DroidSense: A Toolkit for Smartphone-Based Data-Intensive embedded Sensing Systems

Mohammad-Mahdi Moazzami
Michigan State University, USA
moazzami@cse.msu.edu

1. INTRODUCTION

Recent years have witnessed the emergence of a class of data-intensive embedded sensing applications such as structure health monitoring and volcano surveillance [6]. Sensors in these applications samples at high rates (e.g., 100 Hz) and hence generate a large amount of data. Due to the limited communication bandwidth, these applications typically process the raw data at sensors and only transmit summaries to a central station [6]. The MSP430/ATmega-based mote platforms have limited computation and storage resources, making them inappropriate for these applications. The high-end mote platforms such as Imote2 and SunSPOT, although equipped with sufficient processing capability, have significantly higher power consumption and are typically based on non-extensible designs, e.g., limited on-board flash memory that prohibits logging raw sensor data valuable for offline analysis.

We propose a smartphone-based versatile sensing platform to meet the needs of a diverse set of data-intensive sensing applications. Smartphones have several salient advantages such as multiple network interfaces (3G/4G, WiFi, Bluetooth), various integrated sensors, and user-friendly development tools. The price of smartphones has been decreasing drastically in the last decade. In particular, low-end Android phones like LG Optimus Net (800 MHz CPU and 2 GB memory) cost only about US$50 [2]. These features make smartphone a promising base platform for developing data-intensive embedded sensing applications.

In the VolcanoSRI project [3], we are currently building smartphone based systems for real-time volcano tomography. Each node samples seismic signal at 100 Hz, detects earthquakes, estimates hypocenters, and updates the tomographic model. Smartphones provide a low-cost platform that has sufficient processing capability and allows for rapid prototyping with a large number of nodes for embedded sensing applications like VolcanoSRI. However, several system challenges must be addressed. First, the sensitivity of on-phone sensors often does not meet the requirement of high-fidelity monitoring applications. For instance, the on-phone accelerometers cannot reliably detect the earthquakes lower than magnitude 4 [5]. Second, the smartphone operating systems do not provide real-time functionalities, such as constant sampling rate and accurate timestamping, which are crucial to many embedded sensing applications. Our measurements show that the detecting of a pulse signal through USB interface of Android phones suffers an unpredictable delay up to 5 ms, which makes it impossible to achieve constant high sampling rate. Third, the smartphone power management schemes, designed to adapt to user activities, are not well applicable to the embedded sensing applications. The continuous sensor sampling can prevent the smartphone from entering sleep state, which hence can quickly deplete the batteries.

This extended abstract presents the design of *DroidSense*, a toolkit for smartphone-based platform to build data-intensive sensing applications. To address the above challenges, DroidSense supplements the smartphone with an extension board (referred to as *extBoard*) that enables the integration of high-sensitivity external sensors, and is capable of real-time sensor sampling and lightweight signal processing. By offloading sensor sampling to extBoard, the smartphone can sleep for most of the time to save energy. DroidSense provides a rich and extensible library of signal processing algorithms, which can be easily hooked together to build sophisticated sensing applications. Owing to these features, DroidSense is a powerful toolkit for developers to build a wide spectrum of energy-efficient data-intensive sensing applications.

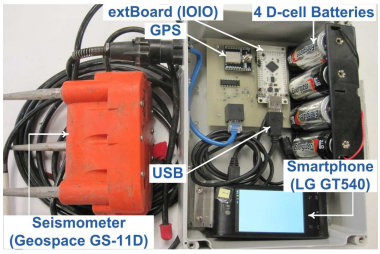

Figure 1: A DroidSense node for real-time volcano monitoring and tomography.

2. SYSTEM OVERVIEW

Hardware Composition: Each DroidSense node is composed of an Android smartphone and an extBoard (e.g., IOIO [1] and Arduino [4]) that is equipped with a low-power MCU and multiple A/D channels. The extBoard, powered by external batteries, is connected to the smartphone using a USB cable for both communication and charging the on-phone battery. Fig. 1 shows a DroidSense node built for the VolcanoSRI project [3]. Various sensors can be connected to the A/D pins of the extBoard.

System Architecture: The system architecture of DroidSense is illustrated in Fig. 2. DroidSense consists of: (1) an embedded program running on the extBoard, which samples the sensors and performs lightweight signal processing tasks, (2) an Android app running on the smartphone, which performs computation-intensive signal processing tasks and coordinates the operations of smartphone and extBoard by the Task Controller, and (3) a Task Partitioner that dynamically optimizes the dispatch of signal processing tasks to smartphone and extBoard to minimize energy consumption subject to processing delay.

Power Management: DroidSense disables many Android system services so that the smartphone can go to the sleep state sooner, then the extBoard wakes up the smartphone to run computation-intensive signal processing tasks in an on-demand manner.

3. SIGNAL PROCESSING LIBRARY

DroidSense provides a hierarchical and extensible library of signal processing algorithms to enable the developer to quickly build

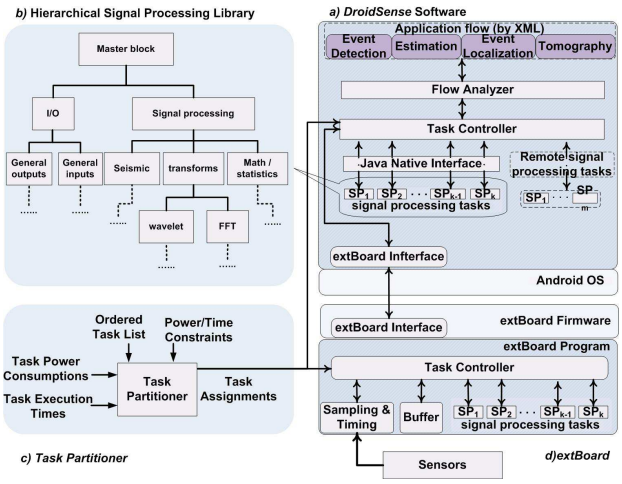

Figure 2: System architecture of DroidSense: a)DroidSense application on the smartphone. b)Hierarchical Signal Processing Library. c)Task Partitioner. d)extBoard program on the extension board.

reliable sensing applications. The signal processing algorithms are implemented in C++, such that they can be executed on both the smartphone and the extBoard. On smartphone, these C++ implementations are wrapped by Java Native Interface. By leveraging the object-oriented programming paradigm, DroidSense organizes the data structures and algorithms into a hierarchical library shown in Fig. 2(b). Current prototype version of the library includes 12 commonly used algorithms that have been fully tested on Arduino [4] and three different smartphones.

As the blocks implement unified I/O interfaces, they can be connected to build applications. An application is specified by an XML file. For instance, Fig. 3 shows an application of coarse P-phase picking, i.e., detecting the arrival time of wavefront in P-wave of earthquake, over 1600 data points of volcanic seismic signal [6]. This XML shows how seismic samples pass through a low pass filter and then are transformed to the wavelet domain. The transformed signal is then passed to the sparsity task to compute signal sparsity and to the picker task to find the p-phase. The Task Partitioner interprets this XML file and generates the task assignment (cf. Section 4). According to the assignment result, the Task Controller manipulates the execution of the tasks, the buffers of intermediate results, and the communication with the extBoard (including execution requests and receiving/sending the data).

4. TASK PARTITIONING

There exists a trade-off between energy consumption and processing delay. Intuitively, if all algorithms are executed on the smartphone, the shortest processing delay can be achieved, however, with the cost of high energy consumption by the smartphone. Hence, it is desirable to schedule the execution of each algorithm on either smartphone or extBoard, such that the energy consumption of the smartphone is minimized, subject to upper-bounded processing delay on extBoard that is responsible for real-time sensor sampling. Let s_i denote the i^{th} signal processing algorithm (referred to as *task*). Its execution times on the smartphone and extBoard are denoted by t_i^{p} and t_i^{b}, respectively. Let P denote the power consumption, where the scripts 'p' and 'b' represent smartphone and extBoard, and the scripts 'a' and 's' represent active power and sleep power. We assume that the power consumption and latency of downloading or uploading a data unit from/to the phone and to/from extBoard, is the same, which are denoted by P_{c} and

```
<application ID="1" name="P-phase picking" delay="1.0">
<!-- property delay specifies processing delay upper bound -->
  <task ID="0" name="low-pass" insize="1600" outsize="1600" />
  <task ID="1" name="wavelet4" insize="1600" outsize="100" />
  <task ID="2" name="sparsity" insize="100" />
  <task ID="3" name="picker" insize="100" />
  <!-- fromOut/toIn are index of output and input pins -->
  <connection from="0" to="1" fromOut="0" toIn="0" />
  <connection from="1" to="2" fromOut="0" toIn="0" />
  <connection from="1" to="3" fromOut="0" toIn="0" />
</application>
```

Figure 3: Specification of coarse P-phase picking [6].

t_{c}. Let l_i denote the size of the input data to s_i, and T denote the upper bound of processing delay. Let I_i denote whether s_i is executed on the phone ($I_i = 0$) or extBoard ($I_i = 1$). For a sequence of tasks $<s_i, s_2, \ldots, s_n>$, the Task Partitioner finds a solution $<I_1, I_2, \ldots, I_n>$ to minimize the energy consumption of smartphone, i.e., $\sum_{i=1}^{n} I_i P_{\text{s}}^{\text{p}} t_i^{\text{b}} + (1 - I_i) P_{\text{a}}^{\text{p}} t_i^{\text{p}} + |I_{i-1} - I_i| P_{\text{c}} t_{\text{c}} l_i$, subject to $\sum_{i=1}^{n} I_i t_i^{\text{b}} + |I_{i-1} - I_i| t_{\text{c}} l_i \leq T$. The parameters (i.e., P_{s}^{b}, P_{a}^{p}, P_{c} and t_{c}) can be measured in offline experiments. The profile of s_i (i.e., t_i^{p}, t_i^{b}, and l_i) is updated by Task Controller in run time. For the example shown in Fig. 3, the optimal solution is <1,1,0,0>, resulting power consumption of 305.1 mW for processing 1600 samples. Compared with the solutions <0,0,0,0> and <1,1,1,1>, it reduces energy consumption by 22% and 11.9%, respectively. These results are based on the power profile of the LG GT540 phone and Arduino [4] as the extBoard. Under the optimal solution, the projected lifetime over 4 D-cell batteries is 40 to 60 days if the duty cycle of the extBoard is 10% and the application shown in Fig. 3 is executed once every second.

5. CONCLUSION AND FUTURE WORK

This extended abstract presents *DroidSense*, a toolkit for smartphone based data-intensive sensing systems. DroidSense features a two-tier hardware architecture and provides a rich and extensible library of signal processing algorithms. It dispatches the execution of the signal processing algorithms either to the smartphone or to the extension board to minimize energy consumption subject to bounded processing delay. In the future, we will explore the online partitioning of the tasks. In addition, we will integrate the distributed signal processing and collaborative data analysis in DroidSense and account for power consumption of wireless interface in the design of task partitioner.

Biography

Mohammad-Mahdi Moazzami received the BS degree from Sharif University of Technology, Iran and the MS degree in computer science from Michigan State University, USA, in 2007 and 2012, respectively. He is currently a second year Ph.D candidate in the Department of Computer Science and Engineering at the Michigan State University under the supervision of Dr Guoliang Xing, and is expected to graduate in 2014. His research interests include collaborative sensing in sensor networks and Android-based embedded systems. In his dissertation research, he collaborates with Dr Rui Tan, and Dennis Phillips.

6. REFERENCES

[1] IOIO. `https://www.sparkfun.com/products/10748`.
[2] LG Optimus Net. `http://amzn.to/Y4BvO9`.
[3] VolcanoSRI project. `http://sensornet.cse.msu.edu`.
[4] Arduino. `http://www.arduino.cc`.
[5] M. Faulkner, M. Olson, R. Chandy, J. Krause, K. Chandy, and A. Krause. The next big one: Detecting earthquakes and other rare events from community-based sensors. In *IPSN*, 2011.
[6] G. Liu, R. Tan, R. Zhou, G. Xing, W.-Z. Song, and J. M. Lees. Volcanic earthquake timing using wireless sensor networks. In *IPSN*, 2013.