastava. "Balancing Behavioral Privacy and Information Utility in Sensory Data Flows." *Pervasive and Mobile Computing*, Elsevier, 2012.

[CCL09] Ramón Cáceres, Landon Cox, Harold Lim, Amre Shakimov, and Alexander Varshavsky. "Virtual Individual Servers as Privacy-Preserving Proxies for Mobile Devices." In *Proceedings of the Workshop on Networking, Systems, and Applications for Mobile Handhelds*. ACM, 2009.

[CCS12] Haksoo Choi, Supriyo Chakraborty, and Mani B. Srivastava. "Design and Evaluation of SensorSafe: A Framework for Achieving Behavioral Privacy in Sharing Personal Sensory Information." In *Proceedings of the International Conference on Trust, Security, and Privacy in Computing and Communications*. IEEE, 2012.

[CDB06] Xingchen Chu, B. Durnota, Rajkumar Buyya, et al. "Open Sensor Web Architecture: Core Services." In *Proceedings of the International Conference on Intelligent Sensing and Information Processing*. IEEE, 2006.

[CG05] Lorrie F. Cranor and Simson Garfinkel. *Security and Usability: Designing Secure Systems That People Can Use*. O'Reilly Media, 2005.

[Cho13] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly Media, 2013.

[CKK08] Cory Cornelius, Apu Kapadia, David Kotz, Dan Peebles, Minho Shin, and Nikos Triandopoulos. "AnonySense: Privacy-Aware People-Centric Sensing." In *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. ACM, 2008.

[CKL11] David Chu, Aman Kansal, Jie Liu, and Feng Zhao. "Mobile Apps: Its Time to Move Up to Condos." In *Proceedings of the Conference on Hot Topics in Operating Systems*. USENIX, 2011.

[CLL11] David Chu, Nicholas D. Lane, Ted T. Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. "Balancing Energy, Latency, and Accuracy for Mobile Sensor Data Classification." In *Proceedings of the Conference on Embedded Networked Sensor Systems*. ACM, 2011.

[CMS11] Justin Cranshaw, Jonathan Mugan, and Norman M. Sadeh. "User-Controllable Learning of Location Privacy Policies with Gaussian Mixture Models." In *Proceedings of the Conference on Artificial Intelligence*. AAAI, 2011.

[CRF12] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johannes Gehrke. "Towards Statistical Queries over Distributed Private User Data." In *Proceedings of the Symposium on Networked Systems Design and Implementation*. USENIX, 2012.

[CRT06] Emmanuel J. Candes, Justin K. Romberg, and Terence Tao. "Stable Signal Recovery from Incomplete and Inaccurate Measurements." *Communications on Pure and Applied Mathematics*, Wiley, 2006.

[Cum03] Peter Cumming. "The TI OMAP Platform Approach to SoC." In *Winning the SOC Revolution*. Springer, 2003.

[CYH06] Kevin Chang, Nathan Yau, Mark Hansen, and Deborah Estrin. “SensorBase.org-A Centralized Repository to Slog Sensor Network Data.” In *Proceedings of the International Conference on Distributed Computing in Sensor Systems*. IEEE, 2006.

[DJT10] Stephen Dawson-Haggerty, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler. “sMAP: A Simple Measurement and Actuation Profile for Physical Information.” In *Proceedings of the Conference on Embedded Networked Sensor Systems*. ACM, 2010.

[DMA12] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Bernheim Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. “An Operating System for the Home.” In *Proceedings of the Symposium on Networked Systems Design and Implementation*. USENIX, 2012.

[DMN06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. “Calibrating Noise to Sensitivity in Private Data Analysis.” In *Theory of Cryptography*. Springer, 2006.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. “Tor: The Second-Generation Onion Router.” In *Proceedings of the Symposium on Security*. USENIX, 2004.

[Dri08] Gwendolyn Driscoll. “New Study Finds California Neighborhoods “Designed for Disease”.” UCLA Center for Health Policy Research, `http://bit.ly/1AXtxQ9`, 2008.

[Dwo08] Cynthia Dwork. “Differential Privacy: A Survey of Results.” In *Theory and Applications of Models of Computation*. Springer, 2008.

[EBF13] Daniel A. Epstein, Alan Borning, and James Fogarty. “Fine-Grained Sharing of Sensed Physical Activity: A Value Sensitive Approach.” In *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2013.

[EGH14] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones.” *Transactions on Computer Systems*, ACM, 2014.

[EK96] Dawson R. Engler and M. Frans Kaashoek. “DPF: Fast, Flexible Message Demultiplexing Using Dynamic Code Generation.” *Computer Communication Review*, ACM SIGCOMM, 1996.

[ES10] Deborah Estrin and Ida Sim. “Open mHealth Architecture: An Engine for Health Care Innovation.” *Science*, AAAS, 2010.

[ESK11] Emre Ertin, Nathan Stohs, Santosh Kumar, Andrew Raij, Mustafa al'Absi, and Siddharth Shah. "AutoSense: Unobtrusively Wearable Sensor Suite for Inferring the Onset, Causality, and Consequences of Stress in the Field." In *Proceedings of the Conference on Embedded Networked Sensor Systems*. ACM, 2011.

[FR14] Roy Fielding and Julian Reschke. "Hypertext Transfer Protocol (HTTP/1.1): Authentication." *RFC 7235*, Internet Engineering Task Force, 2014.

[FWC10] Benjamin Fung, Ke Wang, Rui Chen, and Philip S. Yu. "Privacy-Preserving Data Publishing: A Survey of Recent Developments." *Computing Surveys*, ACM, 2010.

[GKO14] Eva Galpearuin, Nadia Kayyali, and Kurt Opsahl. "Dear NSA, Privacy is a Fundamental Right, Not Reasonable Suspicion." Electronic Frontier Foundation, `http://bit.ly/1pO8Nos`, 2014.

[GNG12] Michaela Götz, Suman Nath, and Johannes Gehrke. "Maskit: Privately Releasing User Context Streams for Personalized Mobile Applications." In *Proceedings of the International Conference on Management of Data*. ACM SIGMOD, 2012.

[Goe58] Gerald Goertzel. "An Algorithm for the Evaluation of Finite Trigonometric Series." *American Mathematical Monthly*, JSTOR, 1958.

[GPT08] Ranti Ganti, Nam Pham, Yu-En Tsai, and Tarek Abdelzaher. "PoolView: Stream Privacy for Grassroots Participatory Sensing." In *Proceedings of the Conference on Embedded Networked Sensor Systems*. ACM, 2008.

[Gre11] Peter Greenhalgh. "big.LITTLE Processing With ARM Cortex-A15 & Cortex-A7." *White Paper*, ARM Holdings, 2011.

[Hal08] Emily H. Halili. *Apache JMeter: A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites*. Packt Publishing, 2008.

[Ham10] Eran Hammer-Lahav. "The OAuth 1.0 Protocol." *RFC 5849*, Internet Engineering Task Force, 2010.

[Han06] Saul Hansell. "AOL Removes Search Data On Vast Group Of Web Users." The New York Times, `http://nyti.ms/1yMSeMp`, 2006.

[Har10] Amy Harmon. "Where'd You Go With My DNA?" The New York Times, `http://nyti.ms/1w7QLBs`, 2010.

[HGH08] Baik Hoh, Marco Gruteser, Ryan Herring, Jeff Ban, Daniel Work, Juan-Carlos Herrera, Alexandre M. Bayen, Murali Annavaram, and Quinn Jacobson. "Virtual Trip Lines for Distributed Privacy-Preserving Traffic Monitoring." In *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. ACM, 2008.

[HHL11] Jing Han, E. Haihong, Guan Le, and Jian Du. "Survey on NoSQL Database." In *Proceedings of the International Conference on Pervasive Computing and Applications*. IEEE, 2011.

[Jef13] Brian Jeff. "big.LITTLE Technology Moves Towards Fully Heterogeneous Global Task Scheduling." *White Paper*, ARM Holdings, 2013.

[JKM03] M. Jani, Juha Kela, Esko-Juhani Malm, et al. "Managing Context Information in Mobile Devices." *Pervasive Computing*, IEEE Computer Society, 2003.

[JLY12] Younghyun Ju, Youngki Lee, Jihyun Yu, Chulhong Min, Insik Shin, and Junehwa Song. "SymPhoney: A Coordinated Sensing Flow Execution Engine For Concurrent Mobile Sensing Applications." In *Proceedings of the Conference on Embedded Networked Sensor Systems*. ACM, 2012.

[JML12] Younghyun Ju, Chulhong Min, Youngki Lee, Jihyun Yu, and Junehwa Song. "An Efficient Dataflow Execution Method for Mobile Context Monitoring Applications." In *Proceedings of the International Conference on Pervasive Computing and Communications*. IEEE, 2012.

[KAB09] David Kotz, Sasikanth Avancha, and Amit Baxi. "A Privacy Framework for Mobile Health and Home-Care Systems." In *Proceedings of the Workshop on Security and Privacy in Medical and Home-Care Systems*. ACM, 2009.

[KLM10] Seungwoo Kang, Youngki Lee, Chulhong Min, Younghyun Ju, Taiwoo Park, Jinwon Lee, Yunseok Rhee, and Junehwa Song. "Orchestrator: An Active Resource Orchestration Framework for Mobile Context Monitoring in Sensor-Rich Mobile Environments." In *Proceedings of the International Conference on Pervasive Computing and Communications*. IEEE, 2010.

[KPC07] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon. "Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks." In *Proceedings of the International Symposium on Information Processing in Sensor Networks*. IEEE, 2007.

[Kri05] R. Krishnakumar. "Kernel Korner: Kprobes—a Kernel Debugger." *Linux Journal*, Belltown Media, 2005.

[Kru07] John Krumm. "Inference Attacks on Location Tracks." In *Pervasive Computing*. Springer, 2007.

[KWM11] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. "Activity Recognition Using Cell Phone Accelerometers." *Explorations Newsletter*, ACM SIGKDD, 2011.

[LCS14] Billy Lau, Simon Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. "Mimesis Aegis: A Mimicry Privacy Shield—A System's Approach to Data Privacy on Public Cloud." In *Proceedings of the Symposium on Security*. USENIX, 2014.

[Lee08] Edward A. Lee. “Cyber Physical Systems: Design Challenges.” In *Proceedings of the International Symposium on Object Oriented Real-Time Distributed Computing*. IEEE, 2008.

[Lee11] Min-Hyoung Lee. “An Android App-Developer Indicted Without Physical Detention Due to Violations of Location Information Protection Laws.” Digital Daily, `http://bit.ly/1vRpRI7`, 2011.

[Lew04] Franck L. Lewis et al. “Wireless Sensor Networks.” *Smart Environments: Technologies, Protocols, and Applications*, Wiley, 2004.

[Lim13] Charles Limonard. “The Future of Power Management in the Mobile Computing Market.” Embedded Computing Design, `http://bit.ly/1Lr7PUj`, 2013.

[LLV07] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. “t-Closeness: Privacy Beyond k-Anonymity and l-Diversity.” In *Proceedings of International Conference on Data Engineering*. IEEE, 2007.

[LPL09] Hong Lu, Wei Pan, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. “SoundSense: Scalable Sound Sensing for People-Centric Applications on Mobile Phones.” In *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. ACM, 2009.

[LWG13] Sangmin Lee, Edmund L. Wong, Deepak Goel, Mike Dahlin, and Vitaly Shmatikov. “πBox: A Platform for Privacy-Preserving Apps.” In *Proceedings of the Symposium on Networked Systems Design and Implementation*. USENIX, 2013.

[LWZ14] Felix X. Lin, Zhen Wang, and Lin Zhong. “K2: A Mobile Operating System for Heterogeneous Coherence Domains.” In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2014.

[LYL10] Hong Lu, Jun Yang, Zhigang Liu, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. “The Jigsaw Continuous Sensing Engine for Mobile Phone Applications.” In *Proceedings of the Conference on Embedded Networked Sensor Systems*. ACM, 2010.

[McS09] Frank D. McSherry. “Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis.” In *Proceedings of the International Conference on Management of data*. ACM SIGMOD, 2009.

[MHM10] Min Mun, Shuai Hao, Nilesh Mishra, Katie Shilton, Jeff Burke, Deborah Estrin, Mark Hansen, and Ramesh Govindan. “Personal Data Vaults: A Locus of Control for Personal Data Streams.” In *Proceedings of the International Conference on Emerging Networking Experiments and Technologies*. ACM, 2010.

[MKG07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. "l-Diversity: Privacy beyond k-Anonymity." *Transactions on Knowledge Discovery from Data*, ACM, 2007.

[MLF08] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. "Sensing Meets Mobile Social Networks: the Design, Implementation, and Evaluation of the CenceMe Application." In *Proceedings of the 6th Conference on Embedded Networked Sensor Systems*. ACM, 2008.

[Mor10] J. Paul Morrison. *Flow-Based Programming: A new Approach to Application Development*. CreateSpace, 2010.

[MRS09] Min Mun, Sasank Reddy, Katie Shilton, Nathan Yau, Jeff Burke, Deborah Estrin, Mark Hansen, Eric Howard, Ruth West, and Péter Boda. "PEIR, the Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research." In *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. ACM, 2009.

[MSW14] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S. Wang, and Alex S. Pentland. "openPDS: Protecting the Privacy of Metadata Through SafeAnswers." *PloS One*, 2014.

[NDA13] Shahriar Nirjon, Robert F. Dickerson, Philip Asare, Qiang Li, Dezhi Hong, John A. Stankovic, Pan Hu, Guobin Shen, and Xiaofan Jiang. "Auditeur: A Mobile-Cloud Service Platform for Acoustic Event Detection on Smartphones." In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2013.

[Nem10] Evi Nemeth. *UNIX and Linux System Administration Handbook*. Pearson Education, 2010.

[NS08] Arvind Narayanan and Vitaly Shmatikov. "Robust De-Anonymization of Large Sparse Datasets." In *Proceedings of the Symposium on Security and Privacy*. IEEE, 2008.

[NS09] Arvind Narayanan and Vitaly Shmatikov. "De-Anonymizing Social Networks." In *Proceedings of the Symposium on Security and Privacy*. IEEE, 2009.

[Par10] Tara Parker-Pope. "The Pedometer Test: Americans Take Fewer Steps." The New York Times, `http://nyti.ms/1LoNpwW`, 2010.

[PKG10] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. "Energy-Efficient Rate-Adaptive GPS-Based Positioning for Smartphones." In *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. ACM, 2010.

[PRH11] Kurt Plarre, Andrew Raij, Syed M. Hossain, Amin A. Ali, Motohiro Nakajima, Mustafa Al'absi, Emre Ertin, Thomas Kamarck, Santosh Kumar, Marcia Scott, Daniel Siewiorek, Asim Smailagic, and Lorentz E. Wittmers Jr. "Continuous Inference of Psychological Stress from Sensory Measurements Collected in the Natural Environment." In *Proceedings of the International Conference on Information Processing in Sensor Networks*. IEEE, 2011.

[RAP11] Md Mahbubur Rahman, Amin A. Ali, Kurt Plarre, Mustafa al'Absi, Emre Ertin, and Santosh Kumar. "mConverse: Inferring Conversation Episodes from Respiratory Measurements Collected in the Field." In *Proceedings of the Conference on Wireless Health*. ACM, 2011.

[RBA10] Andrew Raij, Patrick Blitz, Amin A. Ali, Scott Fisk, Brian French, Somnath Mitra, Motohiro Nakajima, M. Nuyen, Kurt Plarre, Mahbubur Rahman, et al. "mStress: Supporting Continuous Collection of Objective and Subjective Measures of Psychosocial Stress on Mobile Devices." In *Proceedings of the Conference on Wireless Health*. ACM, 2010.

[RBB11] Anthony Rowe, Mario E. Berges, Gaurav Bhatia, Ethan Goldman, Ragunathan Rajkumar, James H. Garrett, José MF Moura, and Lucio Soibelman. "Sensor Andrew: Large-Scale Campus-Wide Sensing and Actuation." *Journal of Research and Development*, IBM, 2011.

[RGK11] Andrew Raij, Animikh Ghosh, Santosh Kumar, and Mani B. Srivastava. "Privacy Risks Emerging from the Adoption of Innocuous Wearable Sensors in the Mobile Environment." In *Proceedings of the Conference on Human Factors in Computing Systems*. ACM SIGCHI, 2011.

[RMB10] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani B. Srivastava. "Using Mobile Phones to Determine Transportation Modes." *Transactions on Sensor Networks*, ACM, 2010.

[RPK12] Moo-Ryong Ra, Bodhi Priyantha, Aman Kansal, and Jie Liu. "Improving Energy Efficiency of Personal Sensing Applications with Heterogeneous Multi-Processors." In *Proceedings of the International Conference on Ubiquitous Computing*. ACM, 2012.

[RSK10] Indrajit Roy, Srinath TV Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. "Airavat: Security and Privacy for MapReduce." In *Proceedings of the Symposium on Networked Systems Design and Implementation*. USENIX, 2010.

[SBE09] Katie Shilton, Jeff Burke, Deborah Estrin, Ramesh Govindan, Mark Hansen, Jerry Kang, and Min Mun. "Designing the Personal Data Stream: Enabling Participatory Privacy in Mobile Personal Sensing." In *Proceedings of the Research Conference on Communications, Information, and Internet Policy*. TPRC, 2009.

[Sei07] David Seifman. "'Track' Man is Sacked." New York Post, `http://nypost.com/2007/08/31/track-man-is-sacked`, 2007.

[Sin15] Natasha Singer. "With a Few Bits of Data, Researchers Identify 'Anonymous' People." The New York Times, `http://nyti.ms/1Df6gFI`, 2015.

[SLS13] Youngmin Shin, Hoi-Jin Lee, Ken Shin, Prashant Kenkae, Rajesh Kashyap, DongJoo Seo, Brian Millar, Yohan Kwon, Ravi Iyengar, Min-su Kim, et al. "28nm High-K MetalGate Heterogeneous Quad-Core CPUs for High-Performance and Energy-Efficient Mobile Application Processor." In *Proceedings of the International SoC Design Conference*. IEEE, 2013.

[SNL06] Andre Santanche, Suman Nath, Jie Liu, Bodhi Priyantha, and Feng Zhao. "SenseWeb: Browsing the Physical World in Real Time." In *Proceedings of the International Conference on Information Processing in Sensor Networks*. ACM/IEEE, 2006.

[SSN10] Seok-Won Seong, Jiwon Seo, Matthew Nasielski, Debangsu Sengupta, Sudheendra Hangal, Seng K. Teh, Ruven Chu, Ben Dodson, and Monica S. Lam. "PrPl: A Decentralized Social Networking Infrastructure." In *Proceedings of the Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010.

[SWA15] Stefan Saroiu, Alec Wolman, and Sharad Agarwal. "Policy-Carrying Data: A Privacy Abstraction for Attaching Terms of Service to Mobile Data." In *Proceedings of the International Workshop on Mobile Computing Systems and Applications*. ACM, 2015.

[Swe02] Latanya Sweeney. "k-Anonymity: A Model for Protecting Privacy." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, World Scientific, 2002.

[TCH10] Eran Toch, Justin Cranshaw, Paul Hankes-Drielsma, Jay Springfield, Patrick G. Kelley, Lorrie Cranor, Jason Hong, and Norman Sadeh. "Locaccino: A Privacy-Centric Location Sharing Application." In *Proceedings of the International Conference Adjunct Papers on Ubiquitous Computing-Adjunct*. ACM, 2010.

[TSG09] Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. "Lockr: Better Privacy for Social Networks." In *Proceedings of the International Conference on Emerging Networking Experiments and Technologies*. ACM, 2009.

[WCS13] Shomir Wilson, Justin Cranshaw, Norman Sadeh, Alessandro Acquisti, Lorrie F. Cranor, Jay Springfield, Sae Y. Jeong, and Arun Balasubramanian. "Privacy Manipulation and Acclimation in a Location Sharing Application." In *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2013.

[WLL01] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer SJ Pister. "Smart Dust: Communicating with a Cubic-Millimeter Computer." *Computer*, IEEE, 2001.

[wor] "World-Tracker: UK Mobile Phone Location." `http://www.world-tracker.com`.