Hindawi
Mobile Information Systems
Volume 2017, Article ID 1856329, 9 pages
https://doi.org/10.1155/2017/1856329

Research Article

Offloading Method for Efficient Use of Local Computational Resources in Mobile Location-Based Services Using Clouds

Yunsik Son[1] and Yangsun Lee[2]

[1]*Department of Computer Science and Engineering, Dongguk University, 3-26 Pil-dong, Jung-gu, Seoul 100-715, Republic of Korea*
[2]*Department of Computer Engineering, Seokyeong University, 16-1 Jungneung-dong, Sungbuk-ku, Seoul 136-704, Republic of Korea*

Correspondence should be addressed to Yangsun Lee; yslee@skuniv.ac.kr

Received 9 December 2016; Revised 6 February 2017; Accepted 20 February 2017; Published 14 March 2017

Academic Editor: Subramaniam Ganesan

With the development of mobile computing, location-based services (LBSs) have been developed to provide services based on location information through communication networks or the global positioning system. In recent years, LBSs have evolved into smart LBSs, which provide many services using only location information. These include basic services such as traffic, logistic, and entertainment services. However, a smart LBS may require relatively complicated operations, which may not be effectively performed by the mobile computing system. To overcome this problem, a computation offloading technique can be used to perform certain tasks on mobile devices in cloud and fog environments. Furthermore, mobile platforms exist that provide smart LBSs. The smart cross-platform is a solution based on a virtual machine (VM) that enables compatibility of content in various mobile and smart device environments. However, owing to the nature of the VM-based execution method, the execution performance is degraded compared to that of the native execution method. In this paper, we introduce a computation offloading technique that utilizes fog computing to improve the performance of VMs running on mobile devices. We applied the proposed method to smart devices with a smart VM (SVM) and HTML5 SVM to compare their performances.

1. Introduction

Advancements in mobile technology and location-based services (LBSs) have helped improve the quality of life of users and have fostered many business opportunities [1]. With the increasing use of LBSs, their value has likewise increased, and this value extends to the services to which the LBSs promote access.

In its early stages, the LBS used only simple location information. It has recently developed into an intelligent system that employs multiple types of information, such as the user's location, time, personal information, and behaviors [2]. It is thus difficult to effectively and efficiently perform services on a mobile device without considerable computing power. To solve this problem, a computation offloading technique can perform certain tasks in an alternative environment, such as a cloud or fog, instead of executing in the mobile device.

In addition, mobile services have the disadvantage of running various platform-dependent applications developed in different languages, such as C C++, Java, and Objective C. The smart cross-platform is a program that enables applications developed with C/C++, Java, and Objective C to run on various mobile devices, smart devices, and browsers that support HTML5 [3]. Nevertheless, owing to the characteristics of the virtual machine (VM), the performance of the hardware platform or browser on which the VM operates is greatly impacted by this approach, even if optimization is performed at the interpreter and code level. When running VM applications on a low-performance hardware platform—depending on the content complexity—it is difficult to ensure the quality of service (QoS) in terms of execution.

In this study, we strived to solve the above problems by using fog computing and a smart VM (SVM) platform to effectively and efficiently provide LBSs on mobile and smart devices. Unlike use of a centralized cloud, in the proposed approach, a local unit is employed to enable smart LBSs to effectively operate on a variety of platforms.

The remainder of this paper is organized as follows. In Section 2, we examine the features of an existing offloading scheme, the smart cross-platform, and the SVM. We analyze

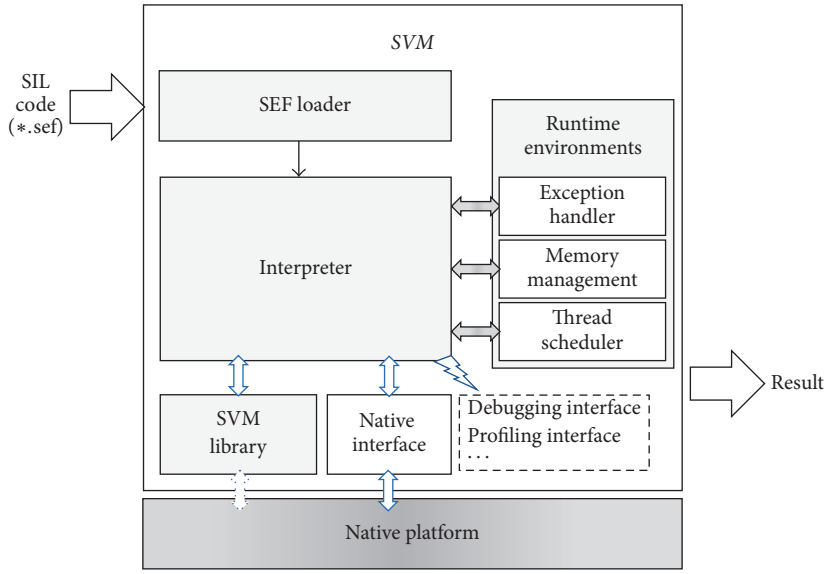


Figure 1: System configuration of the smart virtual machine.

the limitations of these respective techniques. In Section 3, we describe the proposed offloading scheme for SVMs. In Section 4, the performance of the proposed method is verified through experiments. Finally, Section 5 concludes the paper.

2. Related Works

2.1. Smart Virtual Machine Model. The SVM is a stack-based VM solution that is loaded on smart devices. It enables dynamic application programs to be downloaded and run independently of the platform. The SVM is designed to employ the Smart Intermediate Language (SIL), which can accommodate both procedural and object-oriented languages. It thus can accommodate multiple languages, such as C/C++ and Java, as well as the Objective C language used in iOS. These languages are now widely used by developers [3, 4].

The SVM system consists of three parts: a compiler that compiles application programs to create a Smart Assembly Format (SAF) file from SIL code, an assembler that converts the SAF file into a Smart Executable Format (SEF) file, and a VM that receives the SEF file and runs the program. The SVM configuration is shown in Figure 1.

2.2. HTML5 Smart Virtual Machine. HTML5 SVM is a VM-based solution that provides an integrated environment in development and execution phases by supporting both a web-based execution environment and multiple programming languages [4]. Because separate development and execution environments exist in smart devices, separate development work must be performed based on the target device and respective platform to provide a specific type of content to multiple smart device types.

HTML5 SVM can provide various contents on heterogeneous target devices with web browsers. However, it requires adequate device performance (e.g., many frames per second) to enable the smooth production of results. Thus, the performance and QoS of the given contents depend on the target device's computing power.

Figure 2 shows the system configuration of the HTML5 SVM. It basically has two layers: SVM core and SVM adaptation layer. The SVM core performs content execution. It is comprised of four components: the SEF loader, interpreter, runtime environment, and built-in library. The second layer is the SVM adaptation layer, which has six components for interfacing with target HTML5-based web browsers.

2.3. Computational Offloading. Mobile devices provide services in an approach that differs from that of traditional personal computers (PCs) because the mobile device computing power—processing speed, memory, storage space, and battery life—is limited. In particular, the LBS requires considerable processing and battery power because it provides services based on location information collected using the global positioning system (GPS) or WiFi. Moreover, in recent years, use of LBS has increasingly required more complex computations and energy to expand its services to collect and provide more complex information.

Computational offloading is a cloud computing technique that is used to run programs and provide content when there is a computing-powered-restricted environment [5–8]. Complex tasks require higher computing power. If the target device has insufficient computing power, the QoS of the provided contents/programs decreases. In this case, the offloading technique is a possible solution. Accordingly, the target device delegates complex tasks to a cloud server instead of directly executing them [9]. Figure 3 shows the proposed offloading concept.

Methods for selecting computation offloading operations are largely classified into static and dynamic approaches.

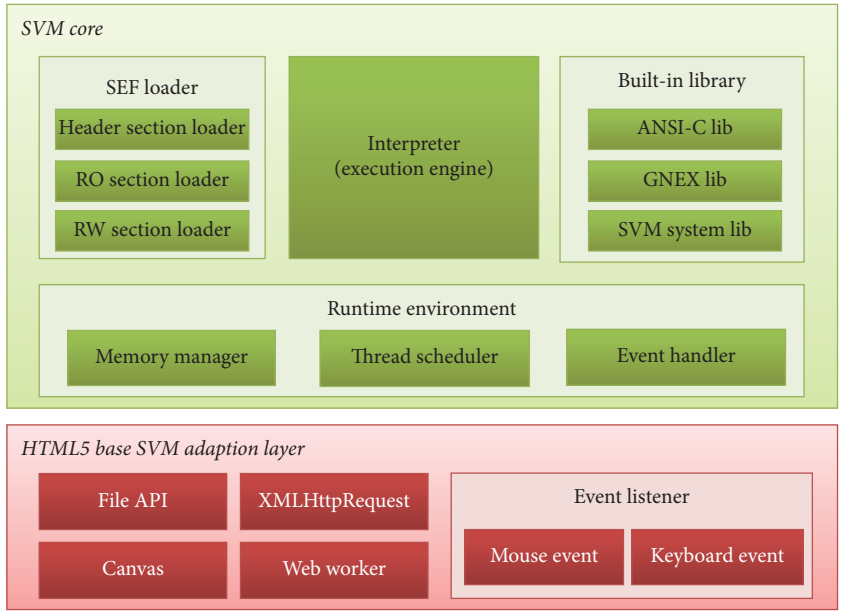


Figure 2: System configuration of the HTML5 smart virtual machine.

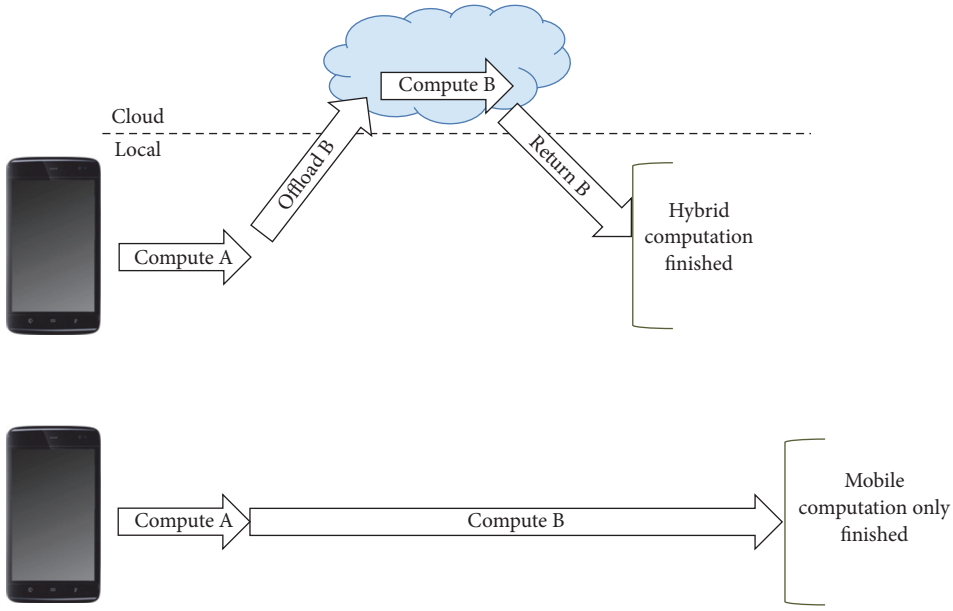


Figure 3: Computational offloading concept model.

La and Kim [10] classified offloading techniques as shown in Figure 4.

The static technique reduces the execution load by selecting the part to be offloaded during program development. The static method has the advantage of a low load in terms of cost analysis at runtime. However, the cost analysis is possible only by using predictable variables [11]. Meanwhile, the dynamic method selects the part to be offloaded with consideration of the fluctuation factors, such as the network state and remaining battery power, during execution. The dynamic method can accurately reflect the current state of the mobile device. Nevertheless, it is difficult to design a model that reflects all variables, and the required workload for the cost analysis is significant [12, 13].

Partial offloading is a method of submitting some of the work to the cloud. When a specific task is frequently used and cannot be performed in parallel, the communication costs and waiting times are increased. The full offloading method, on the other hand, addresses only the interaction with the user on the mobile device; it defers the execution to the cloud. When frequent interaction with a user occurs, synchronization problems likewise occur. Therefore, it is necessary to

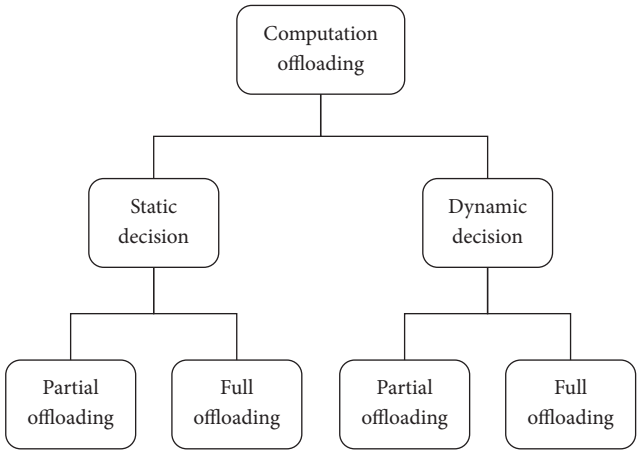


Figure 4: Classified computation offloading methods.

selectively assign an operation that is suitable for offloading to the cloud.

In the proposed approach, it was difficult to reflect real-time fluctuating factors, such as mobility and communication scenarios. Therefore, the offloading decision is based on the Mobile Augmentation Cloud Service (MACS) model [14], which estimates the code transmission cost, memory, and CPU usage for each function using the profiler at the content compile time. The offloading object determines the unit of work based on the function.

3. Offloading Module for Smart Location-Based Services

The overall system architecture for supporting smart LBs is shown in Figure 5. Each mobile device can use an appropriate fog based on its location when offloading is needed to perform the service. We designed and implemented an SVM offloading module. Offloading can improve the performance of utilizing the computing power of the cloud but may result in overhead; thus if the offloading gain is greater than the overhead cost without offloading a given task unconditionally this should be done selectively.

The offloader proposed in this paper automatically determines to offload by statically offloading through profiling of source code in the compilation step and automatically generates offloading code and transfers selected function unit work to the fog based on the location.

In the proposed approach, fog computing is employed. It is a localized service of cloud computing. It processes large amounts of data at the point of data origin, rather than at a remote server, such as centralized cloud server. Moreover, it operates on the basis of location information, similar to LBSs. Therefore, services can be provided more effectively by using fog computing when offloading complex operations in LBSs. The overhead value includes the transmission time of the data over the network, as well as the serialization/deserialization time of data transmitted locally and from the server. Analysis of the overhead is very important because the offloading performance based on the overhead can be lower than when the local operation is performed.

Figure 6 shows the offloading module structure of the SVM proposed in this paper. First, the SVM is divided into the mobile device and cloud server. Except for the thread scheduler in the runtime environment and the adaptation layer required by the actual host mobile platform, the two VMs are equivalent. In terms of content, the local SVM directly loads and executes the downloaded content. Meanwhile, the cloud SVM loads the same content as the user-executed content from the SVM application database in the cloud. This cloud refers to both the existing centralized cloud and the fog environment.

The SVM on the server has the same configuration and operation method for both cloud and fog environments. By performing the loading equivalent of VMs and content on the mobile device and the cloud, it is easy to offload the functions defined by the profiler without requiring additional work or implementing a server interface.

The process of performing the function unit offloading is as follows. First, the local SVM delegates the function unit job to the server when loading the content and calling the function (designated as offloading) during the command execution. In this process, the required context information for executing the corresponding function is extracted, serialized, and transmitted to the server. The context data refer to general data, such as the program counter, command information, and stack information used during content operation in the interpreter (the driving engine of the SVM).

The offloading module synchronizes the context between the device and server by sending the context data of the SVM to the server. The server parses the corresponding data to extract the context information required for executing the function. It performs the task of the requested function on the cloud SVM through context switching. When the function unit is finished, the changed context information is serialized and transferred to the local SVM, which reflects the changed context information to its own context to ensure that the state aligns with the result of directly executing the function.

4. Experimental Results

We applied the proposed offloading technique to SVM for a smart device and an HTML5 SVM to verify the improvement of the content execution speed. To measure the overhead caused by offloading, the local SVM was used with RaspberryPi B+, which has a Quad-Core ARM Cortex-A7 900 Mhz processor, 512 MB of memory, and the Raspbian operating system.

Figure 7 depicts the comparison of execution times before and after application of offloading through the SVM in various mobile devices and web browsers. The SVM for the smart devices was tested on an iPad2 and a Galaxy Tab 10.1. The algorithms used were the prime number, n-queen, and perfect number. HTML5 SVM was tested on a PC, iPad2, and Galaxy Tab 10.1. The prime number and n-queen algorithms were used.

The experimental results showed that the algorithm performance improved by offloading as a whole, as shown in

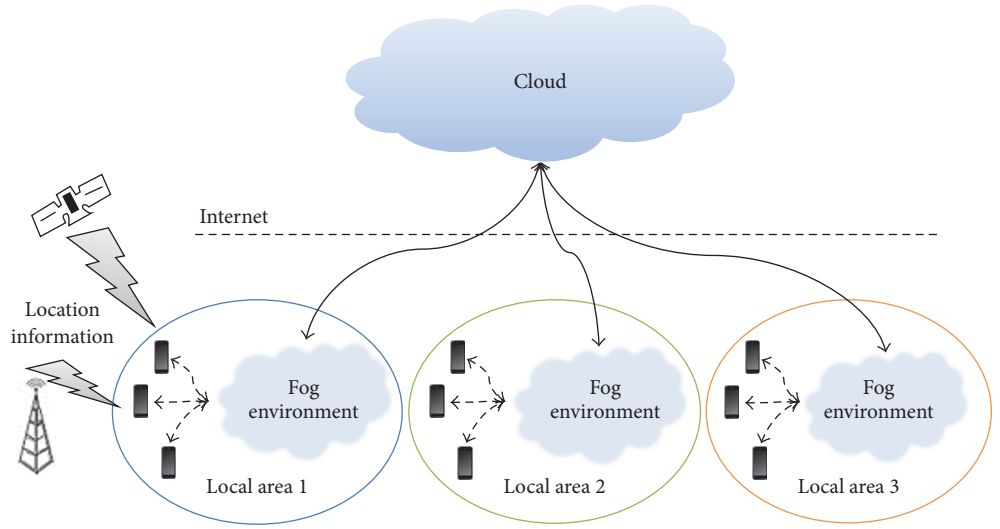


Figure 5: The system architecture to support smart LBS.

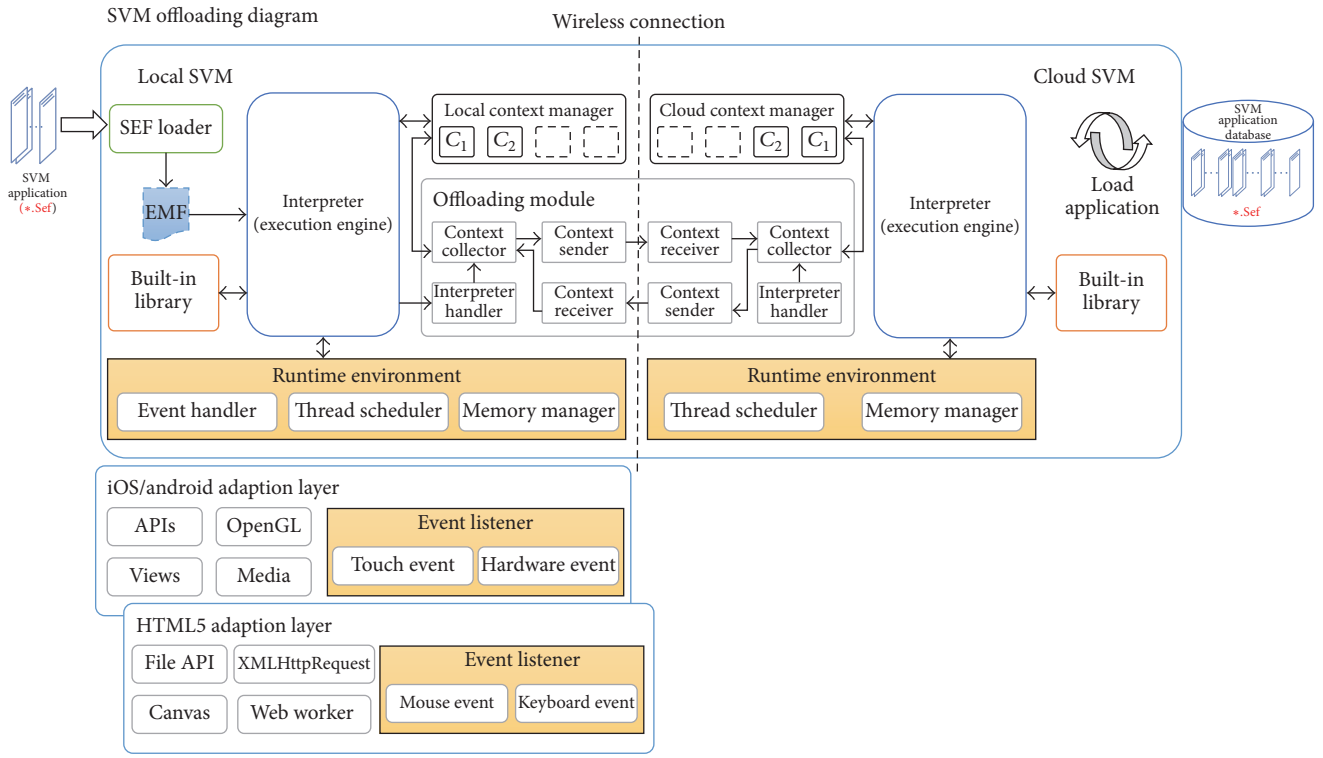


Figure 6: Proposed computational offloading modules for smart location-based service.

Figure 8. The execution time of the SVM with offloading was higher than those of the algorithm with a lower complexity and the algorithm with a higher complexity. The execution time of the SVM before offloading increased as the complexity of the algorithm increased. No significant difference was evident in terms of processing time. Therefore, the higher the algorithm complexity was, the greater the performance improvement rate was before and after offloading, as shown in Figures 7 and 8.

However, in some experiments, the offloading results showed a performance deterioration due to offloading overhead, which was incurred during the algorithm execution in the HTML5 SVM, as shown in Figure 9. The offloading overhead could be generally divided into types, such as context serialization and parsing time on the client, context serialization and parsing time on the server, transferring context data and clients, additional time incurred due to context switching, and function loading time at the server.

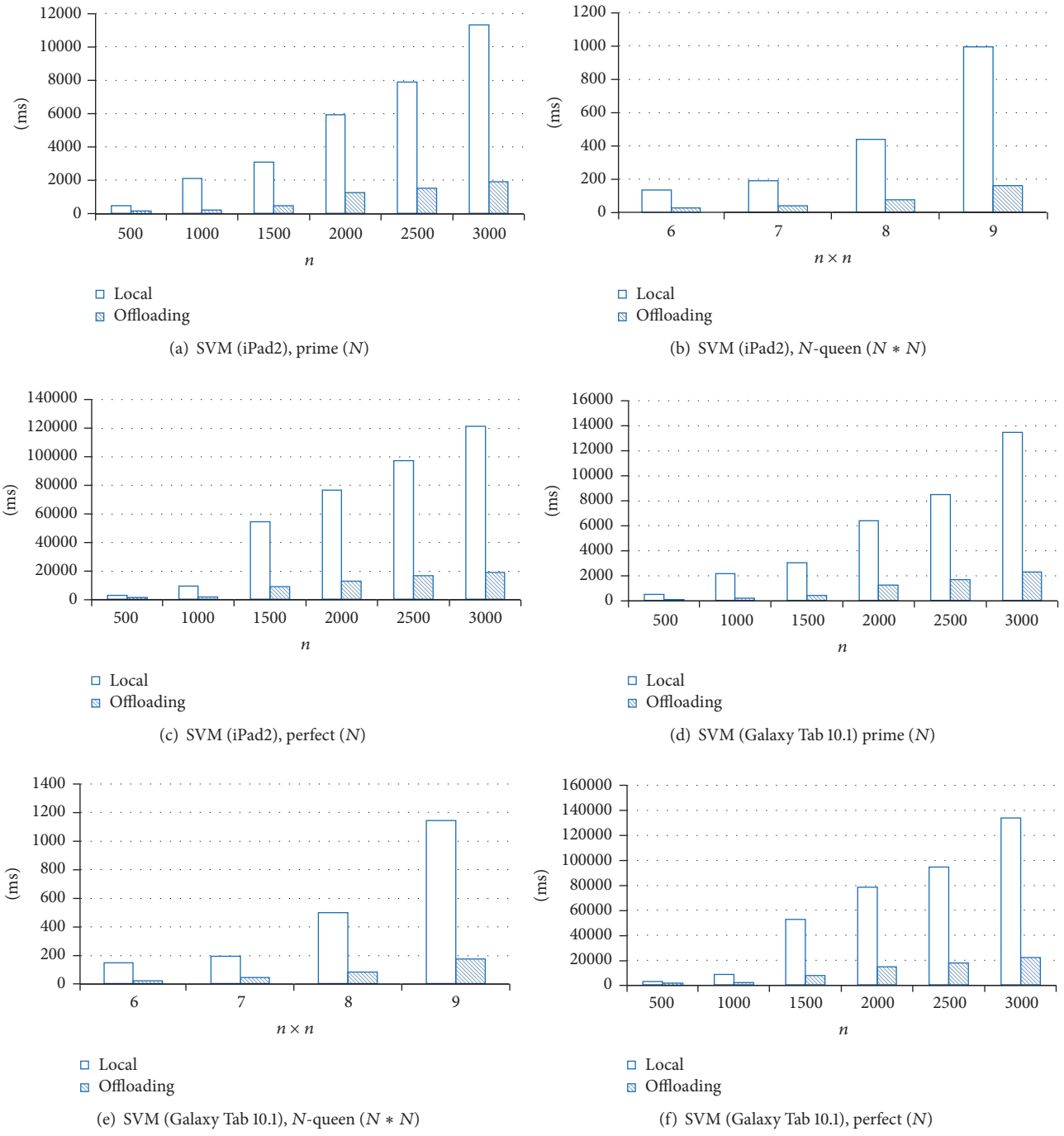
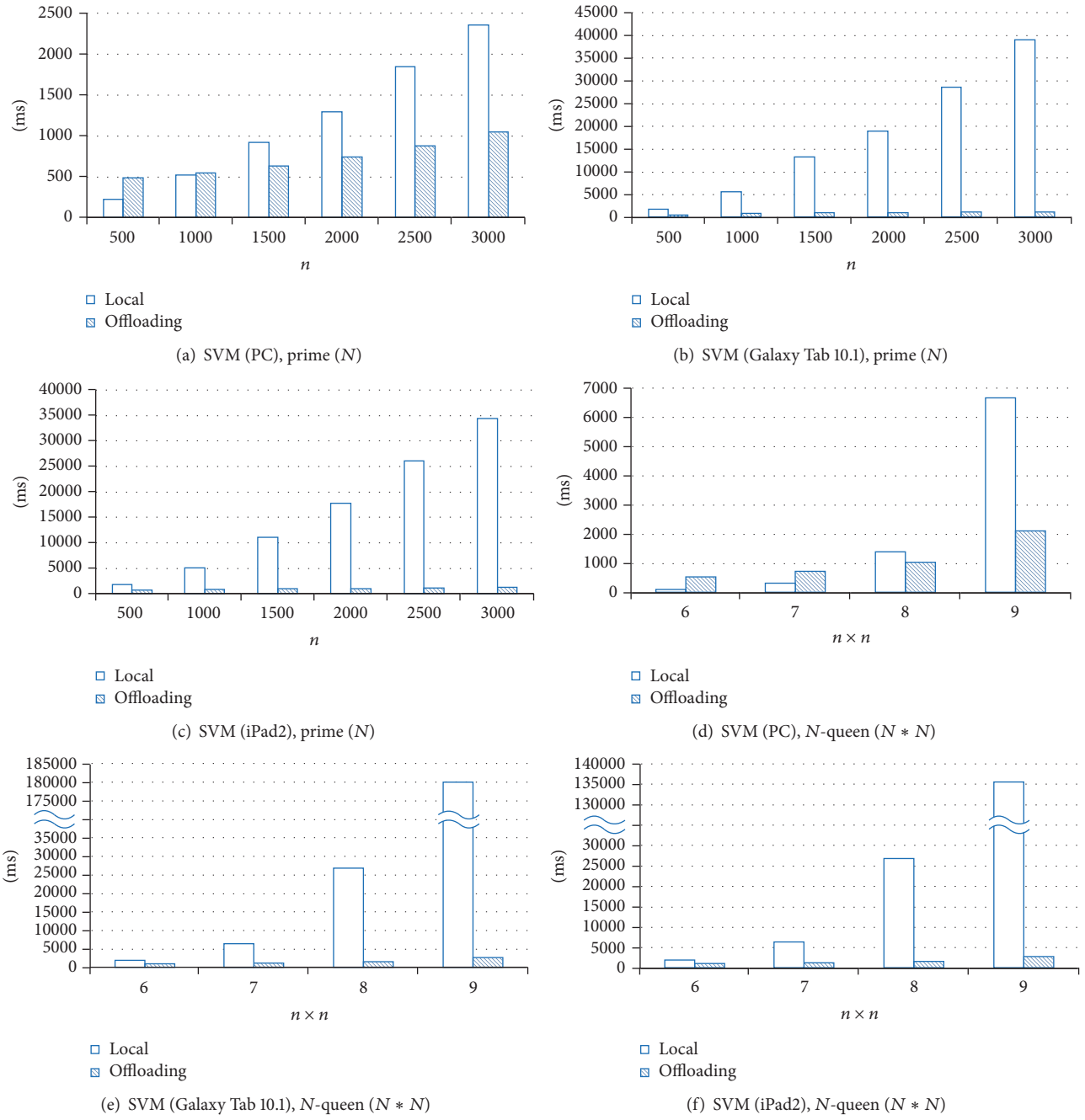


Figure 7: Comparisons of the performance evaluation using offloading in SVM on smart devices.

As a result of running the algorithm with offloading in each device, the serialization and parsing time of the server were constant. Meanwhile, the serialization and parsing time of the client differed depending on the device performance. In addition, since the additional overhead in the context switch and function loading during the operation of the client SVM depended on the device performance, it was confirmed that the time varied depending on the device.

5. Conclusions

LBSs have recently become highly available and valuable, and they now involve a variety of applications. However, with their increasing sophistication, more complicated operations are required, which causes issues on the mobile device—the main execution environment of the LBS. A complex computation load requires high computing power; thus, it is difficult



(a) SVM (PC), prime (N)

(b) SVM (Galaxy Tab 10.1), prime (N)

(c) SVM (iPad2), prime (N)

(d) SVM (PC), N-queen ($N * N$)

(e) SVM (Galaxy Tab 10.1), N-queen ($N * N$)

(f) SVM (iPad2), N-queen ($N * N$)

Figure 8: Performance evaluation of the offloading in HTML5 SVM for same algorithms with different target devices.

to perform tasks that require high computations on devices with low computing power. Offloading addresses this issue.

With offloading, the device delegates to the server a task with a high computational complexity. It sends to the server the data required for the task or the running-program context information. Because the server receives the job execution result and the changed context information, the resource consumption of the computing job can be drastically reduced based on the job characteristics. Such offloading can overcome low performance by providing high computing power of the server. Nevertheless, additional overhead is incurred on account of the communication cost of transmitting and receiving data between the device and server. The average communication cost in this study was 8 ms for the prime number algorithm (N), 7 ms for the n-queen algorithm ($N \times N$), and 10 ms for the perfect algorithm (N). Therefore, the problem could be mitigated by leveraging fog computing, which can perform intermediate processing for each region using location information.

Through the offloading technique presented in this paper, the SVM provides high computing performance from the server. It can thus perform tasks that require high computing

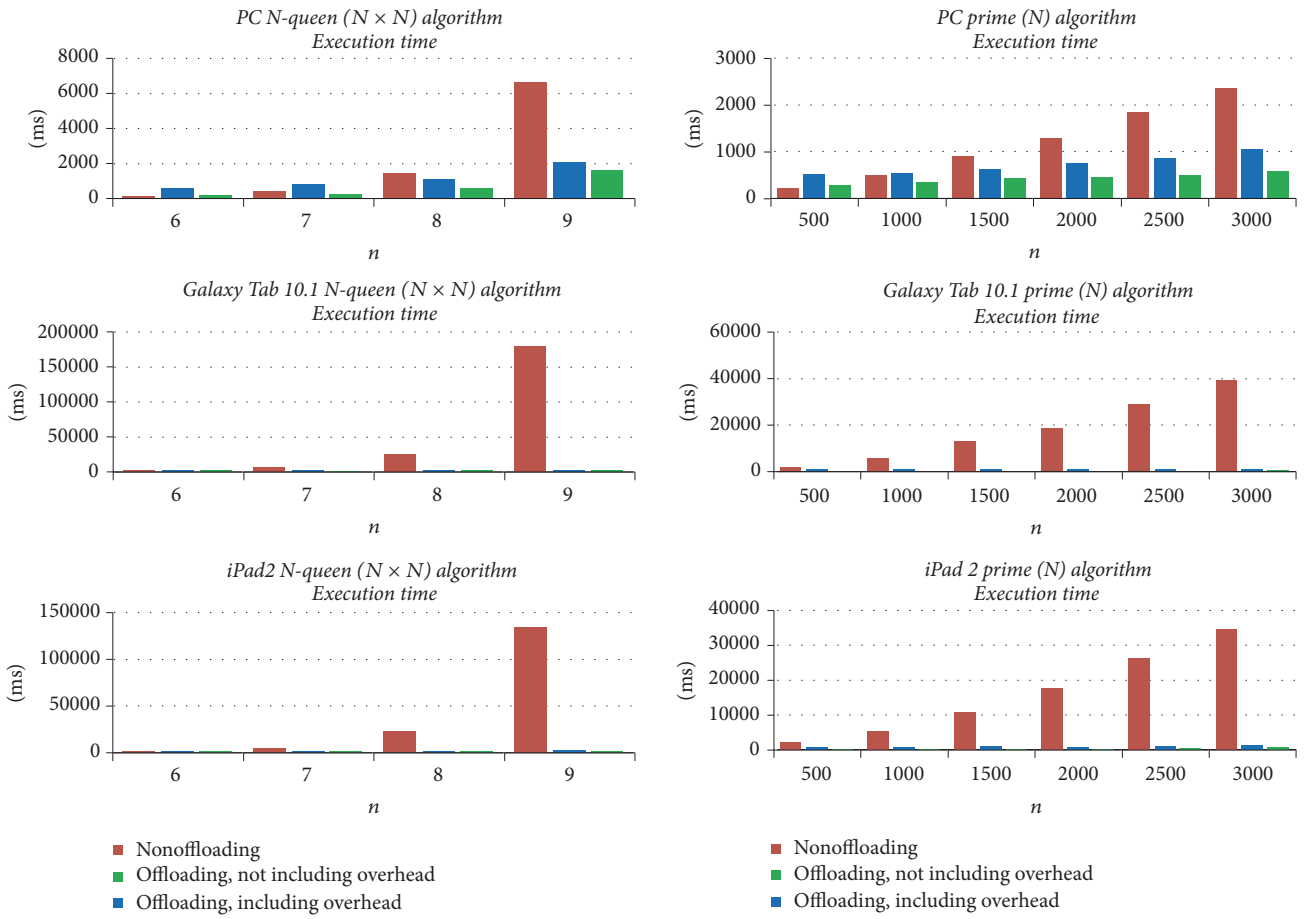


Figure 9: Offloading overhead on HTML5 SVM.

operations, even on low-performance platforms. Offloading includes context serialization and parsing time, server serialization and parsing time, and data transmitting and receiving time. Thus, because additional overhead, such as server drive time, is incurred, the offloading should be performed only when the execution time of the job to be offloaded is larger than the overhead incurred in offloading.

The SVM offloading module currently under study has a structure for delegating a task to a server through offloading during a specific function call. This call is made using precalculated profile information while the content is running. Even though indiscreet offloading is a simple task with less time than overhead, its application to offloading can be used for lower performance than the existing one. To solve this problem, the overhead caused by the offloading module of the SVM is minimized through a decision model that determines in advance the efficiency of offloading by calculating the performance difference before and after applying the offloading application. We intend to perform research to improve the performance accordingly.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (no. 2016R1A2B4008392).

References

[1] E. Kaasinen, "User needs for location-aware mobile services," *Personal and Ubiquitous Computing*, vol. 7, no. 1, pp. 70–79, 2003.

[2] A. Pingley, Y. Wei, Z. Nan, F. Xinwen, and Z. Wei, "CAP: A context-aware privacy protection system for location-based services," in *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS '09)*, pp. 49–57, Montreal, Canada, June 2009.

[3] Y. S. Lee and Y. S. Son, "A study on the smart virtual machine for executing virtual machine codes on smart platforms," *International Journal of Smart Home*, vol. 6, no. 4, pp. 93–106, 2012.

[4] Y. Son, S. Oh, and Y. Lee, "Design and implementation of HTML5 based SVM for integrating runtime of smart devices and web environments," *International Journal of Smart Home*, vol. 8, no. 3, pp. 223–234, 2014.

[5] K. Yang, S. Ou, and H.-H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier

mobile internet applications," *IEEE Communications Magazine*, vol. 46, no. 1, pp. 56–63, 2008.

[6] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[7] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "COSMOS: computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '14)*, pp. 287–296, Philadelphia, PA, USA, August 2014.

[8] B. G. Chun, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the 6th ACM Conference on Computer Systems*, pp. 301–314, Salzburg, Austria, April 2011.

[9] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[10] H. J. La and S. D. Kim, "A taxonomy of offloading in mobile cloud computing," in *Proceedings of the 7th IEEE International Conference on Service-Oriented Computing and Applications (SOCA '14)*, pp. 147–153, Matsue, Japan, November 2014.

[11] C. Wang and Z. Li, "A computation offloading scheme on handheld devices," *Journal of Parallel and Distributed Computing*, vol. 64, no. 6, pp. 740–746, 2004.

[12] H.-Y. Chen, Y.-H. Lin, and C.-M. Cheng, "COCA: computation offload to clouds using AOP," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '12)*, pp. 466–473, IEEE, Ottawa, Canada, May 2012.

[13] T.-Y. Lin, T.-A. Lin, C.-H. Hsu, and C.-T. King, "Context-aware decision engine for mobile cloud offloading," in *Proceedings of the IEEE Wireless Communications and Networking Conference Workshops (WCNCW '13)*, pp. 111–116, Shanghai, China, April 2013.

[14] D. Kovachev, T. Yu, and R. Klamma, "Computation offloading from mobile devices into the cloud," in *Proceedings of the IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pp. 784–791, 2012.