A Comparison of Alternative Client/Server Architectures for Ubiquitous Mobile Sensor-Based Applications

Gary M. Weiss
Fordham University
gweiss@cis.fordham.edu

Jeffrey W. Lockhart
Fordham University
lockhart@cis.fordham.edu

ABSTRACT
Mobile devices such as smart phones, tablet computers, and music players are ubiquitous. These devices typically contain many sensors, such as vision sensors (cameras), audio sensors (microphones), acceleration sensors (accelerometers) and location sensors (e.g., GPS), and also have some capability to send and receive data wirelessly. Sensor arrays on these mobile devices make innovative applications possible, especially when data mining is applied to the sensor data. But a key design decision is how best to distribute the responsibilities between the client (e.g., smartphone) and any servers. In this paper we investigate alternative architectures, ranging from a "dumb" client, where virtually all processing takes place on the server, to a "smart" client, where no server is needed. We describe the advantages and disadvantages of these alternative architectures and describe under what circumstances each is most appropriate. We use our own WISDM (WIreless Sensor Data Mining) architecture to provide concrete examples of the various alternatives.

Author Keywords sensors, sensor mining, data mining, smartphone, ubiquitous computing, mobile computing.

ACM Classification Keywords C.2.4 [Computer Communication Networks]: Distributed Systems– Client/Server

General Terms Design, Measurement, Performance, Security.

INTRODUCTION
Mobile devices possess significant and rapidly developing computational, network, and sensing capabilities. These devices typically contain many sensors, such as vision sensors (cameras), audio sensors (microphones), acceleration sensors (accelerometers) and location sensors (e.g., GPS), and also are able to send and receive data wirelessly. These mobile devices, such as smartphones, smart music players, and tablets are now ubiquitous and smartphone sales alone now exceed PC sales [6]. However, the sensor and network capacity of these devices is underutilized. Most applications only support direct user interaction (e.g. reading email, making calls, playing games) and the potential for ubiquitous smart sensing applications which run continuously in the background is largely unexplored.

As the next wave of ubiquitous computing applications is developed to take advantage of and integrate the sensors on these devices into intelligent systems, questions about the architecture of these systems will be critical [4]. Designing ubiquitous systems for these platforms presents a number of unique challenges. Although smart devices have unprecedented computing, memory, and network capacities for mobile sensor platforms, they are still very limited in their resources. Mobile devices must have reasonable battery life and performance when running other applications: mobile sensing is still only a secondary concern to activities like making calls, reading emails, and browsing the web on these devices. Network resources are also critical to users, who may have expensive data plans or limited network access for their devices. In order to navigate these concerns, it is important to consider the advantages of different architectures for ubiquitous sensor-based applications. The "Code in the Air" mobile sensing application development platform makes the architecture decisions for developers, pushing all responsibilities to the phone except those which require data from multiple devices [7]. We believe that this is not always the most efficient or desirable distribution of responsibilities given limited device resources and researcher/developer interests.

Throughout this paper we will use our own Actitracker [1] application to illustrate alternative ways of partitioning responsibilities between the client and the server. This application utilizes our activity recognition research [3] to recognize a person's physical activity using the accelerometer in their Android smartphone, and then presents this activity information to the user via a graphical user interface. Actitracker is built on top of our WISDM (Wireless Sensor Data Mining) platform [5], which will ultimately allow the client and server responsibilities to be partitioned in a number of different ways.

CLIENT/SERVER RESPONSIBILITIES
In this section we describe how responsibilities can be divided between the mobile client and the server. We talk about "client/server responsibilities" rather than "client/server architectures" because we feel the former term is more accurate and focused, although the latter term certainly is appropriate.

Basic Responsibilities for Sensor-Based Applications
In order to describe alternative ways to partition responsibilities between the client and server, we need to first identify the component tasks of sensor-based applications. While these responsibilities could be broken up in many ways—based on the specific application and on the level of

Copyright is held by the author/owner(s).
This material is based upon work supported by the National Science Foundation under Grant No. 1116124.
UbiComp'12, September 5-8, 2012, Pittsburgh, USA.
ACM 978-1-4503-1224-0/12/09.

granularity at which these responsibilities are represented—we feel that the following list is reasonably general and applicable to a wide range of applications. Nonetheless, the list is undoubtedly influenced by our recent work [3, 5, 8] and also by the data mining/knowledge discovery paradigm. However, we have tried to generalize the terminology below so that it is not restricted to data mining applications. The first four steps are generally viewed as sequential, while the last step occurs outside of this sequence (it could occur before, during, or after the other steps). Some of the steps may not be present for all applications.

1. Sensor Collection: This includes the measurement and recording of the raw sensor data.
2. Data Processing and Transformation: This involves simple modifications to the raw data as well as data transformation that may summarize/aggregate the data to a different level.
3. Decision Analysis/Model Application: This involves the application of some decision process to the data, which for data mining applications usually involves applying a predictive model. This will generate results that can be viewed as knowledge.
4. Data and Knowledge Reporting: This involves storing the results and presenting them either periodically or on demand. The results may trigger responses or automated actions by the system.

Learning/Model Generation: This involves learning from the data to generate some form of decision strategy or predictive model.

Table 1. Basic Responsibilities for Sensor-Based Applications

We use our Actitracker application in "dumb client" mode to illustrate what the responsibilities in Table 1 entail and how they can be partitioned between a mobile client and a server. Because these responsibilities occur sequentially we refer to them as "steps." In step 1 Actitracker collects the raw accelerometer data from the smart phone 20 times per second and then sends it to the server. Then, in step 2, every 10 seconds the server aggregates the resulting 200 raw samples into a single example, which describes the data using several dozen features, such as average acceleration [3]. This step is omitted in the framework created by Lane et al. [4] based on a survey of existing systems, but we feel that it is an architecturally significant step. Next, in step 3, a classification model, which was previously built using labeled training data, is applied to the generated examples to identify the activity that the user is performing. In step 4 the identified activity is saved for future use, and at some point later this information will be viewed by the user, most likely via a web-based interface.

The learned model can be generated via classifier induction methods (e.g., decision trees or neural networks) at various points. Our impersonal models are generated using labeled data from other people and hence may have been built in the distant past, while personal and hybrid models use data from the current user, and thus would be built after the user entered a training phase [8].

Alternative Schemes for Partitioning Responsibilities

Because the basic responsibilities in Table 1 are usually applied sequentially, they can be partitioned in only a limited number of meaningful ways. Table 2 indicates the possible partitions by specifying the client responsibilities (all other responsibilities are assumed to be done by the server). The client configurations (CC) specify these responsibilities and vary from CC-1, a dumb client, to CC-4, a smart client that assumes all responsibilities and does not require a server. In the following, we defer the discussion of model generation until the end.

		Client Configuration			
Responsibility		CC-1 Dumb	CC-2	CC-3	CC-4 Smart
1	Sensor Collection	✓	✓	✓	✓
2	Data Transformation		✓	✓	✓
3	Model Application			✓	✓
4	Reporting				✓
	Model Generation			?	?

Table 2. Four Basic Client Configurations

Client configuration 1, CC-1, is a "dumb" client in that it assumes the minimum possible responsibility; it collects the sensor data and transmits it to the server, which then assumes the other responsibilities. CC-2 assumes a bit more responsibility and processes and/or transforms the raw data before sending it to the server. This will usually reduce the amount of data that needs to be transmitted, since the transformed data is usually much smaller than the raw data. CC-3 applies the decision process on the client and the results are then transmitted to and stored on the server so that they can be viewed via the web. Finally, CC-4 provides everything on the client, including the reporting function, thus avoiding the need for a server.

Thus far we have not covered the model creation step, where learning occurs. This step is needed for intelligent applications. Model creation may occur on the client or on the server, although it is only recently that mobile devices have become powerful enough to run sophisticated machine learning algorithms. It makes little sense to create a model on a client and then not apply that model to the data on the client, so CC-1 and CC-2 will not normally perform model generation. In CC-3 the model is executed on the client, which may or may not generate the model; we refer to the variant that performs model generation on the client as CC-3_{MG}. Because it is often very simple to apply a model, as is the case with decision trees or rules, but computationally intensive to construct one, CC-3, and not CC-3_{MG}, may be the best choice for many applications. In CC-3, the model can be either hard-coded in the client (based on learning that occurred before the software was coded) or generated by the server and downloaded to the client. Similarly for CC-4, the "smart" client, the model may be generated before the client is installed and packaged with the client or it can be generated on the client (CC-4_{MG}). Neither configuration requires a server.

In our WISDM platform, our initial work supported CC-1, which in many ways is the simplest to implement. We are currently making enhancements to the client to support CC-2 and CC-3. Implementing CC-3 to move the models to the smartphone will only require a modest amount of work, since we use the WEKA data mining suite [10] to build our models, and WEKA can export models as Java code, which Android smartphones can run natively. We may implement CC-3_{MG} as well, if model generation with our data set does not over-tax the device. We do not plan to implement CC-4, since we want to provide web-based reporting capabilities.

COMPARISON OF CLIENT/SERVER PARTITIONS

Each partition of client and server responsibilities has its own benefits and drawbacks. Before we can compare the various alternatives, we need to identify the factors that will form the basis of the comparison. These factors are described below:

- Resource Usage: Mobile devices contain limited battery power, processing, memory, and transmission bandwidth, all of which must be conserved.
- Scalability: The architecture should be scalable in terms of the number of mobile sensor devices.
- Access to Data: Developers and/or researchers will often want full access to the data, including the raw data, for future use (transformed data loses some information).
- Privacy/Security: Users will often want to maintain the privacy of their data and keep it secure.
- User Interface: Users will want the best interface possible, where "best" factors in aesthetics as well as ability to view the results on multiple platforms.
- Application Restrictions: Some applications will require a central server to aggregate data from multiple users or devices, or to make data available to external systems.

We now analyze each of these factors, one at a time, with respect to the various client/server responsibilities. The CC-1 dumb client configuration keeps the client simple, which reduces processing and memory requirements, but requires the raw data to be transmitted to the server. Since the transformed data will take up less space than the raw data, CC-2 will make less use of the wireless radio. When using our WISDM architecture to support the activity recognition application, ten seconds worth of raw data takes up 3.1kB, while the resulting transformed example takes up only 176 bytes, a reduction of nearly 95%. Cellular data network connections often charge based on the amount of data transmitted, so there is financial incentive to limit transmissions. Additionally, wireless network connections require large amounts of power to transmit data, so reducing network use saves power. But computationally intensive data processing and data transformations, such as Fourier Transforms, can rapidly negate the power savings. If the CPU is overused, device performance and other applications may suffer, and the processor's battery use will exceed the power required for Bluetooth and Wi-Fi transmissions (3G cellular radio requires even more power). In such cases it may be more efficient to transmit the raw data and have the computationally intensive processing occur on the server.

CC-3 requires some extra computing power to apply the model or decision logic. This power will be minimal for rule or tree based logic, and extensive for algorithms like nearest-neighbor, where most of the work is done at classification time. However, in this case only the results need to be transmitted over the network; for Actitracker that could be represented in 1 Byte, which would encode the user's activity. In this case it is not clear whether less power would be used, especially considering the overhead involved in establishing and maintaining wireless network connections. Finally, CC-4 would require no network access and use the same processing power as CC-3, except to the extent that viewing the results on the phone will consume some processing and display power. Overall, it is not clear which client configuration will use the least power, although we are fairly sure that CC-2 would be more power efficient than CC-1. At all stages, the hardware, protocols, and algorithms used will have significant impact on power consumption. However, within the next year we may have empirical results for CC-1, CC-2, and CC-3, for Actitracker.

With respect to memory and clock cycles, as the client takes on more work, the amount of memory and CPU time consumed by the application will grow. Given all of these factors, it is not clear which client configuration performs best with respect to overall resource usage. This question is best answered empirically, although there are many factors that may make it difficult to generalize from a single platform.

Scalability is a much easier factor to assess—the more functionality that is moved to the client, the more scalable the architecture. Ultimately the smart client is perfectly scalable, since no server is required and the client performs all tasks. In Actitracker's CC-1 dumb client configuration, the server requires 1 minute to process and store 1 minutes worth of data from 942 simultaneous users (on a server with 8 processing cores running at 2.26 GHz and 24 GB of RAM). We find that we do not exceed 25% total CPU use and 5GB RAM use, because the server is limited by the rate we can commit results to our MySQL database, which is hosted on the same machine. With an additional server to store the database we might be able to support three or four thousand users simultaneously, but clearly there are scalability issues if we wanted to support millions of users.

As researchers who would like to preserve every piece of data, CC-1 is the optimal client configuration. Our transformation process involves summarizing the accelerometer data with high level features such as average acceleration. Over time, we may certainly decide to alter our data transformation process and incorporate additional features. If we have the raw data, we can apply our new data transformation scripts and evaluate our results. However, once we move to CC-2 we lose this ability. Maintaining all of this raw data comes at a steep price, however, in the form of data transmission costs as well as the cost of preserving all of this data on the server. Given that we sample every 50ms, a single user will generate 791MB of raw data in a month. Once we move to CC-3, only the results are sent to the server. This drastically reduces the historical data we maintain on the server, while still preserving the results. For our Actitracker application that means that we know when and how much time each person spends walking, sitting, etc., which is mainly all we need to know from an application perspective. Finally, with the smart client configuration, there is no server to send data to and hence no information will be generally available (unless we are permitted to query the mobile device, but that only has limited storage space). It is for these reasons that our project will continue to use the CC-1 configuration for some clients even after we have implemented some of the other configurations.

On the other hand, privacy and security favor the smart client configuration, in which no data or knowledge leaves the mobile devices. But for many applications, there may not be much a difference between CC-1, CC-2, and CC-3 in terms of security and privacy because the aggregated information may be almost as sensitive as the raw data. For example, aggregated GPS information that describes the regions that someone occupied might be as sensitive as the raw GPS traces, even though the latter contains more detailed information. Overall, privacy is a key reason for preferring the smart client configuration, although good security measures (e.g., encrypting data during transmission) can help address the privacy concerns. But even when data is kept on one’s mobile device, it is still not completely private or secure, since the device can be stolen or compromised. This can partially be addressed by encrypting the data that is stored on the device.

A central server can enhance the user interface both by making data available to one's desktop or other large displays and by making data available from anywhere over the web rather than only on the device which generates it. Additionally, a central server can help the user share or compare her data with friends or colleagues and via social networks.

Finally, some applications will benefit from a central server performing analytics on group or aggregate data. For example, traffic applications need access to the locations and speeds of multiple users, while home automation may require accounting for the needs of multiple occupants or gathering data from multiple sensing devices. Such applications will require client configurations CC-1 or CC-2 since they require the server to have access to data from multiple devices.

CONCLUSION

In this paper we described four basic models for distributing client and server responsibilities for ubiquitous sensor applications. This framework is then used to evaluate the comparative advantages and disadvantages of the different client/server architectures. Applications which do more processing on the mobile device will inherently be more scalable and use less bandwidth, but they will encounter other resource barriers. Further, some applications will need central servers to report data in a user-friendly way, or to process data aggregated from several devices. Finally, researchers and developers have incentives to maintain as much data as possible on the server, while users who desire privacy will prefer to keep data and information on the mobile devices. Ultimately each system must balance these concerns based on the needs of the application and the users. In our Actitracker application, we aim to implement multiple schemes, so that a researcher or user could in theory choose the one they find most appropriate. Such a solution could also work for future commercial systems.

REFERENCES

1. Actitracker. http://actitracker.com
2. Android, Google. http://www.android.com
3. Kwapisz, J., Weiss, G.M., and Moore, S.A. Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations,* 12(2):74-82.
4. Lane, N.D., Miluzzo, E., Hong Lu, Peebles, D., Choudhury, T., and Campbell, A.T. A survey of mobile phone sensing. *Communications Magazine, IEEE* Vol.48, 9 (Sept. 2010): 140-150.
5. Lockhart, J.W., Weiss, G.M., Xue, J.C., Gallagher, S.T. Grosner, A.B., and Pulickal, T.T. Design considerations for the WISDM smart phone-based sensor mining architecture. *Proc. 5th International Workshop on Knowledge Discovery from Sensor Data*, San Diego, CA (at KDD-2011): 25-33.
6. Menn, J. Smartphone shipments surpass PCs. February 8, 2011. Retrieved from http://www.ft.com/cms/s/2/d96e3bd8-33ca-11e0-b1ed-00144feabdc0.html
7. Ravindranath, L., Thiagarajan, A., Balakrishnan, H., and Madden S. Code in the air: simplifying sensing and coordination tasks on smartphones. *Proc. HotMobile’12 San Diego, CA.*
8. Weiss, G.M., Lockhart, J.W. The impact of personalization on smartphone-based activity recognition. *Proc. of the Activity Context Representation Workshop*, Toronto, Canada (at AAAI-2012).
9. WIreless Sensor Data Mining (WISDM) Lab. http://www.cis.fordham.edu/wisdm
10. Witten, I.H. and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Ed. 2005.