International Journal of Database Theory and Application
Vol.9, No.7 (2016), pp.65-74
http://dx.doi.org/10.14257/ijdta.2016.8.07

An App-Driven Garbage Collection to Enable Continual Vehicular Sensing in the Android Smartphone Apps

Hoa-Hung Nguyen, Rachmad Nafisholeh and Han-You Jeong

Pusan National University, Busan, Republic of Korea
nguyenhoahungit@gmail.com, rachmadnafisholeh@gmail.com, hyjeong@pusan.ac.kr

Abstract

Thanks to their multiple sensors, the Android smartphones have drawn significant attention for supporting vehicular sensing apps. These apps usually require continual sampling of sensors to collect the events of driving. However, the standard garbage collection (GC) of the Dalvik virtual machine blocks these apps from collecting sensing information for hundreds of milliseconds, which leads to the failure of detecting vehicular events. To alleviate this problem, we present a scheme that proactively invokes the app-driven GC to reduce the duration of blocking based on free heap memory ratio or periodic calling. The experimental results show that we can significantly reduce 75% of the blocking time.

Keywords*: Vehicular sensing, smartphone, Android, garbage collection*

1. Introduction

Vehicular sensing applications (VSA) are becoming important functionalities in intelligent transportation systems. These applications provide various information about the road conditions which can be used by advanced driver assistance system (ADAS) or road authority. In VSA, each vehicle plays a role of a sensors in gathering, processing and sharing the location-relevant data such as road surface anomaly [1]. These applications require (1) high sampling-rate sensors to gather sufficient samples, (2) computational power to process the raw sensor samples and to detect target event, and (3) communication ability to share the event information. In this paper, we focus on the road surface anomaly detection applications, particularly speed bumps and potholes detection, in which the requirement of sensor sampling rate is critical.

In recent years, the Android smartphones have become promising platforms for VSA [2-3]. They are usually equipped with GPS, accelerometer sensor, gyroscope sensor, and camera, which fulfill the requirements of high sampling rate in VSA. For example, they can provide accelerometer and gyroscope sample whose sampling rate is up to *200* Hz. Together with continuous improvement in their processing power and integration of various communication technologies such as Wi-Fi and LTE, they are becoming comparable with the dedicated hardware platform for VSA [4-5]. On the other hand, the availability of the Android smartphones has enabled a large-scale deployment of VSA where each vehicle on the road can become a sensing source.

In the Android smartphone, VSA can be implemented as a module in the ADAS app. In the road surface anomaly detection, the accelerometer samples are continuously accessed and processed at high rate to detect the existence of speed bumps and potholes. The results of this detection can be used directly by the ADAS or can be sent to central server.

However, apps in Android-based smartphones are regularly blocked by garbage collection (GC) of the Dalvik virtual machine (VM) which leads to consecutive sensor sample losses. In the Android smartphones, the GC reclaims memory space occupied by

ISSN: 2005-4270 IJDTA
Copyright ⓒ 2016 SERSC

objects that are no longer in use. When there is not enough free memory space for a new object, Dalvik VM automatically invokes GC to gather the unused memory space which is known as GC_FOR_ALLOC [6] and blocks the whole app during which GC_FOR_ALLOC runs. As a result, a vehicular sensing app may not get the sensor samples during the blocking time, which leads to the failure in road anomaly detection.

There exists a few works [7-8] attempting to alleviate the blocking of GC by modifying the Android operating system (Android OS). In [7], the real-time patch is added to linux kernel of Android OS besides the Android app framework to support the real-time apps, therefore, the apps do not suffer from GC blocking. The authors of [8] suggest to modify Dalvik GC to reduce its blocking time. However, all of these works require to change the Android OS itself, *i.e.*, they cannot be applied to existing Android apps. Moreover, all existing works in road anomaly detection on the Android smartphones have not considered the impacts of GC blocking on the detection performance.

In this paper, we first address how the Android smartphone can be used to detect road anomalies. Then, we figure out the problem of road anomaly detection in the Android smartphones and propose two approaches to reduce the blocking time of GC by proactively calling the explicit GC which is known as GC_EXPLICIT. The free heap memory ratio approach invokes GC_EXPLICIT when free heap memory ratio is below a given threshold while the periodic calling approach invokes GC_EXPLICIT periodically with a given period. Since these approach uses the built-in functionality of the Android OS, it is applicable to existing Android apps. In addition, we attempt to find the optimal threshold and calling period in which the GC blocking time is low while GC does not frequently run. The experiment results show that, the optimal setting from our approaches reduces *75* % of the blocking time of GC.

The rest of this paper is organized as follows: In Section 2, we describe the detection of road anomalies. Section 3 provides the detailed description of our approaches to reduce the loss of sensor sample. The experimental results are discussed in Section 4. Finally, we conclude our paper in Section 5.

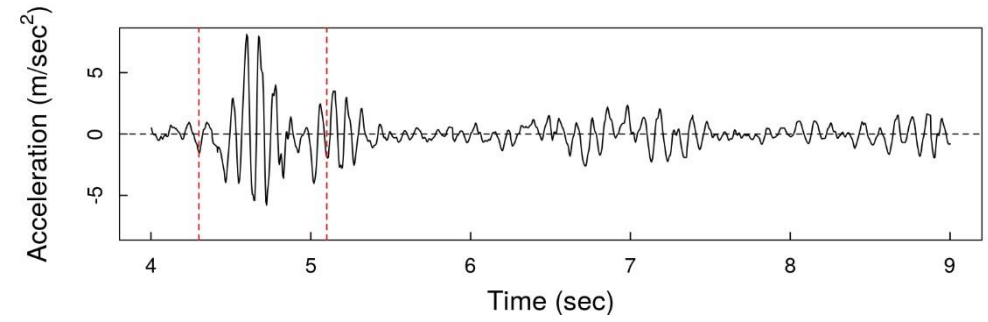
2. Vehicular Sensing in the Android Smartphones

2.1. Detection of Speed Bumps and Potholes

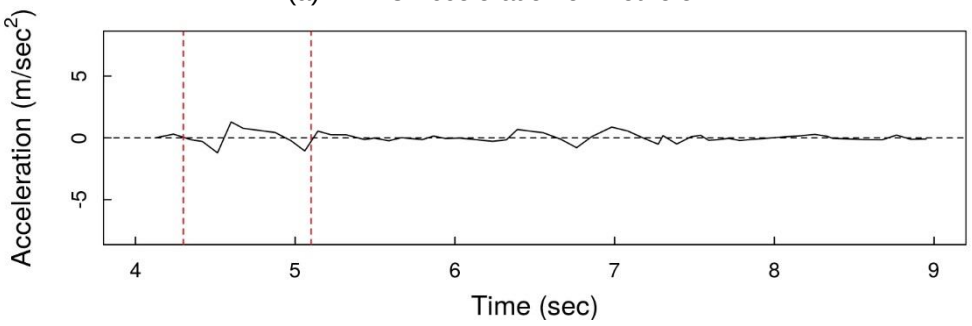
In this section, we describe the detection mechanism of speed bumps and potholes in the Android smartphone. There are several methods in the literatures to detect speed bumps and potholes using smartphone accelerometer sensors. They can be detected based on the different visual appearance [9] from the road surface, or based on the acceleration perpendicular to the ground surface, which is generated when the vehicle travels through these road anomalies [2-3]. The latter approach is suitable for smartphones since it requires less computational power. To simplify the description, we assume that the smartphone is mounted on a vehicle in the way that the direction of gravity vector has opposite direction with smartphone's local x-axis. With this assumption, acceleration caused by speed bumps and potholes impacts mostly on the x-axis. The approach that we use in our app is based on peak-to-peak average acceleration value from accelerometer sensor.

Figure 1 shows a detection example of speed bumps and potholes using accelerometer sensor. Figure 1(a) shows the x-axis acceleration value when the vehicle travels over a potholes. The interval T_P that the vehicle crosses over the potholes is *(4.4, 5.1)* sec. Because of the vibration of the vehicle and smartphone mounting device, there is always oscilation in accelerometer value. Therefore, the change in acceleration from smartphone measument itself can not be used directly to indicate the existence of speed bumps.

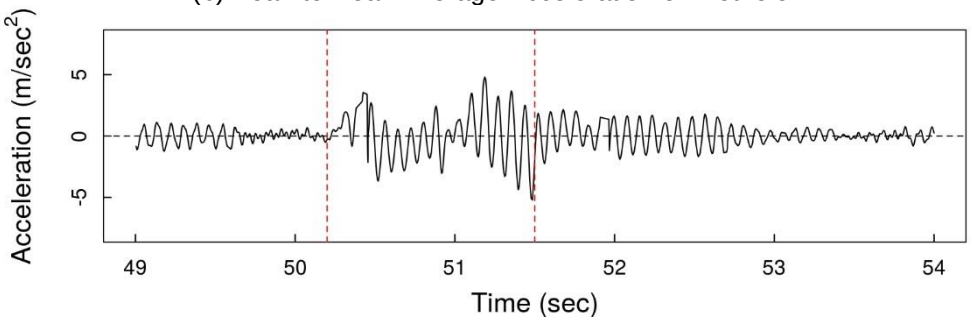
 Copyright ⓒ 2016 SERSC



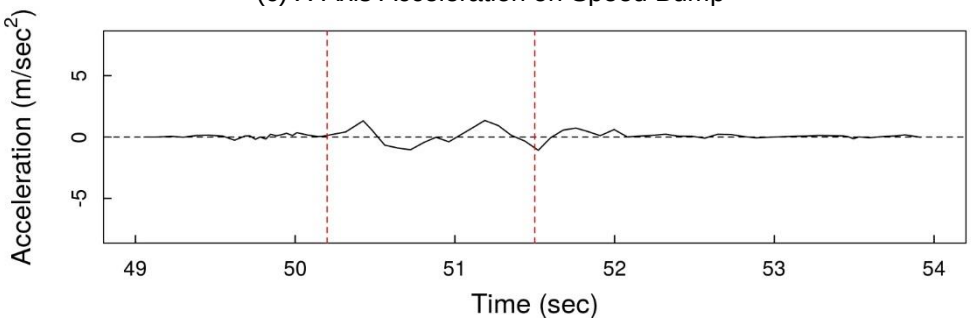
(a) X-Axis Acceleration on Pothole



(b) Peak-to-Peak Average Acceleration on Pothole



(c) X-Axis Acceleration on Speed Bump



(d) Peak-to-Peak Average Acceleration on Speed Bump

Figure 1. Pothole Detection Using the Android smartphone

However, the average value acceleration in one oscillation period reflects the actual change caused by speed bumps and potholes. We call this value as peak-to-peak average acceleration.

Figure 1(b) shows the peak-to-peak average acceleration of the corresponding acceleration in Figure 1(a). We can see that there is a clear pattern when the vehicle enters and leaves the potholes. First, when the front wheels enter the potholes, the vehicle moves downward which means accelerating to the opposite direction of smartphone x-axis, when

Copyright ⓒ 2016 SERSC

the front wheels leave the potholes, the vehicle moves upward. When the rear wheels enter and leave the potholes, they create a similar pattern as front wheels.

Similarly, Figure 1(c) and Figure 1(d) show the acceleration and peak-to-peak average acceleration when the vehicle travels over speed bumps. The interval T_B that the vehicle crosses over the speed bump is *(50.2, 51.5)* sec. A reverse pattern can be seen from the peak-to-peak average in Figure 1(d), except that the pattern is reversed. The reason is that the vehicle accelerate in the same direction of smartphone x-axis first and then the opposite direction when the wheels travel over speed bumps.

In order to categorize a given period of road segment as the segment with speed bumps, with potholes or normal segment, a threshold-based or a machine learning approach can be applied to the pattern of peak-to-peak average acceleration.

2.2. Continual Sensing of Speed Bumps and Potholes Detection

To process each sensor sample, the vehicular sensing app needs to create several temporary objects which gradually fill up memory over time. Therefore, GC_FOR_ALLOC is regularly invoked by the Dalvik VM to clean up these objects. In the Android smartphones, GC_FOR_ALLOC is invoked when there is not enough free memory space to allocate a new object. During this time, the app is blocked at the object allocation command. As a result, vehicular sensing app cannot receive and process the sensor samples from the accelerometers during the blocking time of GC_FOR_ALLOC. Usually, the period of GC_FOR_ALLOC can be more than 100 msec.

As described in the previous section, the detection of speed bumps and potholes depends on the pattern of peak-to-peak average acceleration measurement. If vehicles pass speed bumps or potholes during the period that GC_FOR_ALLOC executes, the pattern of peak-to-peak average acceleration may not be correctly recognized. In Figure 1(a), blocking period of *100* msec means more than *14* % of T_P. Especially, when these periods overlap with one of the highest or lowest value in Figure 1(b), potholes may not be detected. In this example, vehicle speed is below *5* m/sec which is relatively low speed, however, if vehicle speed is larger, the period that vehicle crosses the pothole is much smaller. As a result, *100* msec of blocking time may cover a much bigger ratio of pothole period, the problem of detection is even severer.

3. App-Driven Garbage Collection

In this section, we start by describing the difference between GC_FOR_ALLOC and GC_EXPLICIT. Then, we describe two approaches to proactively GC_EXPLICIT call: free heap memory ratio and periodic calling. As stated in the previous section, GC_FOR_ALLOC is invoked automatically when there is not enough memory to allocate an object. GC_EXPLICIT, on the other hand, is called by the app through *System.gc()* command. It runs on a different thread from the app, therefore, the app can still run in parallel with GC_EXPLICIT on smartphones with multi-core processors which are popular in the recent smartphones. As a result, the blocking time of GC_EXPLICIT is smaller than GC_FOR_ALLOC.

To reduce the blocking time, we utilize the GC_EXPLICIT to gather the temporary objects of vehicular sensing app before the memory becomes full and GC_FOR_ALLOC is invoked. We proposed two approaches for utilizing GC_EXPLICIT: a free heap memory ratio approach and periodic calling approach.

 Copyright ⓒ 2016 SERSC

Table 1. Experiment Smartphone Specification

Phone Name	S3	S3 Neo	Note2
Model	SHV-E210L	GT-I9300I	SHV-E250L
Android Version	4.4.4	4.4.4	4.4.2
Processor core	4	4	4
Processor frequency	1.4 GHz	1.4 GHz	1.6 GHz
RAM	2 GB	1.5 GB	2 GB

In free heap memory ratio approach, we schedule a *Timer* object to periodically check the ratio of free heap memory. Free heap memory ratio is calculated by accessing the amount of free heap memory using method *Runtime.freeMemory()* and total memory using method *Runtime.totalMemory()*. If the ratio is below a given threshold *R*, GC_EXPLICIT will be called. If the threshold *R* is set too high, the blocking time is smaller but GC_EXPLICIT will be called more frequently. On the other hand, the low *R* leads to less frequent run of GC_EXPLICIT, but there will be a possibility that GC_FOR_ALLOC is called before GC_EXPLICIT is called. In this case, there is little benefit of using GC_EXPLICIT.

In the periodic calling approach, we schedule GC_EXPLICIT to run repeatedly every *T* period using a *Timer* object regardless of free heap memory ratio. The small *T* value is expected to cause shorter blocking time but GC_EXPLICIT runs more frequent. The larger *T* value may increase the blocking time since there are more objects to be freed but GC_EXPLICIT runs less frequent. In the meantime, GC_FOR_ALLOC may be invoked by Dalvik VM before GC_EXPLICIT. The optimal *R* and *T* values variate depending on smartphone model and are determined by the experiment results in the next section.

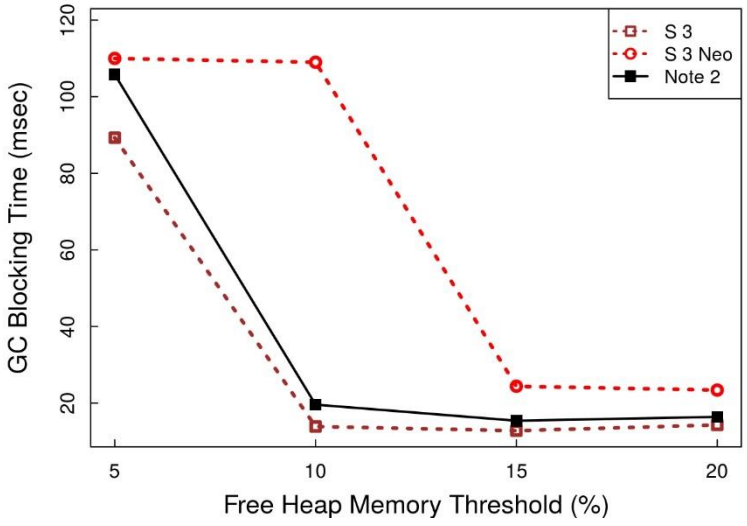
4. Experimental Results

In order to examine the performance of different app-driven garbage collection approaches, we conduct a set of experiments using the vehicular sensing app in Section 2. The set of experiment smartphones includes Samsung Galaxy S3, Samsung Galaxy S3 Duo, and Samsung Galaxy Note 2. The detailed specification is given in Table 1.

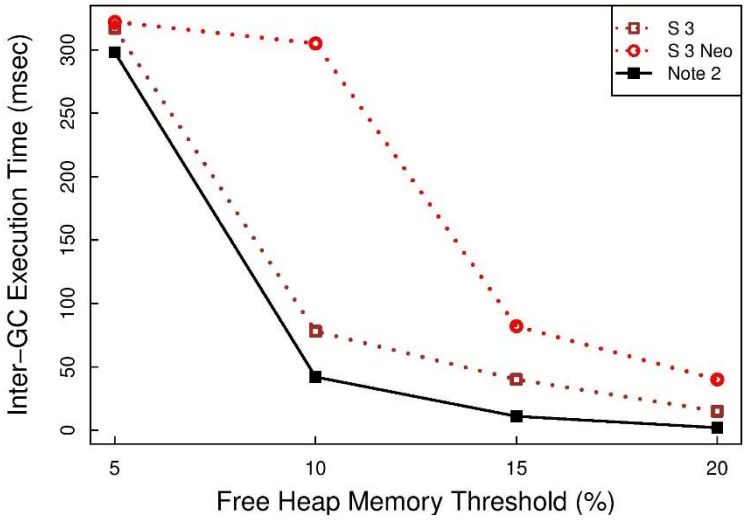
In free heap memory ratio approach, the threshold *R* is set to *5* to *20* %. In periodic calling, the calling period *T* ranges from *60* to *360* sec. In each experment, the app will run for approximately *30* min. We investigate two performance metrics of the approaches in terms of GC blocking time and inter-GC execution period to find the optimal value. GC blocking time is the average blocking time caused by both GC_EXPLICIT and GC_FOR_ALLOC if existed. This metric indicates how good an approach can reduce the impact of GC blocking. On the other hand, inter-GC excecution period shows how frequent the GC runs. A good approach should simultaneously achieve a low GC blocking time and a high inter-GC execution period.

Figure 2 shows the GC block time as well as inter-GC execution period for different free heap memory threshold *R*. We can see from Figure 2(a) that the larger *R* value leads to lower GC blocking time. The lowest blocking time is achieved at *20* %. At *R=10 %*, GC blocking time of S3 and Note 2 significantly reduces after this ratio. At these threshold, GC_FOR_ALLOC is replaced completely by GC_EXPLICIT therefore the GC blocking time relatively low and the same at *10*, *15* and *20* %. In case of S3 Neo, this optimal range of threshold is *15* or *20* %.

Copyright ⓒ 2016 SERSC



(a) GC Blocking Time



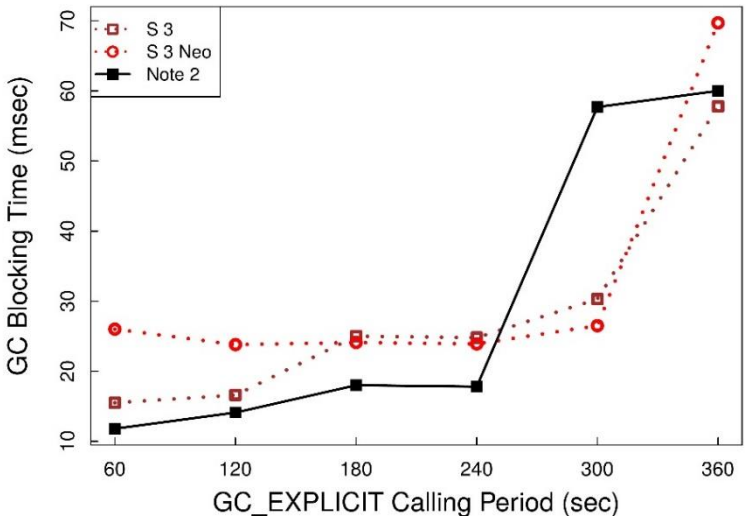
(b) Inter-GC Execution Period

Figure 2. Inter-GC Execution Period and GC Blocking Time of Free Heap Memory Ratio Approach

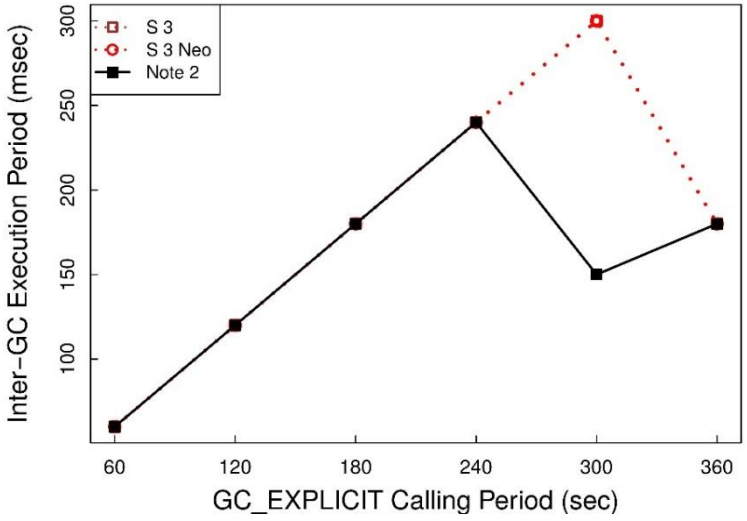
In Figure 2(b), we can see that inter-GC execution period reduces with the increasing of threshold *R*. Among the optimal values in Figure 2(a), *R=10* % achieves the longest inter-GC execution time for S3 and Note 2. Since $R_{opt,1}$*=10* % is the lowest threshold, it takes almost double the time to reach the threshold comparing to *R=15* % and more than *3* times comparing to *R=20* %. In case of S3 Neo, the optimal value is $R_{opt,2}$*=15* %.

Figure 3 shows the GC blocking time as well as inter-GC execution period for different GC_EXPLICIT calling period *T*. We can see from Figure 3(a) that the larger *T* value leads to higher GC blocking time. The lowest blocking time is achieved at *60* sec. At *300* sec, the GC blocking time of S3 and Note 2 significantly increases and keep at high value at *70* msec. This sudden change happens at *240* sec in S3 Neo. After these point, GC_FOR_ALLOC is invoked before GC_EXPLICIT is called therefore the GC blocking time is high.

Copyright ⓒ 2016 SERSC



(a) GC Blocking Time



(b) Inter-GC Execution Period

Figure 3. Inter-GC Execution Period and GC Blocking Time of Periodic GC_EXPLICIT Calling Approach

In Figure 3(b), we can see that inter-GC execution period is the same with GC_EXPLICIT calling period until *300* sec for S3 and S Neo and *240* sec for Note 2. After that, inter-GC execution period is half of GC_EXPLICIT calling period at higher period since GC_FOR_ALLOC runs in between GC_EXPLICIT. At these points, the maximum inter-GC execution period is achieved. From these results, we see that the optimal GC_EXPLICIT calling period $T_{opt,1}$ of S3 and S3 Neo is *300* sec and $T_{opt,2}$ of Note 2 is *240* sec, since they outperforms the higher GC_EXPLICIT calling period in both GC blocking time and inter-GC execution period and has a double inter-GC execution period while slightly higher blocking time comparing to lower GC_EXPLICIT calling period.

Next, we compare the performance of the optimal point in two approaches. We can see that the GC blocking time of $R_{opt,1}$ is about *20* msec and of $R_{opt,2}$ is *25* msec is almost similar as the one of $T_{opt,1}$ is about *25* msec and of $T_{opt,2}$ is *20* msec. However, the inter-GC execution time of the periodic approach is at least *4* times larger than free heap memory ratio approach. Therefore, we can conclude that periodic approach performs better and the optimal period value is *300* sec for S3 and S3 Neo and *240* sec for Note 2.

Copyright ⓒ 2016 SERSC

Moreover, the GC blocking time of *25* msec only overlaps *4* % of the pothole period in Figure 2(b).

5. Conclusion

In this paper, we propose the app-driven garbage collection to reduce the blocking time of GC_FOR_ALLOC in the Android smartphone. The solution is based on the built-in feature the Android smartphone called GC_EXPLICIT. We propose two approaches to call GC_EXPLICIT based on free heap memory ratio or periodic calling. We find optimal value in each approaches based on experimental results. The results show that the periodic calling approach outperforms the free heap space ratio approach in terms of inter-GC execution time. Moreover, the optimal value of periodic calling approach reduces *75* % of the GC blocking period and requires infrequent GC execution.

From the Android version 5.0, the improvement of GC has resolved the problem of GC_FOR_ALLOC. However, the majority of current Android smartphones are below version 5.0, therefore, they have not yet benefited from these improvement. As a result, the proposed solution is still a viable method in near future.

Acknowledgments

This research was supported by Global Frontier Program (2011-0031863) and by Basic Science Research Program (2013R1A1A1012290) through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning.

References

[1] P. Mohan, V. N. Padmanabhan and R. Ramiee, "Nericell: Rich Monitoring of Road and Traffic Conditions Using Mobile Smartphones", in ACM SenSys, **(2008)**.
[2] G. Strazdins, A. Mednis, G. Kanonirs, R. Zviedris and L. Selavo, "Towards vehicular sensor networks with android smartphones for road surface monitoring", in 2nd international workshop on networks of cooperating objects, **(2011)**.
[3] A. Mednis, G. Strazdins, R. Zviedris, G. Kanonirs and L. Selavo, "Real time pothole detection using Android smartphones with accelerometers", in DCOSS'11, **(2011)**.
[4] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden and H. Balakrishnan, "The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring", in International Conference on Mobile Systems, Applications, and Services, **(2008)**.
[5] R. Madli, S. Hebbar, P. Pattar and V. Golla, "Automatic Detection and Notification of Potholes and Humps on Roads to Aid Drivers", IEEE Sensors Journal, vol. 15, no. 8, **(2015)**, pp. 4313-4318.
[6] "Android Developer", Google, [Online]. Available: http://developer.android.com/tools/debugging/debugging-memory.html. [Accessed 28 04 2016].
[7] I. Kalkov, D. Franke, J. F. Schommer and S. Kowalewski, "A real-time extension to the Android platform", in ACM JTRES'12, **(2012)**.
[8] T. Gerlitz, I. Kalkov, J. F. Schommer, D. Franke and S. Kowalewski, "Non-blocking garbage collection for real-time android", in ACM JTRES'13, **(2013)**.
[9] G. V. S. B. S. Murthy, "Detection of potholes in autonomous vehicle", IET Intelligent Transport Systems, vol. 8, no. 6, **(2013)**, pp. 543-549.

Authors

Nguyen Hoa Hung, He received the B.Eng degree in computer science and engineering from Ho Chi Minh city University of Technology, Ho Chi Minh city, Vietnam, in 2009 and M.S. degree in computer science from Pusan National University, Busan, Korea, in 2013. He is currently pursuing the Ph.D. degree in electrical, electronic and computer at Pusan National University.

 Copyright ⓒ 2016 SERSC

His research interests include scalable beacon dissemination protocol over unreliable channel in vehicular networks and intelligent driver assistance systems.

Rachmad Nafisholeh, He received the B.S. degree in Informatics Engineering from Sepuluh Nopember Insitute of Technology, Indonesia.

Currently, he is a master student in Pusan National University, Busan, South Korea. His research interests include smartphone-based vehicular sensing, with a particular focus on vehicle mobility and road surface sensing.

Han-You Jeong, He received the B.S., M.S., and Ph.D. degrees in the Department of Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea, in 1998, 2000, and 2005, respectively. From 2005 to 2007, he was a senior engineer with the Telecommunication R&D Center, Samsung Electronics, Suwon, Korea. In 2008, he joined the Digital Technology Center, University of Minnesota, Minneapolis, as a postdoctoral research fellow.

He is currently an associate professor with the Department of Electrical and Computer Engineering, Pusan National University, Busan, Korea. His research interests include wireless networks, vehicular networks, and optical networks.

Copyright ⓒ 2016 SERSC

Copyright ⓒ 2016 SERSC