Design Considerations for the WISDM Smart Phone-based Sensor Mining Architecture

Jeffrey W. Lockhart, Gary M. Weiss, Jack C. Xue,
Shaun T. Gallagher, Andrew B. Grosner, Tony T. Pulickal
Department of Computer and Information Science
Fordham University
441 East Fordham Road
Bronx NY 10458
{lockhart, gweiss, xue, sgallagher, grosner, pulickal}@cis.fordham.edu

ABSTRACT
Smart phones comprise a large and rapidly growing market. These devices provide unprecedented opportunities for sensor mining since they include a large variety of sensors, including an: acceleration sensor (accelerometer), location sensor (GPS), direction sensor (compass), audio sensor (microphone), image sensor (camera), proximity sensor, light sensor, and temperature sensor. Combined with the ubiquity and portability of these devices, these sensors provide us with an unprecedented view into people's lives—and an excellent opportunity for data mining. But there are obstacles to sensor mining applications, due to the severe resource limitations (e.g., power, memory, bandwidth) faced by mobile devices. In this paper we discuss these limitations, their impact, and propose a solution based on our WISDM (WIireless Sensor Data Mining) smart phone-based sensor mining architecture.

Categories and Subject Descriptors
H.2.8 [**Database Applications**]: Data Mining

General Terms
Performance, Design, Security, Experimentation, Human Factors

Keywords
Sensor mining, sensors, smart phone, cell phone, data mining, accelerometer, mobile applications.

1. INTRODUCTION
Smart phones and other wireless mobile devices, including tablet computers, music players, and portable gaming systems, comprise a large and rapidly growing market. In fact, since the fourth quarter of 2010 smart phone sales have exceeded those of PCs [11]. Smart phones provide unprecedented opportunities for sensor mining since they include a variety of sensors, including an: acceleration sensor (accelerometer), location sensor (GPS), direction sensor (compass), audio sensor (microphone), image sensor (camera), proximity sensor, light sensor, and temperature sensor. The ubiquity and portability of these devices, combined with the many sensors that they host, provides us with an unprecedented view into people's lives—and great opportunities for data mining.

Sensor mining applications, however, face many challenges due to the severe resource limitations imposed on mobile devices. In this paper we discuss these limitations, their design impact, and propose solutions based on our WISDM (Wireless Sensor Data Mining) smart phone-based sensor mining architecture. The design considerations we identify will be useful to others who build smart phone-based data mining applications and can also be used to evaluate future smart phone-based sensor mining platforms.

We briefly introduce a few representative smart phone-based sensor mining applications, since it is difficult to evaluate design considerations independent of any applications. One class of sensor mining applications relies mainly on the smart phone's accelerometer to learn what activities (walking, jogging, etc.) a user is performing [4, 9, 12, 19]. Such activity recognition applications can be used to determine if a user is getting enough physical activity or to customize smart phone behavior based on context. Since a user's movements form a distinctive signature, a smart phone's accelerometer can also be used for biometric identification and authentication [6, 10]. Location-based sensor data mining is a particularly popular and expanding application area, which has matured sufficiently to spawn commercial applications. For example, Sense Networks [14] provides several location based data mining applications, including Citysense™, which identifies hot spots and also learns where each user likes to spend time. Other applications include Google Maps and Navigator, which identify traffic based on real-time GPS location data from large numbers of smart-phone users, which is then combined with knowledge about historical traffic patterns. Our WISDM research group [17] has developed smart phone-based activity recognition [9] and biometric [10] applications and is in the process of implementing them within our WISDM architecture so that they operate in real-time. We have chosen to implement our WISDM sensor mining architecture using the Android mobile operating system and Android phones for reasons described in Section 3.1, but this architecture can be adapted to use other mobile operating systems.

There are many high-level design constraints that a mobile sensor mining architecture must satisfy. First, it must be scalable to thousands, if not millions (e.g., Google Navigator), of users. It must also generate results in real-time, which means applying any learned models to the sensor data in real-time (most activity recognition systems require this). The architecture must also be able to *generate* predictive models in close to real-time and automatically deploy them, since some sensor mining applications will require this. The architecture should also support varying levels of distributed processing, where at one extreme the phone acts as a "dumb" client and the server is responsible for all data processing and mining, while at the other extreme everything is done locally on the "smart" client. For a variety of reasons including scalability, application independence, and user privacy, it makes sense to push processing tasks to the mobile devices, when feasible. But

SensorKDD '11, August 21, 2011, San Diego, CA, USA.

some tasks, by their very nature, may require centralized processing of data (e.g., road traffic analysis). Additionally, a server-based solution is needed if the application is too compute intensive for a smart phone or if the sensor data must be collected and saved (a key requirement for much academic research). Thus far, we have identified three high level design issues that impact the design of a phone-based sensor mining architecture:

Design Issue 1: Mobile devices have severe resource limitations with respect to battery power, computational speed (CPU), memory, and bandwidth which must be accommodated.

Design Issue 2: The sensor mining architecture must be scalable so that it can accommodate thousands or even millions of users.

Design Issue 3: Sensor mining applications must often generate results in real-time—and sometimes must also be able to learn (i.e., build predictive models) in real-time.

The remainder of this paper is organized as follows. In Section 2 we describe the sensors available on smart phones and some key issues that we have encountered using them. Our WISDM data mining architecture is then described in Section 3. Security and Privacy, two very important issues for smart phone sensor mining, are discussed in Section 4. Resource usage issues are described in Section 5 and then Section 6 summarizes our main contributions.

2. SENSORS

The sensors contained in smart phones are described in Section 2.1. Section 2.2 then describes the frequency with which sensor information is updated and Section 2.3 discusses how sensor usage can interact with normal phone functioning.

2.1 Description of Sensors

In this section we describe smart phone sensors, but provide more details for the most powerful and complex sensors. One key sensor is the accelerometer, which was initially provided on smart phones to handle screen rotation and to support advanced games. All Android phones and iPhones include a tri-axial accelerometer that measures acceleration in all three spatial dimensions. The accelerometer in Android phones varies by manufacturer, but as an example, the popular HTC Hero uses Bosch's BMA150 digital acceleration sensor [3] and Apple uses an LIS302DL accelerometer [15] in iPhones and recent iPods. The force of gravity is measured by most of these accelerometers and this information can be used to determine the direction downward (i.e., to Earth). These accelerometers tend to produce highly accurate measurements.

Smart phones also include sensors for determining location. The most accurate information is provided by the GPS receiver, which can provide location information to within a few meters. This functionality relies on triangulation with satellites in Earth orbit but typically does not function indoors. When the GPS signal is not available, most smart phones rely on other triangulation which is much less accurate, and uses the estimated distance to local cell phone towers and WiFi networks to calculate location. Such "artificial GPS" methods can still be quite accurate, however, in highly populated areas with many cell and WiFi signals. Smart phones are now routinely used by millions of users for mapping and route information via services like Google Navigator.

The audio sensor (i.e., microphone) can be used to monitor speech and background noise. Speech-to-text capabilities provide the opportunity for interesting (and perhaps troubling) sensor mining applications, but even gross measurements like sound level can provide clues about a user's environment or activity. For example, one application uses the microphone to help determine if a user is at a party [12]. The image sensor (i.e., camera) can also be useful, especially as the quality of the cameras improves. These sensors can be combined with image mining programs to perform face recognition and automatically identify objects and places. Unfortunately, the camera cannot be used for continuous monitoring since it is typically obstructed when the phone is not in active use.

Smart phones contain what we refer to as secondary sensors, which provide only limited capabilities. One such sensor is the light sensor, which measures the intensity of ambient light. The main purpose of this sensor is to determine appropriate screen brightness. It only operates when the smart phone is open, limiting its usage for other purposes. The proximity sensor determines the distance of an obstruction from the phone and the main intended purpose is to determine if the phone is pressed against one's ear, in which case the screen is turned off to save battery power. Some proximity sensors measure distance while others only support a binary measurement (near or far). Additionally, some proximity sensors use an infrared signal while others are only a "pseudo-sensor" implemented using the light sensor. Smart phones also often include a magnetic compass, which measures the Earth's magnetic field. While these secondary sensors may be limited in their function, they still could provide useful information to augment sensor mining applications. Additionally, smart phones can be connected to external sensors (e.g., we are experimenting with a Zephyr HxM Bluetooth heart rate monitor [20]).

2.2 Sensor Polling and Sensing Rates

A two-stage process is involved in providing sensor values to smart phone applications. In the first stage, the sensor hardware stores measurements from the sensors in shared memory, based on the sensing rate, S_r. In the second stage, the application requests the sensor measurement and the value is returned to the application. Assuming that the application continuously monitors the sensor, the rate at which the value is requested is the polling rate, P_r. The two rates may differ for each sensor. Only the polling rate is under programmer control. It is important that P_r should not be more frequent than S_r, to preserve limited resources.

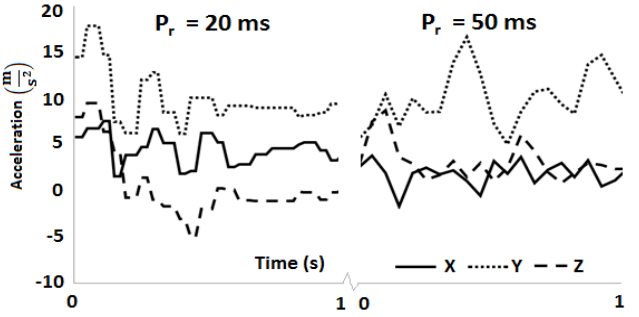

Figure 1. Accelerometer data using different polling rates

Figure 1 shows actual accelerometer data from an Android phone. When values are polled at 20 ms intervals, there is significant repetition—between two and six values will repeat, with the variance most likely due to system load. As we decrease P_r, the repetitions decrease until at 50 ms the application reliably returns unique values for each consecutive sample. This indicates that in practice, for the accelerometer on this phone S_r is 50ms and thus P_r should never be set more frequent that this.

Design Issue 4: The sensing rate S_r may sometimes not provide sufficient granularity for some sensor mining applications, and the application will have to take this into account.

For our activity recognition work, S_r is not sufficient to provide smooth curves for repetitive activities such as walking, jogging, and climbing stairs. The repetitive patterns can be approximated, but due to the sparse data (there are only 3 to 5 data points within a cycle) the peak values cannot be determined precisely and thus neither can the frequency of the repetitions.

Design Issue 5: The polling rate P_r should be configurable by application developers and users to match needs of particular applications and conserve limited resources.

One should not poll sensors more frequently than they are needed, since this will waste CPU cycles, while not polling them frequently enough will degrade the performance of the application. Sensors such as GPS need a much lower P_r than the accelerometer, because their values change more slowly—people do not move far in 50ms but acceleration values do change a great deal in 50ms. We address this in our WISDM architecture by providing different default P_r values for each sensor and allowing each to be reconfigured via the client's user interface.

2.3 Interactions with Normal Phone Function

Sensor mining is not the primary function of smart phones. Other functions, namely making and receiving phone calls, have much higher priority and thus sensor mining applications cannot unduly impact these other functions. A good sensor mining architecture must use shared resources very carefully. This topic is addressed in Section 4 and in this section we focus on other interactions.

While developing our client Data Collector application we encountered several software issues. These occurred because sensor applications were not a key issue in the design and testing of the phone and the Android operating system. Some of the problems that we found, which were also encountered by other sensor developers, only occurred for specific models of the phone and Android version. Specifically, when the phone's screen rotated while our Data Collector application was running, our application crashed. After we implemented a temporary workaround to automatically restart the application, we found that rotating the screen swapped the spatial axes of the accelerometer data—something which we had not anticipated. We then disabled the automatic screen rotation function while our application is running, as a workaround. Ultimately, however, our application must be coded to handle the screen rotation changes properly. This demonstrates the need for understanding and testing phone interactions.

Design Issue 6: Sensor mining development must carefully consider interactions with normal phone functioning and incorporate this into the testing process.

A second issue involved the device's hibernation mode, which is entered when the phone is inactive, in order to extend battery life. This mode normally shuts off the screen and changes the operating mode of the CPU. This causes a "sensor freeze" so that the sensor data that we collect contains repeated values. Clearly this is a critical, and fundamental, issue. We found that this problem occurred with some Android models but not others and some Android OS versions but not others. An Internet search revealed that many sensor developers were struggling with this problem. But there was not even universal agreement that this was a bug, since the functionality protected battery life. Ultimately, we addressed this problem by inserting a "Wake Lock" that keeps the CPU active when the phone is in hibernation mode. This demonstrates the issue of using a resource-constrained device for purposes that were not originally deemed important.

Design Issue 7: Smart phones are designed to conserve resources and this sometimes conflicts with the needs of sensor mining.

3. THE WISDM ARCHITECTURE

Data mining is generally done offline, but most sensor mining applications require results to be generated in real-time and potentially for a large number of users. For example, our ActiTracker activity recognition system [1] is being designed to support thousands of users and provide real-time results via a web interface. Similar real time and scalability constraints exist for many other sensor mining applications, including map navigation, traffic analysis, and biometric authentication. Some applications, such as navigation, even require large numbers of users to generate useful results, since traffic information is inferred directly from the users. Finally, applications must also run on mobile devices with inherently limited resources. These were our primary architectural concerns when designing the WISDM sensor mining architecture.

The sensor mining process involves several steps. First the raw time-series data from the phone's sensors must be collected and stored locally. Since traditional predictive data mining algorithms (e.g., decision trees) do not operate directly on time-series data, the next step for traditional methods involves *transforming* the time-series data into examples that summarize the data over a fixed time period. For our current activity recognition and biometric applications [9, 10] we generate one example, with 43 features, from each 10 seconds of accelerometer data. Next, pre-built classifiers are used to generate predictions (our architecture also supports the dynamic creation of classifiers). The final step involves reporting the results back to the user, by sending them to the phone and/or making them available via the Web.

3.1 Mobile Operating System and Platform

The WISDM sensor mining architecture is designed for smart phones and mobile devices running the Android operating system. Our reasons for choosing Android [2] over Apple iOS [8] are summarized in Table 1 and include the fact that Android uses a more popular programming language, provides multiprocessing, has free and well documented developer tools, is open source, has competitive market share, makes it easy to publish applications on the marketplace, and is supported by many hardware vendors—which should encourage innovation and result in lower costs.

Table 1. Mobile Operating System Comparison

Criterion	Apple iOS	Android
Language	Objective C	Java
Language Popularity	Low (Difficult)	High
Multiprocessing	No	Yes
Developer Tools		
Free	No	Yes
Documentation	Limited	Extensive
Open Source	No	Yes
App Approval	Strict Oversight	None
Market Share [8]	13.80%	14.50%
Hardware Venders	Apple	Many

Design Issue 8: The platform's mobile operating system should be easy to develop for: it should use a common programming language, have a well documented and inexpensive software development kit (SDK), and ideally it should be open source.

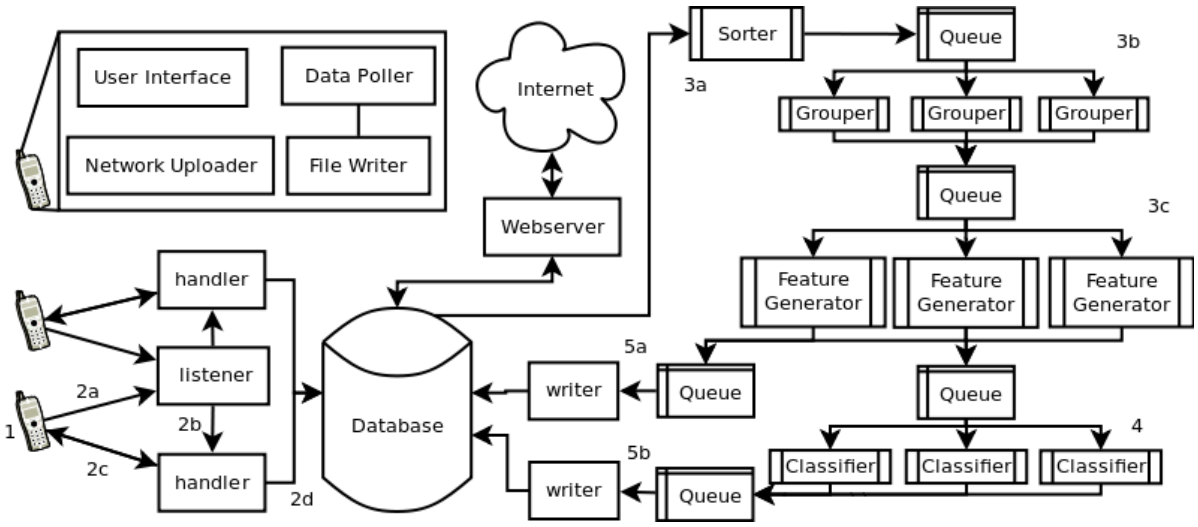

Figure 2. The WISDM Sensor Mining Architecture

There were some problems, however, with using Android. Namely, when we started the project, Android was not very polished and was not stable (in 3½ years the Android OS has been upgraded 8 times). This led to complications including which Android version to develop for, since each Android version provides different functionality. We decided to program the application for an older version (version 1.5) in order to ensure maximum compatibility with existing Android smart phones. Also, unlike the Apple iPhone, there are dozens of Android models, which complicates the process of compatibility testing. In fact, in Section 2.3 we noted several bugs that only occurred for some versions of Android and some phone models.

3.2 Sensor Mining Process Overview

Our architecture uses a client-server model to perform the steps identified at the start of this section. The client application provides a graphical interface for collecting sensor data from user's phones and for reporting results back from the server. The server is built to receive raw sensor data, transform it, classify it, and store the results and also report the results back to the user, via the phone and/or a web interface. As we will discuss, the nature of an application determines how (and whether) these tasks should be distributed between the client and server. In our simplest "dumb client" architecture, the client has minimal responsibilities and most of the processing and mining occurs on the server.

Figure 2 describes the "dumb client" version of our WISDM system. The numbers in the diagram refer to the steps in the sensor mining process. Step 1 involves the recording of the sensor data on the client device. Once the client has collected data, it contacts our server (Step 2a) and then in Step 2b the listener on the server passes the connection off to a handler thread, which then communicates with the client (Step 2c) to authenticate a user and accept his or her data. This data is aggregated and submitted to the database in batches (Step 2d). As more functionality is pushed to the client, Step 2 will need to be performed at a later time—or possibly not at all (see Section 3.3).

Control then passes to the server. In Step 3 the data is transformed to create examples from the time series sensor data (this step may not always be required). In Step 3a newly arrived data is periodically retrieved from the database by a sorting thread, which organizes the data so that records (i.e. snap shots of the sensor values recorded at P_r) that are from the same sensor and cell phone are processed together. Related data are stored in a queue until Step 3b is triggered, which has a grouping thread take the sorted time series records and bind them into examples. In our case an example represents 10 seconds worth of data, recorded at P_r =50 ms, so each example contains 200 sensor readings. The accelerometer returns 3 values when it is polled (vis. x, y, and z acceleration) so these 200 readings contain 600 values. In Step 3c, a feature generator thread takes them and transforms the records into an example. These examples are saved to two separate queues so that they can be stored and processing can also proceed. In Step 5a, examples are taken from one of these queues, grouped into batches for efficiency, and saved to the database. This ensures that we have a record of the intermediate stages of our work. These transformed examples can also be formed into a data set and shared with other researchers. Step 4 has a thread take examples from the other queue and run them through a pre-built classifier. The results are saved to a final queue and then aggregated into batches and saved to the database in Step 5b. With the classification results stored in the database, they can later be queried for visualization and presentation to the user via our web interface.

The scenario that was just described did not cover the dynamic construction of classifiers. We believe that sensor mining applications will need to build personal models for some applications (like activity recognition) in order to obtain highly accurate results. Thus, this capability is necessary and is part of our WISDM architecture. But this will also require smart phone applications to facilitate self-training. Our WISDM platform will facilitate this self-training phase for activity recognition by walking users through the process of collecting labeled training activity data.

Design Issue 9: Some sensor mining applications may require self-training to generate personalized models and the sensor mining architecture should support this ability.

3.3 Client/Server Architectures

The client and server responsibilities can be apportioned in many ways. The two extremes are the "dumb client" architecture, where minimal work is done on the client, and a "smart client" architecture, where the client does all of the work and no server is required. Between these extremes there are several viable intermediary architectures. Table 2 summarizes reasonable alternatives.

Table 2. Possible Client Workloads

Client Type:	1/Dumb	2	3	4	5	6/Smart
Data Collection	✔	✔	✔	✔	✔	✔
Data Transformation		✔	✔	✔	✔	✔
Classification			✔	✔	✔	✔
Model Generation				✔		✔
Data Storage					✔	✔
Data Reporting					✔	✔

Currently the WISDM client application, called Data Collector, implements the responsibilities of the dumb client, although in the future it will be able to be configured to support any of the workloads shown in Table 2. The Data Collector has a very simple user interface for managing the data collection and data upload process. The home screen has username and password fields, and buttons to "start recording," "send data," and "exit" plus a drop-down data label chooser so that people can label the data they collect for supervised learning applications (e.g. for activity recognition, the physical activity they are performing). When one clicks "Start," they are brought to a new screen that displays the sensor polling rates and notifies the user that it is recording. This screen has a "stop collecting" button to stop collection and return a user to the home screen. Sensor polling rates and server connection settings can be modified from within the normal Android application settings screen. Very soon, our application will support the automatic (periodic) upload of data.

In the future we will migrate additional server functionality to the phone, to reduce the workload on the central server. We plan to do this for our production application, ActiTracker [2], in order to make it scalable. The first stage in moving functionality to the client application is to transform the time series sensor data into individual examples on the phone (Client Type 2). This process is straightforward to implement and only moderately compute intensive. It also means that less data will be transmitted to the server, thus saving communication bandwidth.

The next step is to have the classification process occur on the phone using pre-built models (Client Type 3). In most cases implementing this will be quite straightforward. Also, depending on the type of model, the amount of computation required may be low (e.g., decision tree) or high (nearest neighbor). This architecture is quite scalable, with the phone performing most of the routine processing, and the server responsible mainly for storing and reporting the results. In many ways this may be optimal since the client is still relatively simple, the server is offloaded from most work, and the client does not need to store large quantities of results. This is the target architecture for our ActiTracker application, since we want to allow users to view their history of activities over long periods of time via the Internet.

Model generation is perhaps the most complex processing step and for this reason it will often be performed on the server. Since model generation will occur relatively rarely, this may not overly impact the scalability of the architecture. But in some cases having model generation occur on the application will be desirable, and this leads to Client Type 4. It may be much easier to implement model generation on the phone than one might think, since our WISDM platform uses the WEKA data mining suite [18], which is written in Java, and Java code can run under Android. Also, today's smart phones do have the processing power to run such tools. Even computationally intensive tasks, such as discrete Fourier transforms of sound samples, have been shown to be possible on older smart phones [12]. But since model generation is still the most complex step, it still will often be kept on the server. In this case we can still move more functionality to the client by keeping all of the results and reporting facilities on the phone (Client Type 5). In this case, however, since we do not transmit the sensor data to the server, we can only use pre-built models, which means that personalized models cannot be used. But this is fine for many applications. Finally, in the "smart client" extreme (Client Type 6), all functionality is on the mobile device.

Design Issue 10: Smarter clients trade off application scalability with limited resources.

3.4 A Central Database

Our data is stored in a MySQL database which has tables for raw, intermediate, and final data. This allows us to go back and modify our methodology, add new tasks, or visualize data which would be lost if we kept only the resulting classifications. These are critical abilities for academic research, but are likely to be unimportant in many consumer applications. However, there are some practical problems with storing large quantities of records in a relational database.

Design Issue 11: Storing large numbers of records in a database can make finding specific or related records difficult.

For our purposes, we needed to process sequential data together in blocks in order to translate individual readings into 10 second examples of activity. This also means we needed to prevent data from multiple users from mixing. Because all records share the same form, it is tempting to store them together in a single table. If this is done, data from different users, times, and activities will be mixed, and the data will have to be sorted back into homogeneous groups before evaluation. Even with hash tables and indexing, these tables will be difficult to manage, and extracting desired and related records will be CPU intensive. This mixing can be eliminated and table size reduced by partitioning data in separate tables based upon relevant information. Increasing organization increases accessibility, but stricter organization also introduces application overhead to sort new records outside of the database.

Design Issue 12: Relational databases are I/O chokepoints which can impede real time functionality.

In the "dumb client" architecture, even a single user will generate millions of records per day with a 50ms polling rate, and this will cause most queries to take a long time to run. Even the insertion of our research dataset into the database took hours on a dedicated machine because of the overhead involved in relational database systems like MySQL. This overhead includes insertion times, index updates, permission checks, and locking [5]. If inserting new data and querying it back takes longer than processing it, then the application will waste time waiting for the database. This translates to less responsive "real time" applications and in cases where large amounts of data are accepted continuously, it may mean that the server becomes severely backlogged and can never catch up. It is important, therefore, to limit queries and application dependence on a database. Where possible, applications should store and process all sensor data in memory and use the database for long term storage only. If raw sensor readings must be saved, they should be saved independently of their processing.

3.5 Concurrency

The massive quantities of data produced by mining sensor data from mobile devices makes concurrent processing very attractive.

As mentioned before, data mining tasks are frequently done sequentially, which is inadequate for real time applications. Fortunately, sensor mining is comprised of independent steps and discrete records which lend themselves to concurrent processing. There are two important kinds of concurrency for our work: performing the same task on multiple data at once (i.e. parallelism) and performing multiple steps of a single procedure (on different data) at the same time (i.e. pipelining).

Design Issue 13: Parallelism requires careful separation of data so that no data's processing depends on the state or content of other data.

Many of the steps in Section 3.2 can be performed in parallel. In our server each step is comprised of persistent threads, to avoid thread creation overhead after startup. For all steps that can be performed on multiple data in parallel (viz. client connections, record grouping, feature generation, classification), we use thread pools to manage the number of threads and balance the workload.

Design Issue 14: Pipelining requires careful separation and sequencing of tasks to avoid dependencies that can cause unnecessary stalls.

If data must go back and forth between stages, or data from one stage is tied to data from another stage, then some stages may end up waiting for others. Each stage must be able to proceed regardless of the other stages, to ensure that the application does not stall. First, this requires separate threads for each task so that the steps can proceed independently. This also means using separate thread pools for each task so that threads for some steps are not queued behind threads of other steps doing unrelated work. Second, buffers are needed between the steps because they take different amounts of time to complete. We resolve this timing problem using queues between steps. When a step finishes, it puts its result into one queue and takes the next job from another. If the input queue is empty or in use, then the thread waits until there is work. Thus, resources can be kept busy.

To avoid waiting for I/O operations, data to be saved is passed to another queue so that processing threads can continue execution. Dedicated I/O threads take from each of these queues and combine the data into batch queries, then execute them in mass to reduce database overhead. Each I/O thread has its own persistent connection to the database, to reduce connection overhead.

3.6 Language and Operating System

Data mining applications need to interface with other systems and this should affect the choice of programming language for the application. Mobile devices have native languages for applications that must be used to interface with the operating system. There may or may not be support for other languages, but the quality of that support and the overhead necessary to implement non-native code should be evaluated prior to development. Our client and server architecture was coded in Java, the primary development language for Android. Writing our server program in the same language allows for maximum compatibility between different code used by the project and even allows us to share code between the client and server.

Design Issue 15: Choice of programming language can have significant impact on the performance and integration of sensor mining applications.

Third party data mining algorithms also interface best with applications written in the same or friendly languages. Our WISDM platform uses the WEKA data mining suite [18], which is written in Java and can be imported natively into Java applications. Databases interface differently with different languages and so the choice of language and database should be coordinated. In our case Java has strong support for our MySQL database using JDBC.

Some languages are more efficient than others and since limited computation power is an issue for mobile devices, the choice of language is important. Our early work relied on Perl scripts because they were quick to write, but when we began working on automation and integration of the many tasks associated with data mining, we realized that any part implemented in Perl would be inefficient due to the way Perl works. By implementing our preprocessing in Java, we were able to reduce the time required by several orders of magnitude and a language like C could further increase application efficiency. Aside from efficiency, different languages may also better support an application's functionality. Mobile sensor mining is inherently a multi-system environment and generally involves communication over networks. Choosing a language that has strong support for these functions is important, which caused us to choose Java. Because it executes within the standardized, cross platform Java Virtual Machine, our programs will be portable to other systems. Java also provides strong typing at compile time and automatic memory management, which make applications more robust, and considerable support for secure network communication, authentication and cryptography [13].

3.7 Web Interface and Data Representation

The sensor mining results should be available to the user via a web interface. In our design, two separate versions will be supported: one for visitors on a computer and another for mobile devices. Due to the possibility of smart phone users visiting the full version of the website, the full version will be designed as efficiently as possible. This includes taking into account the size of all the images and scripts as well as the language the site was written in. The images and scripts had to be optimized in order to load as quickly as possible and the majority of the site was coded in PHP to reduce the load time by caching the headers and footers of the pages so that they would only have to be loaded once. The data visualization scripts are written in Javascript in order to provide real-time updates, greater user-interaction and to alleviate the load on our server, but this will increase the load on the mobile device's data connection. A possible alternative that we will have to consider implementing as we build the mobile website, would be PHP-based graphical representation which would shift the data rendering to the server just for visitors on mobile devices.

3.8 Algorithm Efficiency

A key component in any sensor mining architecture is the data mining module. Key questions involve: what data mining algorithms should be used, how and when the predictive models are built, how long the training process takes, and how long it takes to classify new data using existing models. Efficiency considerations are critical when addressing these issues due to the resource limitations placed on mobile devices.

There is no best data mining algorithm to use, but for real-time applications some algorithms are more appropriate than others. In particular, instance-based (nearest-neighbor) learning algorithms are especially appropriate, since they do not require any training time at all. Thus, they are good for applications, like activity recognition, where personalized models perform best. It is also critical that new data can be applied to learned models and results can be generated quickly. But most algorithms generate models that can quickly generate results. While instance-based learning meth-

ods may actually take some time to generate results, these methods can be sped up by implementing them directly, independent of any learning program, and by exploiting the matrix operations supported by modern processor architectures [6].

Design Issue 16: No single data mining algorithm will always perform best for all sensor mining applications and thus multiple data mining algorithms should be available.

Design Issue 17: Data mining algorithms that require little or no training time will be ideal for building personalized models.

Design Issue 18: Due to the speed and memory limitations placed on smart phones, data mining algorithms that require low overhead are most appropriate to run on a smart phone.

4. SECURITY AND PRIVACY

Mining mobile sensors provides opportunities to learn a great deal about people's lives and thus user privacy is a significant concern. Secure applications will also attract a wider audience [16].

4.1 Communication

One of the biggest concerns for data security is transmission. Sending data and communicating provide points of attack on data including person-in-the-middle and spoofing as well as on resources through SQL injection and malicious code.

Design Issue 19: Sensor data must be communicated securely over public networks with techniques like public key encryption.

Mobile devices communicate over many unsecure networks. They may be connected to a public WiFi hotspot, a cellular carrier's broadband network, or even a physical tether to a desktop computer. Users move about and may connect to many different and untrusted networks, including public networks. Therefore, it is critical to keep personal data private while it is being transferred over any network. We use public key certificates to verify the interaction between a client smart phone and our server. Using these certificates, we are able to encrypt all of the communication between client and server as an SSL session. These measures prevent a person-in-the-middle attack and keep a user's data private while it is in transit.

Design Issue 20: Connections from public networks and applications cannot be trusted to provide safe and accurate data unless it is verified and sanitized.

Because our server must accept sensor data from clients over the Internet, it is important to verify our interactions. First, we require authentication before any data is accepted, so the first interaction after connecting is user authentication or new user creation. This ensures that no data is sent anonymously (the user's identity may be anonymous but the data must be bound to a user ID so that it can be retrieved later). New users submit a unique user name (an email address) and password. When the application is released to the public, the unique Android device ID will also be used to verify users, but this is not needed in a research setting where multiple test subjects may use the same set of devices.

Second, our server functions as a demilitarized zone (DMZ) between clients and the database. Specifically, our architecture uses predefined strings to signal and communicate with clients. This means that the client asks the server to make database requests on its behalf and the server sanitizes all database queries. The client must know and initiate requests in the proper order for successful interaction. Too many failed requests or strings containing dangerous SQL will cause the server to deny and/or disconnect from the client. In addition to data verification and database isolation, this structure has the added advantage of being able to aggregate queries into batches. By using batches of predefined statements to execute queries, we reduce database load.

4.2 Storage

Other security efforts are meaningless if data is not stored securely. If the application architecture requires storing data locally, potentially malicious applications should be prevented from accessing this information. In a centralized storage situation, a key aspect in keeping user's data private is storing it separately from their personal information. All data that we store on our central server is associated with a unique user ID, which is assigned to each user at his or her creation. Users do not know their ID numbers; rather, when they authenticate with their username and password, the system fetches their user number from the password file and uses it internally. Personal data about users is also stored separately. This ensures that if all the sensor data and classifications were released to the public, it would still be impossible to identify users or link them with their data (although GPS data makes it possible to identify a person based upon their location).

Design issue 21: Sensor and user data must be stored securely both on the mobile device and at any central server.

Some sensors, like the GPS, can give away significant information about a user, ranging from current location and home address to habits and routines. The ambient light and magnetic compass sensors are unlikely to pose privacy risks of the same magnitude. It should be noted, however, that the results of data mining can produce information which is a greater security threat than the raw data, in the way that ActiTracker [1] will produce detailed activity histories from otherwise meaningless acceleration values.

Design Issue 22: some sensor data, such as audio, image, and location data, can be very sensitive, and so extra privacy measures must be in place to secure it.

Generally, more information is better for data mining tasks. However, when designing systems for the general public it is important to keep in mind what information is actually needed. For example, for ActiTracker there is no need to save everything the phone's microphone records and doing so would needlessly put private information at risk. Thus our applications will only save the sensor data that they need. However, applications often involve user accounts which can have unnecessary personal information as well. Because of this, user passwords are never stored. Instead, passwords are repeatedly hashed using SHA-256, and the result is compared to the hash value saved in our password file. Even if an attacker gained access to the password file, no passwords would be exposed.

Design Issue 23: Storing unnecessary data compromises security.

4.3 Application Level User Control

A key component of security is allowing users to control their own data, so that they can make informed decisions about their privacy. Allowing users to select which sensors are recorded increases their privacy. In our design a user can disable the recording of any sensor. This means that users need to understand what is being recorded. When applications are installed on Android devices there is a warning showing what the application has access to. Unfortunately, these permissions are so broad that most applications need access to many of them, so users do not really know how an application is using its permissions. As a result, users quickly learn to click through installation warnings and

permissions screens without paying attention. Even if they do, it is not always clear from the Android interface how different permissions and access rights are used by an application. It is entirely possible, then, for applications to spy on people using their mobile device's sensors. If malicious applications exploit sensor data, people may opt not to use legitimate sensor mining applications. Giving users control over their privacy makes it more likely that they will use sensor mining applications.

Design Issue 24: user privacy means user control over data collection, including notification of which sensors are being collected and options to control them.

5. RESOURCE USAGE

It is important for mobile sensor mining applications to operate continuously without interfering with normal phone operation. To help ensure this, in this section we enumerate a few key design issues related to resource usage. In addition, we present actual performance results associated with our WISDM Data Collector application, which currently operates in the "dumb client" mode. While we have taken some steps to minimize resource usage, we expect that over the next few months we will need to further optimize our design in order to make better use of limited resources. We have evaluated our Data Collector application on a variety of Android devices, including the Google Nexus One, Motorola Droid Pro, HTC Hero and EVO, and the Samsung Epic, Transform, Intercept, and Captivate. The majority of measurements provided, however, are based on the popular HTC EVO.

5.1 Battery

Battery life is an incredibly important issue on smart phones and, in particular, it is critical that such devices operate for at least a full day without being recharged. If sensor mining applications draw too much power and degrade battery life too significantly, then users will be unwilling to run these applications.

Design issue 25: Sensor mining applications must not degrade battery performance too significantly or prevent the device from operating without recharge for a normal 16-hour "waking" day.

Our measurements indicate that on the HTC EVO our Data Collector application consumes 35-51 milliwatts per second, and on an idle device with the screen powered off this corresponds to 6.6% of the device's power consumption. For comparison, the screen takes about 525 milliwatts per second at standard brightness (but the screen is not normally lit continuously). Thus the sensor collection process does not unduly tax the phone's battery, although the Data Collector does consume 31.6 times more power than the Android operating system. When the application is collecting data and in the CPU foreground (visible on the screen) the battery life will be 7:25 hours; under the same circumstances, but without the application collecting sensor data, the expected battery life is 7:30 hours. The lack of a significant change in battery life indicates that it is feasible to have an application constantly polling and recording sensors. The ability of mobile phones to sustain this activity is encouraging for sensor mining applications.

Future optimizations may help reduce power consumption. For example, we can buffer recorded data into memory before it is written to flash storage rather than writing data to flash every 50ms. Note that all these measurements are based on the "dumb client" architecture and that if additional processing occurs on the phone, this will have an impact on battery life. But our belief is that even a "smart" client will not require much additional power, although this has yet to be confirmed.

5.2 CPU and RAM

Computing power and RAM are also limited on smart phones. Since sensor mining applications may run continuously, we need to be even more concerned about their cumulative CPU and RAM usage, so that normal device operation is not compromised.

Design issue 26: The operation of sensor mining applications should not impact normal functioning of smart phones; consumption of CPU and RAM should not exceed 20%, on average.

CPU usage tests were performed on the "dumb" Data Collector client application. The tests were conducted on an HTC EVO 4G, which features a SD8650 chipset and a 1GHz Snapdragon Scorpion processor. While recording, the WISDM Data Collector's service spends approximately 3.4% of its uptime as the active process, which translates to approximately 2% of the CPU's total potential. For comparison, the Android 2.2 kernel occupies 3.6% of the CPU's capacity.

On Android phones services do not occupy the CPU foreground unless the screen is active, resulting in decreased priority on the CPU. In practice, this translates to a slower P_r value because the application does not have the priority to execute commands as frequently. For our application, with a P_r value of 50 ms, we find that we are only able to successfully poll new values every 100 to 200 ms. This problem may not have a significant impact on sensor mining applications that do not require such a high frequency P_r. Future work on the Data Collector includes implementing remote services which run independently of the application. This should prevent the collection service's CPU priority from decreasing when the application leaves the foreground. If this is the case, then any application that continuously polls sensors would need to use this or a similar technique.

The WISDM Data Collector uses 18MB of RAM, of which 12MB is reserved for data and 6MB is shared. Full RAM usage is approximately 3.5% of the HTC EVO's 512 MB RAM total. The application's memory use is relatively small and unlikely to conflict with other applications or interfere with normal operation.

5.3 Data Storage

Sensor mining has the potential to generate massive datasets. Applications can have thousands or even millions of users. More importantly, mobile sensors may continuously report new data, which means that even per user the data will grow rapidly. Applications such as ActiTracker, which polls the accelerometer as frequently as possible, will produce 72,000 records per hour.

Design Issue 27: Compression and efficient encoding are critical to manage the storage needs of sensor mining applications.

It is easiest to demonstrate the data storage requirements with a case study. With ActiTracker, data storage requirements for the same raw accelerometer data vary significantly based upon the format of the data. Each record, for instance, has a user ID and activity label, which are both repeated for many other records. Also, the timestamp for each record is almost identical to the timestamp of the previous and next records. This yields potential for data compression to save space. Encoding can also make a significant difference. We use ASCII encoding to produce human readable files, which turns single precision float values (a single byte) into multiple separate characters (each represented by a byte). A purely binary flat file would be able to compress data down to just 15.7% of the size of our average ASCII text files.

Databases are also prone to inefficiency. Numeric values are stored in fixed length spaces, but this space is not the minimum

binary space (e.g. in our MySQL database, single precision floats require four bytes, rather than one). Timestamps and data labels that are added by an application also get stored in fixed size spaces. This standardization eliminates much of the possibility for compression and application-specific encoding. In the case of ActiTracker, each record in the database requires only slightly less space than it does in an ASCII text file. Additionally, due to the overhead required for indexing and other system functions, the actual space taken up in the database is greater [5].

Design Issue 28: Saving only required data is critical for scalability and reducing storage costs.

Sensor mining applications will have varying degrees of preprocessing involved, and in many cases this will reduce the quantity of data. For our work with ActiTracker the data transformation step allows us to dramatically reduce the number of records stored because we collapse each 200 consecutive time series records into a single, slightly larger, example. The total space required for one example is 2.9% the size of the raw data that it describes. As outlined in Section 3.3, production applications may store only the transformed data, or just the prediction outcomes, dramatically reducing their storage needs.

Data storage and transmission are closely related and for everything other than a "smart" client, data storage first requires data transmission. Therefore, when determining the quantity and size of data stored at a central server by an application, it is important to take network bandwidth into account. Data stored locally on the device also has limitations to consider. Many smart phones are shipping with 8 GB or more of onboard storage, with options to expand to more. However, this space is often taken up by user's pictures, music, and documents, and taking away significant amounts of space from other applications may not be viable.

6. CONCLUSIONS

Sensor mining applications using smart phones will become common over the next decade. In this paper we describe key issues that a sensor mining architecture, and sensor mining applications, will have to deal with in order to provide benefits to the user while running on a resource constrained device. These issues will be critical for developing and evaluating sensor mining architectures and platforms. In addition, we described our WISDM sensor mining architecture. By doing so we demonstrated in a concrete manner how these design issues can be addressed. We also showed how a flexible architecture can support varying levels of responsibility between the client and server. Issues concerning resource usage and security and privacy were also highlighted.

The WISDM platform is partially implemented, with most basic functionality working and tested. We expect that our first production application, ActiTracker [1], will be released around September 1, 2011. This will allow researchers and everyday users to track their activities over time to see if they are receiving sufficient physical activity. It will also provide concrete feedback about the efficacy of our WISDM sensor mining architecture.

7. ACKNOWLEDGMENTS

We would like to thank past and present WISDM members that contributed to this project. This work was financially supported by a Google faculty research award, a Fordham faculty research grant, and several Fordham summer science research internships.

8. REFERENCES

[1] ActiTracker. http://ActiTracker.com/

[2] Android, Google. http://www.android.com/

[3] Bosch. BMA 150 Digital , Triaxial Acceleration Sensor Data Sheet, http://www.bosch-sensortec.com/content/language4/downloads/BST-BMA150-DS000-06.pdf

[4] Brezmes, T., Gorricho, J.L., and Cotrina, J. 2009. Activity Recognition from accelerometer data on mobile phones. In *Proceedings of the 10th International Work-Conference on Artificial Neural Networks*, 796-799.

[5] Elmasri, R. and Navathe, S. 2007. *Database Systems* 5th ed. Pearson, Boston, MA.

[6] Frank, J., Mannor, S., and Precup, D. 2010. Activity and gait recognition with time-delay embeddings. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*.

[7] InMobi Mobile Insights. (2011, April 14). *A Global View Of Mobile Advertising: Global Summary March 2011*.

[8] iOS, Apple. http://www.apple.com/iphone/ios4/

[9] Kwapisz, J.R., Weiss, G. M., and Moore, S.A.2010. Activity recognition using cell phone accelerometers. In *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*, Washington DC, 10-18.

[10] Kwapisz, J.R., Weiss, G.M., and Moore, S.A. 2010. Cell phone-based biometric identification. In *Proceedings of the IEEE 4th International Conference on Biometrics: Theory, Applications and Systems (BTAS-10)*, Washington DC.

[11] Menn, J. February 8, 2011. Smartphone shipments surpass PCs. Retrieved from http://www.ft.com/cms/s/2/d96e3bd8-33ca-11e0-b1ed-00144feabdc0.html#axzz1L2wKclC7

[12] Miluzzo, E. et al. 2008. Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In *The 6th ACM Conference on Embedded Networked Sensor Systems*, 337-350.

[13] Oracle. Java SE Security, http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html

[14] Sense Networks. http://www.sensenetworks.com/

[15] STMicroelectronics, LIS302DL 3-axis Accelerometer, http://www.st.com/stonline/books/pdf/docs/12726.pdf

[16] Sutter, J. D. (2011, April 21). Report: iPhones secretly track their users' locations. *CNN Tech*, mobile. Retrieved from http://www.cnn.com/2011/TECH/mobile/04/20/iphone.tracking/index.html?hpt=T2

[17] Weiss, G. M. 2011. WISDM (Wireless Sensor Data Mining) Project. Fordham University, Department of Computer and Info. Science, http://www.cis.fordham.edu/wisdm/

[18] Witten, I. H. and Frank, E. 2005. Data Mining: Practical Machine Learning Tools and Techniques, 2nd edition.

[19] Yang, J. 2009. Toward physical activity diary: Motion recognition using simple acceleration features with mobile phones, In *First International Workshop on Interactive Multimedia for Consumer Electronics* at ACM Multimedia.

[20] Zephyr. Consumer Heart Rate Monitor. http://www.zephyr-technology.com/consumer-hxm.