Self-Adapting Acquisition of Sensor Data in Mobile Environment

Chao Wang, Chunhong Zhang, Yang Ji
Beijing University of Posts and Telecommunications, Beijing, China
Email: cclcwangchao@gmail.com

***Abstract*—A mobile application meets the highly real-time requirement for data in sensor networks. However mobile applications have to handle many difficulties that do not exist in non-mobile environment. For example, the wireless network provided for mobile is slow, expensive and bandwidth-limited. So mobile application cannot acquire sensor data like a common application does, which will consume a large share of bandwidth, CPU and memory. To address this issue, we propose our design that will limit the concurrently acquisition of the sensor data with a self-adapting mechanism. It can lower the overhead of the system significantly and what's more, it hardly reduces the user experience even in the situation that the total acquisition frequency is limited.**

Keywords-sensor network; mobile; self-adapting.

I. INTRODUCTION

With the development of sensor network, sensing devices become more and more ubiquitous. One can have easy access to the sensor data through the APIs provided by some platforms. Cosm [1] is a typical example, which allows people to connect sensors to their platform. For developers, an application can be easily built with the restful [2] APIs. Meanwhile, the powerfulness of a smart mobile with the advantages of mobility and portability makes it more attractive, so a big boom of mobile applications offering sensor services can be seen in the near future.

However, the development of sensor applications in mobile environment meets many challenges. Hardware is limited in mobile devices such as CPU, memory or wireless network, so Applications that cost too many system resources are not tolerable.

In an application monitoring large amounts of sensors, the system overhead can be enormous. Considering the real-time requirement of sensor data and improving the users' experiences, sensor data should be acquired frequently. Considering a building with hundreds of sensor deployed, acquiring those data at the same time with hundreds of connections will definitely use up the resources of a mobile system.

To address this issue we propose our design to lower the system overhead by reducing the total acquisition for sensor data. It is self-adaptable, which means it can intelligently adapt to the acquisition frequency without outside help. It can be integrated seamlessly to kinds of platforms without the modification of server side. Moreover, users can hardly feel the potential loss of user experience brought in..

The next chapter presents the related work of our design. Chapter III proposes techniques to reduce the overhead of the system facing enormous sensors. In chapter IV, we describe the system architecture of design and evaluate its performance. Chapter V concludes the paper.

II. RELATED WORKS

In the sensor network, there are two methods, pulling and pushing, that can be provided for sensor data acquisition. Pulling data from the platform means a client should request the sensor data proactively such as HTTP protocol. On the contrary, the pushing way means the client should establish a long connection that can usually be a TCP connection with the platform, and when the data updates, client will be notified by the platform.

In a push-based system, it is widely discussed how to ensure the consistency of cache. For example, research [3] presents a notification protocol that satisfies to various consistency requirements.

However, in a sensor network environment, the situation is more complex. Recently, the Web of Things [4] concept is becoming more and more popular, and most sensor services are web services (like Cosm) using HTTP [5] protocol. HTTP protocol is a pull-base protocol. It can't retain a long connection so a push-based notification mechanism cannot be used in those platforms. The pull-based data acquisitions are more common in sensor network but how to ensure the real-time of sensor data in this environment should be more adequately studied.

The key to address this issue is the different real-time requirements of different sensor data. In mobile applications, we can see that only a small part of sensors need a high real-time requirement. Properly dividing sensors into several parts and different updating time interval can significantly reduce the system overhead.

To avoid the modification in existing platforms providing sensor data services, 'self-adapting' is required for mobile terminals. So the mobile terminal must be able to analyze the sensor information and environment to classify real-time requirement of sensors into different levels.

Semantic analysis of sensor description [6] for mobile phone is fully discussed. Taking advantages of these researches, we can extract important information for sensor classifying. In the next chapter, we will present in detail how our self-adaptive acquisition design works using the above techniques.

III. DESIGN

The key to solve the enormous sensor data problem is to lower the concurrent acquisitions, and we propose the

Published by Atlantis Press, Paris, France.

MTUF(the minimum tolerance update frequency) concept of the sensor.

A. *Factors Affecting MTUF*

MTUF is the minimum update frequency of sensor data that can be tolerated in an application. The update frequency no less-than MTUF can ensure the availability and user experience of an application. Considering a sensor deployed in a certain environment, a change of sersor value will not draws the attention until it is significant enough, when a notification may be necessary. So we assume Vmin as the minimum value that can draw the user's attention and the value Tvmin is the time interval when the change of the value is Vmin. So we can make our update time less than Tvmin to ensure the user's experience and we can also assume MTUF = 1/Tvmin.

Not all sensors' MTUF are the same, and they can be affected by several factors.

Table I shows the example of factors that can affect MTUF.

TABLE I. MTUFS FOR SOME FACTORS

Sensor With Factor	MTUF(Hz)
Wind speed sensor in outdoor	mutf1
Temperature sensor in outdoor	mutf2
Temperature sensor in car	mutf3
Sensor on display	mutf5
Sensor to be on display	mutf6

Type and deployment environment are two main factors affecting MTUF. For example, in the outdoor environment, the Tvmin of a wind speed sensor is much smaller than that of temperature sensor because wind speed changes more quickly and it needs a greater MTUF value. Similarly, the Tvmins for the same temperature sensors in the outdoor or in-car environment are also different. Usually the temperature changes more quickly in a car because of the airtight space.

The display status also affects its MTUF. Sensors are acquired at a certain frequency when it is on display. However when it is not on display, there is no need to acquire the data at the same frequency because we don't need its data at that time. We can lower the MTUF of the sensor or even stop acquiring it.

B. *Obtain MTUF Factors*

Now that we have already known the factors affecting MTUF, another issue is how to obtain them from the sensor information or application context.

1) Exacting Factors From Sensor Description: For factors like type and location which can not change in a short time, they can mostly be collected from a sensor description. By analyzing the description of a sensor we can obtain most factors. Mostly, the description of sensor is in XML Format. By parsing the XML document we can easily get these parameters. For example, the Sensor Web Enablement (SWE) [7] framework uses a standard Sensor Model Language (SensorML) providing information model for sensor description. The result of the effort is, with adding SWE to RESTful services, it will be more widely used in current platforms.

For other services in which sensor information is not properly structured, we can simply search the key words in the description if the accuracy is not required.

2) Exacting Factors From Application Run-time: We can use display-graph to measure the logic distance between two views. For example, view A is displayed on the screen. If view B can be directly switched from view A, we can draw a directed connection from A to B. For all the views displaying different sensor data, a graph can be drawn.

Figure 1 shows an example of an application's display-graph.

With the display-graph, we can determine the minimum distance, which means the steps from the current display view to this sensor display view. When it is 0, it means it is on display and when it is 1, it means the sensor is not on display but it may be on display after the user's next action. Simply, we can only acquire the data of sensors that is on display or to be on display.

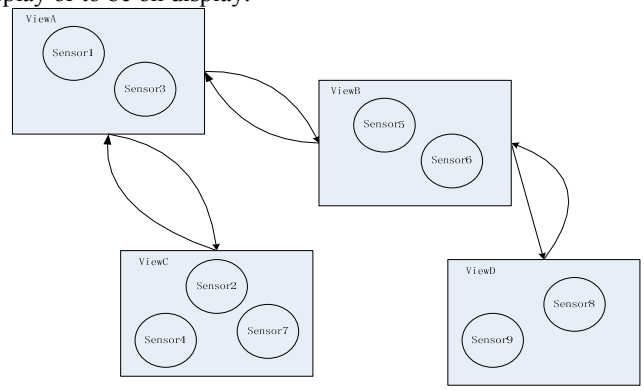

Figure 1. A display graph with four views

C. *Making Acquisition Self-adapted*

With the factors extracted from the sensor or the application, we can evaluate the MTUF for each sensor.

A table of factors should be prepared for each sensor like Table II below.

TABLE II. A SENSOR'S FACTORS TABLE

Factor	Value
Type	Temperature
Environment	In car
Display	No
Distance from display	1

Calculating the last MTUF with too many factors is a little complex, so we only discuss it with the limited factors: type, environment, alarm and display situation.

We consider the type and environment factors first. A MTUF1 can be assumed with those factors according to the common experience. For example, if Vmin is minimum value that can be tolerate for temperature display and in the in-car environment it needs Tvmin's interval to archive this change the MTUF1 = 1/Tvmin.

Second, we should consider the display situation. If the sensor is on display we can also still use MTUF1. If it is not, two situations should be discussed. When the sensor's

distance from the display view is 1, which means its view may be switched to later, we can lower the MTUF for not only cause too much acquisitions but also ensure the continuation of data display. We assume that MTUF2 = λMTUF1, which λ< 1. For other situations when sensor's distance from display view is bigger than 1, the MTUF2 can be assigned as 0 for it will not be displayed after the user' next action.

For some other factors we did not discussed above, MTUFs can also be specified for their own purpose, and we assume them as MTUF3, MTUF4 and so on.

The last MTUF will be made according to the above results. MTUF = max (MTUF2, MTUF3, MTUF4, …). Only in this way, all the requirements can be satisfied.

All the sensors' acquisitions are running at a frequency of MTUF and only a small part of these MTUFs are high. With this mechanism, system's overhead will be reduced to a low level.

IV. ARCHITECTURE AND EVALUATION

The architecture is illustrated in Figure 2. It is divided into layers that together cover the entire process, from information collection to information analysis, until sensor data acquisition. Next, the implementation of these layers is detailed.

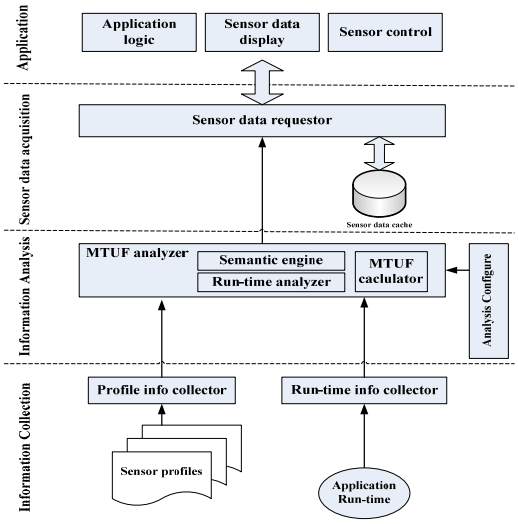

Figure 2. Layered architecture of data acquisition

A. *Information collection layer.*

The information collection layer provides collectors to collect information that may affect acquisition frequency of sensor data. It currently provides two types of collectors.

1) Profile information collector: It is used for collecting profile information of sensor. Usually, the profile of a sensor describes the detail of sensor such as type, location or other parameters. It is in either XML or JSON format. However the information collector doesn't make a distinction and the raw data will be pushed to the up layer.

2) Run-time information collector: It collects the information of the application run-time. In our design, it collects two types of information: the display graph of an application and the current view on display. Some works should be done before starting collecting. For example, building the view graph. Each view in the application should be assigned a tag number and the view graph consisting those tags should be configured first. As same as profile information collector, the run-time information will be sent to the up layer for advanced process.

B. *Information analysis layer*

The information analysis layer is the core layer of the architecture. It is composed of two parts. Analysis configure and MTUF analyzer.

1) Analysis configure: It keeps some values predefined for different scene. Mostly, it saves the MTUF for some certain factors. It can be seen as a MTUF table described in capture III. MUTF analyzer will read configures from this model and determine the MTUF at last.

2) MTUF analyzer: It is used to determine the MTUF of each sensor. It also contains three important parts: semantic engine, run-time analyzer and MTUF calculator.

Semantic engine analysis the sensor's profile and extract the proper factors. Usually XML is used in the profile to organize the data, and it only need to parse the XML file.

However for most profiles, just parsing XML is not enough. For example, the environment factor can be fetched from the "location" parameter of the profile and it can be many values such as: kitchen, bedroom or bathroom. One fact is that they all stands for the "indoor" environment and can be handled in the same way. To address this issue, we can build a semantic base for the words that will appear in the sensor descriptions. In the semantic base, each word has a property list attached to it. For example, the word "kitchen" has an "environment" property and its value is "indoor". That means the kitchen has an indoor environment and the MTUF will be computed as an indoor sensor. The semantic base can be stored in JSON [8] or XML [9] format in the mobile terminals or in a remote service, as shown in Figure 3.

```
{
  - kitchen: {
        word: "kitchen",
        environment: "indoor"
    },
  - playground: {
        word: "playground",
        environment: "outdoor"
    },
  - car: {
        word: "car",
        environment: "car"
    }
}
```

Figure 3. Data format in semantic base

The same way can also be applied to 'sensor type' and other factors. The semantic base can conclude different words to a same word. And the management for the up layer of the system will be sample.

Run-time analyzer analyzes the display graph of views and extract the factors affecting MTUF of run-time. To determine the distance of each sensor view and the current view, the view relationship should be loaded into the memory. Using the method in capture III, we will get the MTUF of this factor for each sensor.

All the factors extracted by semantic engine or run-time analyzer will be sent to MTUF calculator where the last acquisition frequency will be worked out.

C. *Sensor data acquisition layer*

It is the layer that performs actual acquisition of sensor data. It reads the MTUFs provided by information analysis layer and according the MTUFs of the different sensor, different acquisition frequencies will be applied to each sensor. This layer also keeps a cache to store history data of sensor. When new data is not arrived, cached data will be used.

D. *Application Layer*

The application layer uses data coming from lower layer. It maintains the application logic, displays sensor data and so on. It doesn't care about the acquisition of sensor data and just use it. Actually with the work of lower layers, developer can develop mobile applications easier and more efficient.

E. *Evaluation*

We studied our test scenario benchmarking by evaluating the network traffic when the application was running. For benchmarking, we record the network traffic with different sensor amounts in our design and compared it with the application ignoring sensor's differences.

The test scenario is applied for fetching the sensor information in a house. Firstly it fetches the sensor information from a platform that provides the sensor's description of location, type and so on. Then using the information, the fetch of sensor data can be started.

Figure 4 shows the result of our test.

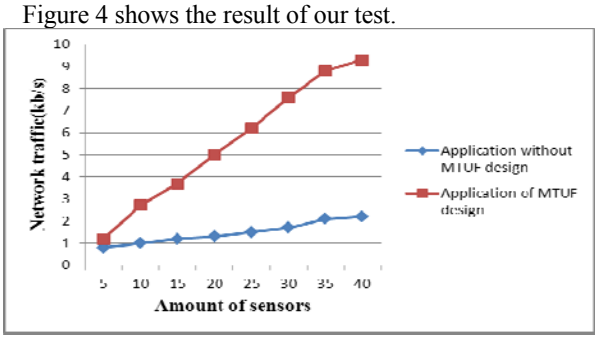

Figure 4. Testing results of MTUF design

In our test scenario, two types of design are displayed. One is our design using MTUF as sensor's update frequency in the application. Another one is the common design ignoring the differences of each sensor and updates them with a same frequency.

We can see that when the amount of sensors keeps increasing, the network traffic also increases greatly for a common design. As for the design using MTUF, the network traffic keeps in a low level.

V. CONCULSON

This paper depicted a mechanism to limit concurrent acquisitions of sensor data. It collects the factors affecting MTUF from a sensor profile or application run-time. With analysis of those parameters, MTUF can be calculated for each sensor. Without the support of platforms, our design can deduce the acquisition frequency itself, and when the factors are changed, MTUF will be changed too. So this design is self-adapting.

There are still several extensions that we can consider as the future work. One issue is that some of the information should be configured in the application. It is not convenient for application design. We plan to detect some information like view graph automatically and we'll make it more intelligent in the future work.

ACKNOWLEDGMENT

This work was supported by project on the Architecture, Key technology research and Demonstration of Web-based wireless ubiquitous business environment (No. 2012ZX03005008-001) , China-Finland Cooperation Project on the Development and Demonstration of Intelligent Design Platform Driven by Living Lab Methodology (No. 2010DFA12780) and project on platform, Key technology research of multi-terminal cooperative control network in wireless ubiquitous environment (No. 2011ZX03005-004-04).

REFERENCES

[1] Cosm, "Internet of Things Platform Connecting Devices and Apps for Real-Time Control and Data Storage," 2012; cosm.com

[2] R. T. Fielding, "Architectural styles and the design of network-based software architectures," doctoral dissertation, Dept. Information and Computer Science, Univ. California, 2000.

[3] Guohong Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," Knowledge and Data Engineering, vol. 15, pp. 1251-1265, Sept.-Oct. 2003.

[4] D. Guinard and T. Vlad, "Towards the web of things: web mashups for embedded devices," Proceedings of the International World Wide Web Conferences, Apr. 2009.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, Hypertext Transfer Protocol -- HTTP/1.1, IETF RFC 2616, June 1999; www.ietf.org/rfc/rfc2616.txt.

[6] V. Loia, G. Fenza, D. Furno and C. De Maio, "Swarm-based Approach to Evaluate Fuzzy Classification of Semantic Sensor Data," Pervasive Computing and Communications Workshops, IEEE CS, 2012, pp. 308-313.

[7] M. Botts, G. Percivall, C. Reed and J. Davidson, "OGC® Sensor Web Enablement: Overview And High Level Architecture", Proceedings of the 5th International ISCRAM Conference, 2008, pp. 713-723.

[8] D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), IETF RFC 4627, July 2006; www.ietf.org/rfc/rfc4627.txt.

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Extensible Markup Language (XML) 1.0, World Wide Web Consortium (W3C) note, Novemember 2008; www.w3.org/TR/REC-xml/.