MPSDataStore: A Sensor Data Repository System for Mobile Participatory Sensing

Junya Niwa, Kazuya Okada, Takeshi Okuda, Suguru Yamaguchi
Graduate School of Information Science
Nara Institute of Science and Technology
Takayama 8916-5, Ikoma, Nara, Japan
niwa2jun@gmail.com, {kazuya-o, okuda, suguru}@is.naist.jp

ABSTRACT

The development of wireless technologies, such as 3G and Wi-Fi, and the rapid growth of mobile devices equipped with sensors have enabled the practical use of Mobile Participatory Sensing (MPS). By gathering and utilizing sensor data using mobile devices, the deployment cost of services can be reduced. In the context of MPS, it is important to establish a method of storing and locating sensor data collected by millions of mobile devices.

In this paper, the development of a sensor data repository system for a large-scale MPS platform is proposed. By storing sensor information in the mobile device's storage, the storage cost can be distributed. The proposed method of tracking the acquisition locations of sensor data can reduce management costs. In addition, a cache mechanism that can minimize duplicate transmissions of sensor data from mobile devices due to overlapping queries is introduced. Based on a two-day simulation, the proposed method can reduce the management cost of the acquisition locations by 80%. Furthermore, the cache method can reduce the transmission of duplicated sensor data on mobile devices.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Algorithms, Design, Experimentation, Human Factors, Performance

Keywords

Mobile Device, Participatory Sensing, Cloud Computing

MCC'13, August 12, 2013, Hong Kong, China.

1. INTRODUCTION

Due to the rapid growth of mobile devices equipped with sensors, Participatory Sensing has received widespread attention in the field of sensing. In Participatory Sensing, services gather information by using mobile devices owned by individuals. Service providers can reduce costs by using resources owned and obtained by mobile device users. In addition, by combining the sensor information with geographical positions, the practical use of the sensor information by searching with geographical locations is possible.

The storage cost of the concentrated central storage server is problematic when the server collects sensor information on a daily basis and for a long time. The rapid growth in the sales volume of mobile devices is expected to reach 50% in the year 2015 by the Japanese government [5]. We propose Mobile Participatory Sensing Data Store (MPS-DataStore), a distributed storage of mitigating the increase of storage cost due to the growth of mobile device users.

MPSDataStore stores sensor information in the mobile devices storage. When service providers send a query to get the sensor information, MPSDataStore transfers the query to the mobile devices which have stored the designated sensor information. Thus, the proposed system can scale-out storage by storing sensor information in mobile devices. In addition, mobile devices send sensor information when and only when service providers require the information. MPS-DataStore can reduce the management cost of the acquisition locations while it enables geographical range searches for retrieving sensor informaton. Furthermore, by using the cache function, MPSDataStore can reduce the size of sensor information transmitted by mobile devices.

The rest of this paper is organized as follows: The requirements and the related works are described in Section 2. We then present the design of the MPSDataStore in Section 3. In addition, the simulation experiments and results are described in Section 4. And Section 5 concludes the paper.

2. RELATED WORK

In this secton, we summarize services that use participatory sensing, and the requirements for realizing the services for sensor information sharing systems. In addition, past research on the sensor information gathering and storing methods using mobile devices are presented.

2.1 Requirements

We focus on the services realized by using community sensing. Community sensing is done by gathering sensor information from a number of users and analyzing that infor-

mation. For example, Ear-Phone [7] uses the smartphone's mic for gathering noise pollution data and stores this data on a central system implemented on a server. Compared with using conventional fixed sensors, community sensing in this case enables service providers to lower the cost of services. In addition, users can obtain noise pollution information by specifying only the geographical range and time. Ear-Phone needs a wider geographical range and a higher frequency of sensing. Therefore, MPS storage systems have to store as much sensor data as possible. Moreover, the system has to be able to retrieve the sensor data by range search.

OurMobilePlanet [2] has reported that the penetration rate of smart phones in Japan was 20% in 2012 Q1, which is equivalent to approximately 25 million smartphone users in the country. When 25 million smart phones gather sensor information every minute and upload them to a concentrated storage server, the server would require 3Gbps of bandwidth and 1,350GB/h of storage. The storage size will be approximately 12PB per year. The management cost of sensor data is calculated by multiplying the number of the locations one mobile device sensed by the number of mobile devices. According to the calculation, 36 billion sensor data are needed to be managed in one hour. The management cost will be approximately 13 trillion sensor data in a year. As the rapid growth of the penetration rate of smart phones is expected, on-demand sensor information transmission and the reduction of the management cost are required.

Consequently, the requirements for sensor data storage systems used by such services are as follows:

- Storing sensor data without compromising scalabillity.
- Tracking the data stored in mobile devices.
- On-demand sensor data transmission.

2.2 Sensor Data Storage Systems

Ear-Phone or other existing services typically use the concentrated data storage method. In the concentrated method, mobile devices gather the necessary sensor information by service providers and upload these to concentrated storages. Using this method makes it easy for service providers to manage data flow. However, they end up spending more costs on storages due to the increase in the number of users participating in the sensing.

To solve the problem, distributed sensor data sharing methods using peer-to-peer (P2P) networks have been proposed. P2P network nodes manage routing tables evenly to share the computation and communication cost. There are two types of P2P networks for implementing a sensor data storage system. One is constructing a P2P network among mobile devices and storing sensor data in the network. For example, LL-Net [3] constructed a P2P network among mobile devices. LL-Net provided geographical range search capability. The other is constructing a P2P network between PCs owned by users and storing sensor data in the network. As an example, Mill [4] provides a range search to retrieve sensor data taken within the range. The distributed method can share the storage cost by storing sensor data in the P2P network. On the other hand, the method LL-Net adopts needs to consider the characteristics of mobile devices such as network instability. When network connectivity is unstable, the management cost of the P2P network is undeniable. Furthermore, the method employed by Mill requires mobile devices to upload sensor data, therefore the on-demand sensor data transmission is compromised.

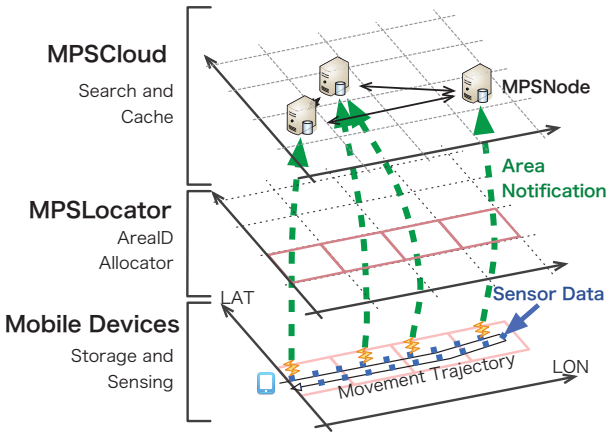

Figure 1: The overview of the MPSDataStore.

3. MPS DATA STORE

We propose a sensor data repository system, MPSDataStore that will collect and store sensor information through users' mobile devices. These mobile devices obtain information every minute, and attach the time and location of where and when the data is acquired. The mobile device stores all sensor data and transmits these only upon receiving queries from the service provider.

MPSCloud uses the IaaS cloud and enables geographical range and time range search functions. MPSDataStore enables on-demand sensor data transmission by separating the search function and storage function between the MPSCloud and the mobile devices.

As shown in Fig. 1, the mobile device travels along the movement trajectory of the user and senses data every minute. Immediately after sensing, the mobile device obtains the geographical location using GPS. Then, the mobile device converts the longitude and latitude information into an AreaID by using MPSLocator. An AreaNotification is the act of notification of sensor data acquisition in the area. If the mobile device has not sent an AreaNotification within the perimeter, it will send this notification to the MPSCloud. Mobile devices' IP address and AreaIDs have to be included in an AreaNotification. It is then transferred to an MPSNode, which manages the particular area. An MPSNode is a virtual machine dynamically created by the MPSCloud and manages the Device Table. The Device Table includes information on which mobile device has the sensor data and in which area the information was acquired. The MPSCloud and all of the mobile devices have an MPSLocator, and all have the same scheme of ID allocation of areas and parameters. The MPSCloud will know, therefore, which mobile device has the sensor data and in which area it was sensed by using the MPSLocator.

3.1 Reducing Management Costs

Managing the equivalence of the mobile device and sensor data acquisition location is needed to track sensor data stored in mobile devices. However, tracking all sensor data acquisition locations is not necessary as the management

cost of the locations have to be considered. Therefore, we designed an MPSLocator which will reduce the amount of sensor data acquisition locations managed by MPSCloud.

The MPSLocator divides the target region and allocates an ID for each divided area using Z-ordering [6]. By this method, it can convert multi-dimension values to a one-dimension value at a low computational cost. In this case, we used Z-ordering because mobile devices need to convert the location to an AreaID every minute. We define ZBits to specify the scale of each area. ZBits are assigned bits converting longitude and latitude information to AreaIDs. Considering a binary representation of longitude and latitude ($x_1x_2x_3x_4x_5\cdots$, $y_1y_2y_3y_4y_5\cdots$) when ZBits=2 the representation of the AreaID would be $x_1y_1x_2y_2$.

The range of the target area is given by the maximum and minimum values of longitude and latitude. The MPSCloud and all mobile devices share the ranges and ZBits. By sharing the information, mobile devices can voluntarily send AreaNotifications to the MPSCloud.

3.2 Distributed Management of Device Tables

The MPSCloud manages the Device Table, which includes the AreaID and mobile devices' IP addresses. The maximum cost of management is calculated by multiplying the number of mobile devices by the number of areas, divided by the MPSLocator. Therefore, the management cost of the Device Table has to be distributed in situations where the number of users increases.

The MPSCloud uses an IaaS cloud to allocate computation resources dynamically. To understand the distributed management of geographical location and geographical range search, constructing a P2P network between virtual machines is needed. For example, Akiyama et al. [1] proposed a distributed geographical data storage method that can balance the load of storage cost among the nodes. The method uses Z-ordering to convert 2-d location information into 1-d Z value. Then the Z value is used for SkipGraph's key space. The method enables geographical range search by using Z-ordering, and load balancing among nodes by allocating resources by properly considering the growth of the geolocation data. However, the method does not balance the load if there is too much geolocation data in a single key. Therefore, we propose the MPSCloud Network that can balance the load of managing the Device Table by allocating more than one MPSNode in an area.

Fig. 2 shows the process of the Device Tables being managed by an MPSNode in an MPSCloud Network. Three mobile devices send an AreaNotification to AreaIDs 0 to 7. First, device **A** sends an AreaNotification to AreaIDs 1, 5 and 6. Secondly, device **B** sends a notification to AreaIDs 6 and 7 and finally, device **C** sends a notification to AreaID 4. We name the MPSNode affiliated with the smaller AreaID at the linked list of level 0, as the left neighbor. Let $AreaID$ be the joined AreaID in the SkipGraph, and $AreaID_L$ be the joined AreaID for the left neighbor. Each MPSNode manages mobile devices, which obtained the sensor data in $AreaID_L < AreaID_T \leqq AreaID$.

Now, we describe the method of joining new MPSNodes in order to distribute the load in an area with concentrated AreaNotification. The MPSCloud has a special virtual machine called a "Dispatcher". When the MPSCloud boots, the Dispatcher generates an initial MPSNode. The initial MPSNode joins the $AreaID = max(AreaID)/2$. The Dispatcher and all the MPSNodes have a system parameter MaxNotification, which is the maximum number of AreaNotifications an MPSNode manages. If an MPSNode receives more notifications than the MaxNotification, the node sends a DivideRequest to the Dispatcher. The DivideRequest includes a new AreaID at which the new MPSNode joins, IP address and $AreaID$ of the requested MPSNode. The requested MPSNode computes the new AreaID where the new MPSNode is affiliated. The new AreaID can be an AreaID where the number of AreaNotifications managed by the requested MPSNode is divided in half. If there are more than one AreaID candidates, then an AreaID having the most AreaNotifications managed by the new MPSNode is selected. The insert operation used by the network is the same as the one SkipGraph uses.

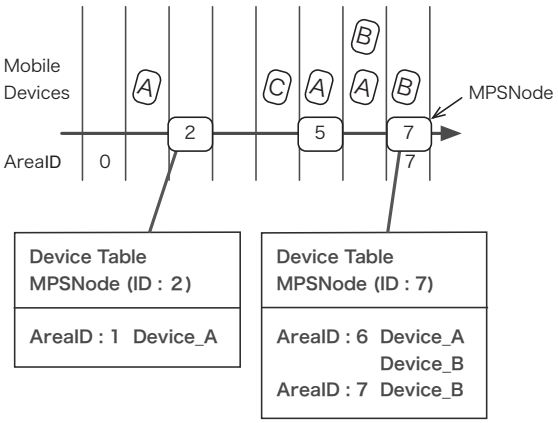

Figure 2: An example of the Device Table.

In this paragraph, we describe how the Device Table is transferred to a new MPSNode. The process of dividing an area can be classified in two types where each type uses a different method to link to the network.

- Non-overlapping new AreaID
  The AreaID that is not overlapped with the requested MPSNode.
- Overlapping new AreaID
  The AreaID that is overlapped with the requested MPSNode.

When the non-overlapping AreaID is selected, the new MPSNode performs the insert operation in a way that is similar to SkipGraph, and the Device Table is then transferred from the requested MPSNode. When the overlapping AreaID is selected, more than one MPSNode manages similar areas. In this case, multiple MPSNodes virtually join as one MPSNode. We call the MPSNode joined at the AreaID of initial interest, the master node, and consider the rest as slave nodes. Moreover, slave nodes notify the master node of their IP addresses, to join the network. The master node transfers the queries to all slave nodes. Then, the new MPSNode receives a half or the closest number of AreaNotification from the requested MPSNode in each area.

The division operation in the area of multiple nodes joined can be classified as:

- New AreaID where the requested nodes joined.

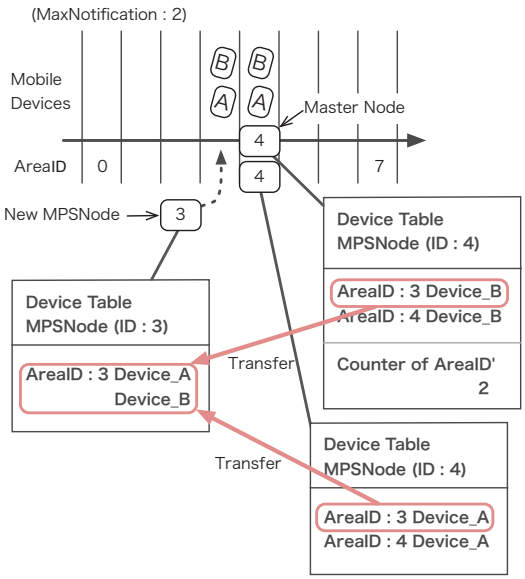

Figure 3: The procedure of the divide operation at AreaID'.

- New AreaID where the requested nodes did not join.

The transfer operation when allocating the AreaID the requested nodes joined at is done by using the same operation as the overlapping AreaID. In contrast, the transfer operation when allocating an AreaID, which is not the same as requested nodes', needs to consider all of the slave nodes' Device Tables. The master node needs to know the number of the AreaNotifications managed by the nodes, to determine the new AreaID for sharing the Device Tables evenly. Therefore, we add the count function to the master node. Let $AreaID'$ be the AreaIDs which are managed by an MPSNode except for the AreaID of joined. From this point, the master node sends a divide request when the number of AreaNotifications in $AreaID'$ exceeds half of the MaxNotifications. Moreover, all MPSNodes at particular AreaIDs transfer DeviceTables to new MPSNodes. Fig. 3 shows the procedure of the division operation at $AreaID'$. As shown in the figure, two MPSNodes joined at AreaID 4 manage two AreaNotifications in $AreaID'$ which excluded MaxNotification/2. Therefore, the master node sends a Divide Request and selects new AreaID 3 the next to master node. Then, a new MPSNode requests a transfer of Device Table to the master node. When a master node gets a transfer request in $AreaID'$, it will transfer the request to the slave nodes and all MPSNodes sends the DeviceTable in $AreaID'$ to a new MPSNode. Using this procedure, the new MPSNode then receives two AreaNotifications in AreaID 3.

3.3 Design of MPSCache

To reduce the sensor data size transmitted from the mobile devices, we designed a cache function called MPSCache. MPSNodes use MPSCache to extract cached sensor data upon receiving a query. An MPSNode stores a set of queries and sensor data from mobile devices in the MPSCache. The MPSCache stores sensor data by using the Least Recently Used (LRU) method, when the free space of the cache storage was lesser than the amount of the sensor data. If the sensor data size obtained by a query exceeded all cache storage size allocated on an MPSCache, then the MPSCache does not store the sensor data.

Table 1: An example of the information specified in queries.

Query ID	AreaID List	Target Time	Sensor Type
A	1-5, 10-15	From 8:00, 10 minutes	GPS and WiFi
B	4-8	From 8:05, 10 minutes	GPS and WiFi
C	1-5, 10-15	From 8:10, 10 minutes	GPS and WiFi

Table 2: An example of the queries made by MPSCache.

Query ID	AreaID List	Target Time	Sensor Type
D	6-8	From 8:05, 10 minutes	GPS and WiFi
E	4-5	From 8:10, 5 minutes	GPS and WiFi

3.3.1 *Read from Cache*

The procedure of reading sensor data from MPSCache is as follows:

1. Extract cached queries overlapping in both target AreaIDs and target time.
2. Extract cached sensor data overlapping in both target AreaIDs and target time.
3. Create new queries which include non-overlapping AreaIDs and target time.

The MPSCache confirms if the overlapping queries are cached or not in the first procedure. If the overlapping queries are not cached, the MPSNode sends the query to the mobile device directly. If the overlapping queries are cached, the MPSCache proceeds to the next operation. In the second procedure, the MPSCache needs to extract cached sensor data which overlaps in both target AreaIDs and target time with the query. The MPSCache, therefore, needs a database capable of associating AreaIDs and sensor data as well time and sensor data. Table 1 shows an example of information specified in a query. Query A targeted AreaIDs from 1 to 5 and from 10 to 15.

At first, the MPSCache verifies the target AreaIDs that are overlapping or not in the first procedure. First, assume that query A is cached in an MPSCache. Then, query B is overlapping in AreaIDs from 4 to 5 with query A. Additionally, query C is overlapping in all AreaIDs with query A. Therefore, query B and C is overlapping in target AreaIDs with query A. Next, the MPSCache extracts queries overlapping in target time from the queries overlapping with AreaIDs. Query B is overlapping in the time from 8:05 to 8:09 with query A. In contrast, query C is not overlapping with query A in terms of the time. Consequently, the MPSNode sends query C to mobile devices directly and does not extract sensor data from the MPSCache. The MPSCache continues to the second procedure because query B is also overlapping in sensor types. In the second operation, the MPSCache extracts cached sensor data. The MPSCache has extracted overlapping cached queries, so in the next phase, the MPSCache extracts an overlapping part with the query.

The MPSCache needs to issue some new queries for the non-overlapping part of the cached query. Therefore, in the last procedure, the MPSCache creates new queries, which include the non-overlapping part of the cached query. The procedure of creating new queries by using queries A and B are described in Table 1. Assume query A is cached in an MPSCache. Query B is overlapping with query A only in AreaIDs from 4 to 5, and time wise for 5 minutes, from 8:05 to 8:09. Therefore, the MPSCache creates new queries which do not include the overlapping part of query B with query A. Table 2 shows an example of the queries created by the MPSCache. Query D includes AreaIDs from 6 to 8 with the non-overlapping range with query A. Query E includes AreaIDs from 4 to 5 and the 5 minutes, from 8:10 to 8:14, which is not overlapping with query A.

3.3.2 *Write in Cache*

The MPSNode transfers the queries which do not collide with cached queries. The mobile devices then reply to the MPSNode with the sensor data matching the queries. The MPSCache stores a pair of a query and sensor data received from the MPSNode. The MPSCache obeys the following rules for storing sensor data:

- The MPSCache does not store sensor data if the sensor data size exceeded the allocated cache storage size.
- The MPSCache stores the sensor data by using the LRU method if the sensor data size exceeded the free space of the cache storage.

Therefore, the MPSCache stores sensor data at any cost, except for that sensor data size exceeding allocated cache storage size.

4. EVALUATION

In this section, we discuss the evaluation of the MPSDataStore through experiments conducted via simulation.

Table 3 is the list of parameters we used in the experiments. To confirm that the MPSLocator can reduce the number of locations managed, we performed a two-day simulation. We simulated movements of the mobile devices and the processes of gathering sensor data for the first day. As for the second day, querying from the service provider was simulated in addition to the movement simulation. We used PFlow data[8] for the movements and the querying. People Flow project reconstructs and accumulates PFlow data from person-trip data which is gathered by questionnaires in Japan. The data includes ID of the person and the one day trajectory data. Because we need 2 day trajectory data, for the second day, we used the PFlow data which was not included in the first day.

The target range of the MPSDataStore is a 96km×116km in Tokyo metropolitan area where 20 bits were used to represent longitude and latitude information. To confirm that the MPSDataStore can distribute the load of managing locations, we used 30% of mobile devices for the MaxNotification number. Moreover, we implemented ZBits ranging from 1 to 13 bits with 1 bit difference, because the area size becomes less than 10m×10m if more than 14bits is used. Furthermore, the values will become too small considering GPS data error. Consequently we assumed that the size of data sensed by a mobile device is 1KB, and the unit of the sensor data size taken by a query is 1KB as well. The target time range is set 10 minutes before the current time, as we assumed the service requires the most recent sensor data.

Table 3: Parameters of the simulation.

Component	variable	value
Simulation	Moving Time	1-1440 (1 day)
	Quering Time	1441-2880 (1 day)
	Target range	Tokyo metropolitan area (96km×116km)
	#bit for latitude	20
	#bit for longitude	20
	#Mobile Devices	[2000,4000,6000,8000,10000]
MPSDataStore	MaxNotification	30% of #Mobile Devices
	ZBits	From 1bit to 13bit
ServiceProvider	Frequency of quering	6 times in a day per 1 service user
	Target time	from 10 minutes ago to current time
	Target range	100m×100m with center current position
	Sensor data amount	1KB per sensing
	Service Users	4× #Mobile Devices

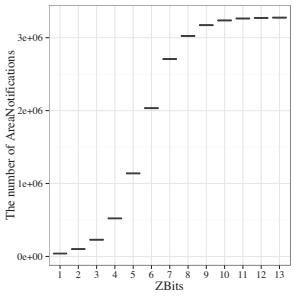

Figure 4: The number of the AreaNotifications managed by MPSDataStore. (MaxNotification = 30% of mobile devices)

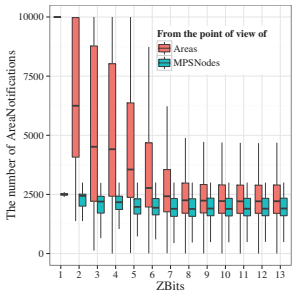

Figure 5: The variability of the AreaNotifications. (#MobileDevices=10,000, MaxNotification=3,000)

4.1 Cost of Location Management

Fig. 4 shows the number of the AreaNotifications managed by the MPSDataStore. When the ZBits value is low, the number of AreaNotificatons managed by the MPSCloud is fewer because the number of AreaNotifications sent by each mobile device is fewer. On the other hand, when the ZBits value is large, the growth of the number slows down. This indicates that the biased distribution of the movement of people lessened the growth of the AreaNotification that needed to be managed. According to the simulation, the number of sensor data collected in two days was 28.8 million and the number of AreaNotifications managed was less than 3.28 million. Hence, MPSLocator can reduce the location information requiring management by at most 80%.

Fig. 5 shows the variability of the AreaNotification sent by mobile devices and managed by the MPSCloud. In this figure, the graph presents the number of AreaNotifications sent to each AreaID, after the non-sensed area was removed. Table. 4 shows the interquartile range of the AreaNotification from the point of view of the areas and the MPSNodes. Comparing the interquartile range with the areas and the MPSNodes, the variability of AreaNotifications managed by MPSNodes is less than the variability of the AreaNotification sent to the areas except for when $ZBits = 1$. This means that the MPSDataStore can smoothen the concentrated load of managing AreaNotifications and therefore, balance the load of managing AreaNotifications.

Table 4: The variabillity of the AreaNotification. (#MobileDevices=10,000, MaxNotification=3,000)

ZBits	areas		MPSNode	
	median	inter-quartile range	median	inter-quartile range
1	9993	6	2501	29
2	6313	2736	2452	235
3	4509	3253	2182	361
4	4514	2767	2194	275
5	3548	2003	1969	318
6	2733	1344	1957	344
7	2385	861	1898	386
8	2216	644	1874	383
9	2225	585	1885	358
10	2187	593	1873	357
11	2236	577	1908	384
12	2200	600	1895	353
13	2224	597	1896	366

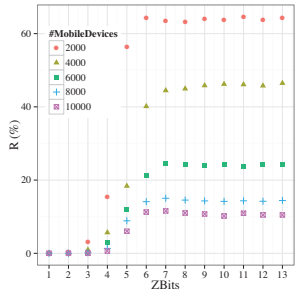

Figure 6: Reduction ratio of the sensor data size transmitted by mobile devices. (Cache storage size = 100KB)

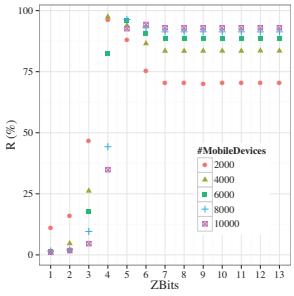

Figure 7: Reduction ratio of the sensor data size transmitted by mobile devices. (Cache storage size = 1MB)

4.2 The effect of MPSCache

Figures 6 and 7 show the reduction ratio of the sensor data size transmitted by the mobile devices through the MPSCache. The reduction ratio R was calculated using Eq.1.

$$R = \frac{\text{Transmitted size without MPSCache}}{\text{Transmitted size with MPSCache}} * 100 \qquad (1)$$

When the cache storage size per MPSNode is 1MB, up to approximately 1.6GB of cache storage was allocated in the MPSCloud. The size of the sensor data transmitted by 10,000 mobile devices was 180GB. By using the MPSCache, the size of the sensor data transmitted by mobile devices was reduced to approximately 12.6GB, meaning that approximately 18.4MB data transmitted by a mobile device is reduced to approximately 1.3MB on average. By allocating approximately 1.6GB of cache storage, more than 60% of the sensor data transmitted by mobile devices can be reduced. Consequently, using the MPSCache can reduce the size of sensor data transmitted by mobile devices.

5. CONCLUSION

In this paper, we proposed a sensor data repository system MPSDataStore. MPSDataStore can distribute the storage cost by storing sensor data among mobile devices. The MPSLocator can reduce the management cost of the sensor data acquisition locations. In addition, the MPSCloud can balance the load of managing location information by using the IaaS cloud. Moreover, MPSCache can reduce the sensor data amount transmitted by mobile devices. By conducting a two-day simulation, we have shown that MPSDataStore can fulfill the requirements.

For a long-term sensor data collection, MPSDataStore needs to be improved by considering the character of the mobile devices. Mobile devices that use 3G or WiFi networks change IP addresses frequently. In the current implementation, MPSDataStore does not consider this changes in IP addresses. If a mobile device changed its IP address, the MPSCloud is not able to communicate with the device, and the sensor information stored in the device can not be reached as a consequence. Furthermore, users of smart phones change their devices in a certain period of time. If a smart phone stored important sensor data, that data eventually goes to waste, therefore the continuity of the service can be compromised. We then need to consider a method to save the sensor data stored in those devices.

6. REFERENCES

[1] A. Daisuke, K. Hosokawa, A. Kota, I. Hayato, and M. Toshio. An efficient distributed management scheme for 2d location information using z-curve. *SIG Technical Report*, 2010-IOT-8(9):1–6, Feb. 2010.

[2] Google. Our mobile planet. `http://www.thinkwithgoogle.com/mobileplanet/ja/`.

[3] Y. Kaneko, K. Harumoto, S. Fukumura, S. Shimojo, and S. Nishio. A location-based peer-to-peer network for context-aware services in a ubiquitous environment. *International Symposium on Applications and the Internet Workshops (SAINTW'06)*, 0:208–211, 2005.

[4] S. MATSUURA, K. FUJIKAWA, and H. SUNAHARA. Mill: A geographical location oriented overlay network managing data of ubiquitous sensors. *IEICE Trans. Commun.*, 90(10):2720–2728, Oct 2007.

[5] Ministry of Internal Affairs and Communications. Information and Communications in Japan. `http://www.soumu.go.jp/johotsusintokei/whitepaper/eng/WP2012/2012-index.html`.

[6] G. Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, 1966.

[7] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu. Ear-phone: an end-to-end participatory urban noise mapping system. In *Proc. of IPSN*, IPSN '10, pages 105–116, New York, NY, USA, 2010. ACM.

[8] Y. Sekimoto, R. Shibasaki, H. Kanasugi, T. Usui, and Y. Shimazaki. Pflow: Reconstructing people flow recycling large-scale social survey data. *Pervasive Computing, IEEE*, 10(4):27–35, 2011.