International Journal of Grid and Distributed Computing
Vol. 10, No. 12 (2017), pp.149-160
http://dx.doi.org/10.14257/ijgdc.2017.10.12.14

A Study on Context Synchronization Method for Offloading on IoT-Cloud Fusion Virtual Machine

JaeHyun Kim[1] and YangSun Lee[1*]

[1]*Dept. of Computer Engineering, Seokyeong University*
16-1 Jungneung-Dong, Sungbuk-Ku, Seoul 136-704, Korea
**yslee@skuniv.ac.kr*

Abstract

The IoT-Cloud fusion virtual machine is a virtual machine for IoT appliances that employs offloading techniques that delegate tasks requiring high computing power to low-performance appliances to a high-performance cloud server environment. IoT devices embedded with IoT-Cloud fusion virtual machines can perform complex tasks using the computing power of high-performance servers. The virtual machine with the existing offloading mechanism sends and receives all the context information to maintain program consistency in the communication between the local environment and the cloud server environment. Most of the overhead costs incurred in offloading are proportional to the size of the context information transmitted over the network. Therefore, the existing context information synchronization structure sends context information that is not necessary for the task to be performed when offloading, thereby increasing the overhead cost incurred in the process of transmitting the context information in low-performance IoT devices.

In this paper, in order to solve this problem, we classify the cases according to the context information required for the offloading function to send and receive only the context information necessary for the task execution without sending or receiving all the context information when offloading. Also, we have studied contextual information synchronization technique to send and receive minimum context information by synchronizing context information according to classified cases. The IoT-Cloud fusion virtual machine with improved contextual information synchronization scheme reduces the size of context information to be transmitted, thereby reducing the overhead cost of communication in low-performance IoT devices, thereby improving the offloading efficiency.

Keywords: *IoT-Cloud Fusion Virtual Machine, Offloading, Context Information, IoT Devices, Contextual Information Synchronization Scheme*

1. Introduction

The IoT-Cloud fusion virtual machine can perform tasks that require high computing power by delegating tasks that are difficult to perform in the development environment of low-performance IoT devices to the cloud server environment by applying the offloading technique. In order to maintain the consistency of programs during the communication between the cloud environment and the local environment, the context information must be transmitted and synchronized. Therefore, the virtual machine to which the conventional offloading technique is applied is a structure to send and receive all context information.

Received (October 20, 2017), Review Result (December 1, 2017), Accepted (December 5, 2017)
[*] Corresponding Author

Since most of the overhead costs that occur during offloading are proportional to the size of the context information transmitted to the network, this structure also collects context information that is not necessary for performing the task, and the overhead cost of sending context information from the IoT devices has increased.

In this paper, we propose a method to reduce the resource consumption rate of context information transmission from low performance IoT equipment to cloud server, to distinguish necessary context information for offloading function to efficiently perform the offloading. the contextual information synchronization technique is also studied. The virtual machine using the improved context information synchronization technique reduces the overhead caused by the context information synchronization process of the low performance IoT device, thereby increasing the offloading efficiency.

2. Related Studies

2.1. The IoT-Cloud Fusion Virtual Machine

The IoT-Cloud fusion virtual machine is a virtual machine for building a platform-independent environment in low-performance IoT device. It is a virtual machine that lightens the existing smart virtual machine [1] to low performance IoT devices. Therefore, it is a virtual machine that can accept contents written in C / C ++, Java, and JavaScript languages and has the advantage of a smart virtual machine that programmers can write programs regardless of development language restrictions [2-4]. In addition, the IoT-Cloud fusion virtual machine can perform by delegating high-complexity tasks to a high-performance cloud server environment by applying the offloading technique in IoT equipment. Figure 1 shows the structure of the IoT-Cloud fusion virtual machine.

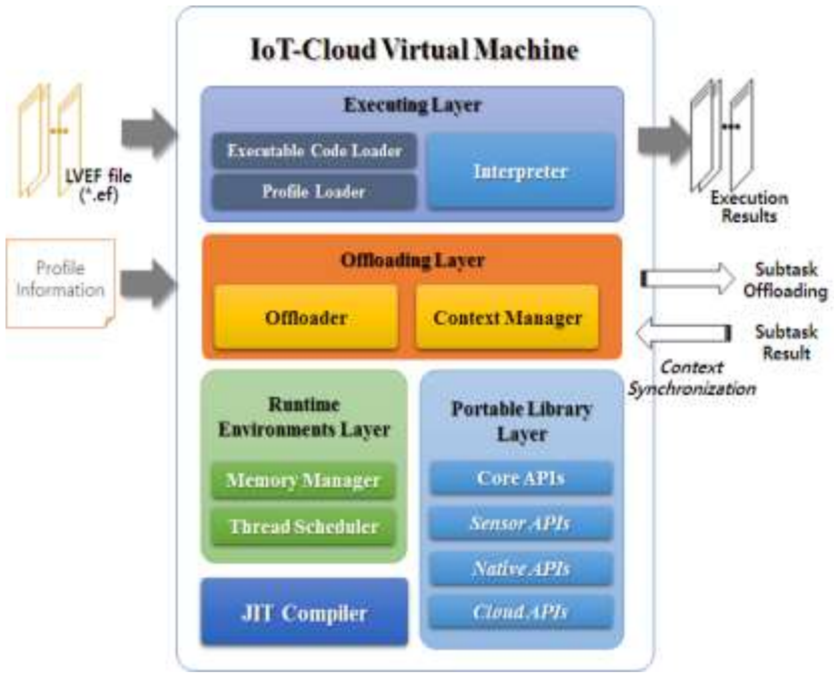


Figure 1. Structure of the IoT-Cloud Fusion Virtual Machine

The IoT-Cloud fusion virtual machine consists of five components: an execution layer, an offloading layer, an execution environment layer, a portable library layer, and a JIT compiler. An execution layer is a layer related to the execution of a program. It is composed of an executable code loader, a profile loader, and an interpreter. The offloading layer is a layer for communication between the IoT equipment and the cloud server, and consists of an offloader and a context manager. When an offloading request occurs in the interpreter, the offloader collects the context information of the execution environment and sends it to the server, and receives the result of the task processed by the server. The portable library consists of APIs used in virtual machines, core APIs essential for program operation, various sensors and native APIs for IoT devices, and APIs for cloud API calls. The JIT compiler takes the intermediate code for the virtual machine as input and generates the native code as output.

2.2. Offloading

Offloading is a cloud computing technique that is used to maximize the efficiency of resource consumption in low-performance equipment development environments such as IoT devices [5-8]. Offloading delegates high-complexity tasks to a high-performance cloud server environment to perform in a local environment, thereby reducing resource consumption due to computing tasks in low-performance IoT devices [9-10].

Offloading is advantageous in that it can overcome low performance by providing high computing power of server, but overhead cost is incurred in network communication between local environment and server environment. If the overhead cost incurred is larger than the overhead cost that occurs in the case of working in the local environment without offloading, offloading performance degrades rather. Therefore, by analyzing the execution load information for each program element through profiling, the data to be offloaded should transmit and receive data with improved performance in offloading. In addition, since the overhead cost incurred in network communication is proportional to the size of data to be transmitted and received, a method of minimizing the size of data to be transmitted is required for efficient offloading.

2.3. Context Information

The context information is the information needed by the interpreter of the IoT-Cloud fusion virtual machine to execute the program. The context information required for the interpreter to execute the program is largely composed of essential context information necessary for performing all functions, and stack context information for managing command execution and variables. Figure 2 shows the composition of the essential contextual information.

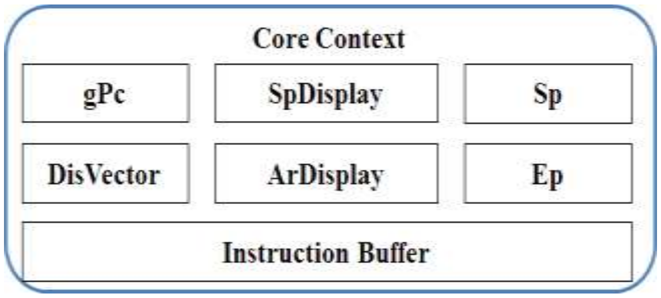


Figure 2. Essential Contextual Information

Essential context information is context information that all functions need, including instruction buffer, gPc for accessing the instruction buffer, sp for accessing the operation stack, ep for accessing the activation record, SpDisplay for manages the frame position information for starting the function in the operation stack, and ArDisplay which manages frame position information where function starts in activation record.

The stack context information consists of a stack of operations used when executing stacking or arithmetic instructions, an activation record to manage local variables, and a constant pool to manage literals and global variables. Figure 3 shows the composition of the stack context information.

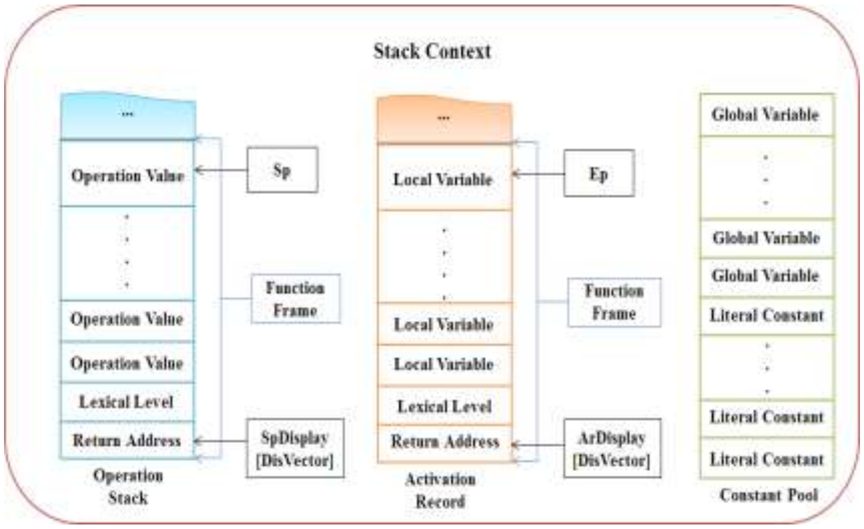


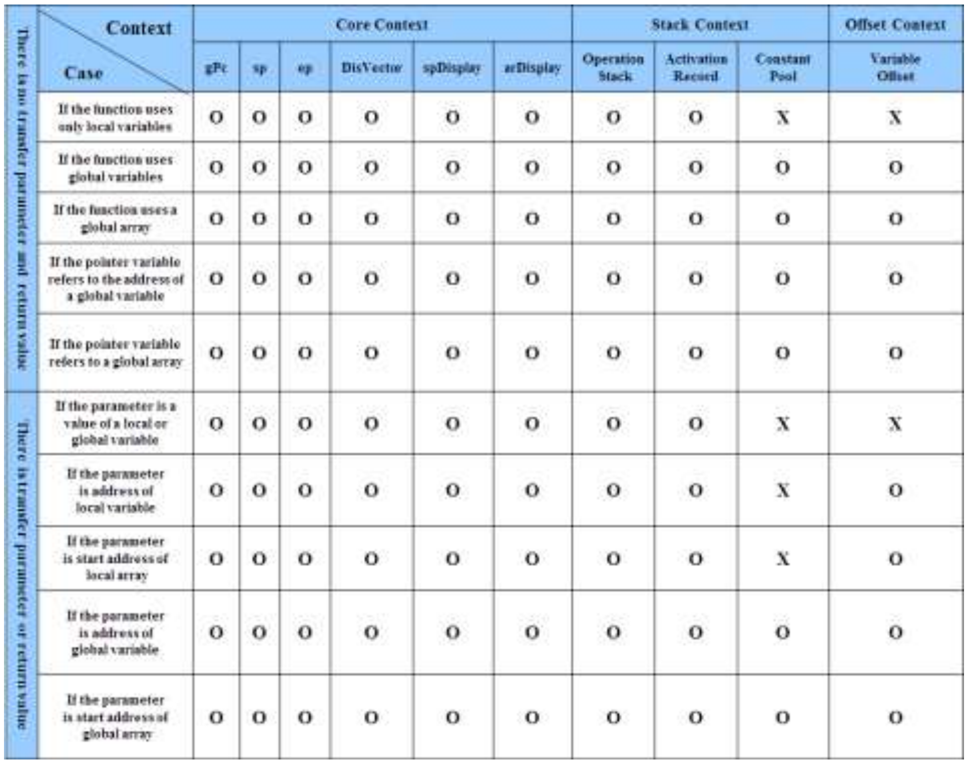
Figure 3. Stack Context Information

The operation stack finishes the function execution at the start position of the function frame, loads the return address and the lexical level of the function, and executes the operation instruction and the load instruction from the subsequent area. The activation record, similar to the operation stack, loads the address to be returned at the start of the function frame, the lexical level of the function, and manages the local variables used in the function from then on.

3. Context Information Synchronization Technique

In the virtual machine to which the conventional offloading technique is applied, all context information is transmitted and synchronized at the time of offloading request. Context information synchronization with such a structure requires a high computing power in low performance IoT devices because the resources consumed in transmitting and receiving context information are large. In order to solve this problem, we have studied a context information synchronization technique that collects only context information that requires an offloading performance function. Figure 4 shows the case of the offloading function according to the context information.

Since context information to be synchronized in offloading differs from case to case in the proposed context synchronization method, it is necessary to study synchronization on a case-by-case basis. Therefore, in this paper, we have studied the essential context information and stack context information needed in all cases for basic program execution.



	Context / Case	Core Contest						Stack Contest			Offset Context
		gPc	sp	op	DisVector	spDisplay	arDisplay	Operation Stack	Activation Record	Constant Pool	Variable Offset
There is no transfer parameter and return value	If the function uses only local variables	O	O	O	O	O	O	O	O	X	X
	If the function uses global variables	O	O	O	O	O	O	O	O	O	O
	If the function uses a global array	O	O	O	O	O	O	O	O	O	O
	If the pointer variable refers to the address of a global variable	O	O	O	O	O	O	O	O	O	O
	If the pointer variable refers to a global array	O	O	O	O	O	O	O	O	O	O
There is transfer parameter or return value	If the parameter is a value of a local or global variable	O	O	O	O	O	O	O	O	X	X
	If the parameter is address of local variable	O	O	O	O	O	O	O	O	X	O
	If the parameter is start address of local array	O	O	O	O	O	O	O	O	X	O
	If the parameter is address of global variable	O	O	O	O	O	O	O	O	O	O
	If the parameter is start address of global array	O	O	O	O	O	O	O	O	O	O

Figure 4. Case of the Offloading Function

3.1. Essential Context Information Synchronization

The essential context information consists of the context buffer, which is essential for the interpreter to execute the program, and a pointer to the instruction buffer, a frame pointer to the start frame of the function, and pointers to access each context information. In the conventional context synchronization, pointers pointing to the function start frame information of the operation stack and the activation record transmitted all the pointers indicating the start frame information of the function.

However, since the information required for offloading is the frame information of the offloading performing function, if only the pointer pointing to the starting frame of the offloading performing function is synchronized, the size of data is reduced and resources consumed in communication are reduced. The pointers except the pointer to the start frame position of the function are collected and transmitted each time the offloading is performed because the value changes flexibly when the command is executed in the interpreter.

3.2. Stack Context Information Synchronization

The stack context information consists of an operation stack, an activation record, and a constant pool. Unlike the constant pool, the operation stack and the activation record are context information required in all function cases. Therefore, in this section, we have studied the synchronization technique between the operation stack and the activation record.

3.2.1. Operation Stack Synchronization

The operation stack is the context information used when executing the load and compute instructions in the interpreter. The operation stack increments the start frame of the function to execute the instruction of the function to be executed at the time of the function call, and loads the address to be returned after completing the function execution. Figure 5 shows the synchronization area of the operation stack when synchronizing existing context information and improved context information when performing offloading.

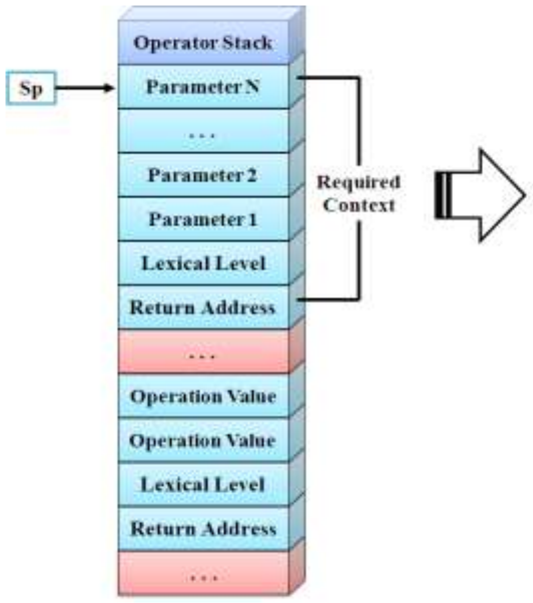


Figure 5. Synchronization Area of the Operation Stack

Since the operation stack executes the instruction by a function, the area requiring synchronization is a frame area of the function to be offloaded. Therefore, the operation stack synchronizes from the start frame position of the offloading performing function to the position where the parameter value is loaded, and reduces the data size of the operation stack to collect when loading. If there is a return value after performing offloading, the data size of the operation stack to be collected is reduced by performing synchronization from the start frame position of the executed function to the position where the return value is loaded.

3.2.2. Activation Record Synchronization

The activation record is context information in which local variables necessary for function execution are managed. The activation record updates the start frame position information of the function to manage the local variables of the function to be executed when the function is called.

Since the activation record manages local variables by function, the area that needs to be synchronized during offloading is the frame area of the function to be offloaded. Therefore, the activation record is synchronized from the start frame position of the function to be offloaded to the end of the function frame, thereby reducing the data size of the activation record synchronized at the time of offloading. In addition, when the function is completed in the offloading environment, the function frame of the activation record disappears, so synchronization of the activation record does not proceed after offloading. Figure 6 shows the synchronized area of the active record when synchronizing existing and improved context information.

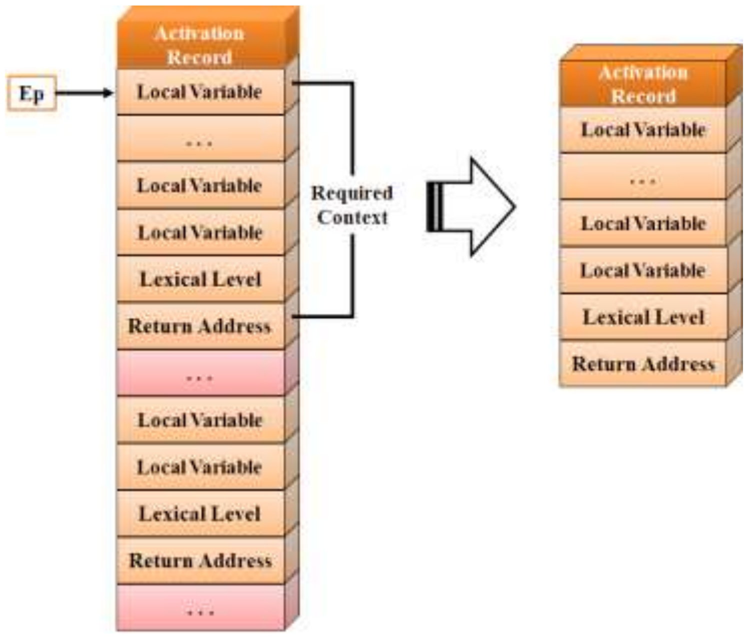


Figure 6. Synchronized Area of the Activation Record

4. Experimental Results

In order to confirm whether the essential context information and the stack context information are normally synchronized in the IoT-Cloud fusion virtual machine applying the proposed context information synchronization method, before the context information synchronization was performed, the context information of the server received the context information transmitted from the IoT device and confirmed that the context information coincided with the context information of the IoT environment at the time of offloading request after synchronization. Also, it is confirmed that the amount of context information is reduced by comparing the amount of context information transmitted in the context synchronization of the existing virtual machine with the amount of context information generated in the virtual machine using the context information synchronization technique. Table 1 shows the experimental environment for contextual information synchronization.

Table 1. Experimental Environment

Server	PC
CPU	Intel(R) Core(TM) i7-4770 3.50Ghz
Memory	16GB RAM
OS	Microsoft Windows 10

Device	RaspberryPi B+
CPU	Quad-Core ARM Cortex-A7 900MHz
Memory	512MB RAM
OS	Raspbian

Example 1 is the source code of the prime number algorithm used to apply the proposed context information synchronization technique.

```
prime.cpp

#include "sys_lib.h"
void prime(int n);
void main(void) {
            prime(3000);
}
void prime(int n)
{ int i, j, prime;
  for (i = 2; i <= n; i++) {
            for (j = 2; j < n; j++) {
                if (i%j == 0) { break; }
            }
            if (i == j) {
              printf("%d", i);
            }
   }
}
```

Example 1. Prime Number Source Code

Tables 2, 3, 4, and 5 show the context information generated before and after applying the proposed contextual information synchronization technique using the examples in the experimental environment. Table 2 shows the context information of the server before contextual information synchronization is performed.

Table 2. Context Information before Synchronization is Performed

Context Data	Value before improvement	Value after improvement
gPc	-842150451	-842150451
DisVector	0	0
sp	1	1
ep	1	1
SpDisplay	0	0
ArDisplay	0	0
Operation Stack	-1, 1	-1, 1
Activation Record	-1, 1	-1, 1

Before contextual information synchronization is performed, the context information of the server is not initialized, and the context information before and after the context information synchronization technique are the same. The context information of the server should be in the same state as the context information of the IoT equipment when requesting offloading after context information synchronization.

Tables 3, 4, and 5 are context information generated in the process of synchronizing the context information of the IoT device to the server environment. Table 3 shows the context information of the IoT devices at the time of offloading request, and Table 4 shows the context information transmitted from the IoT device to the server. When the context information is transmitted from the IoT device to the server, the context information transmitted from the virtual machine to which the context information synchronization technique is not applied transmits the entire context information of the IoT device when the offloading request is made, and It can be confirmed that the virtual machine applying the synchronization scheme transmits the frame region of the offloading function.

Table 3. Context Information of the IoT Devices at the time of Offloading Request

Context Data	Value before improvement	Value after improvement
gPc	6	6
DisVector	1	1
sp	4	4
ep	3	3
SpDisplay	0, 2	0, 2
ArDisplay	0, 2	0, 2
Operation Stack	−1, 1, 4, 1, 3000	−1, 1, 4, 1, 3000
Activation Record	−1, 1, 4, 1	−1, 1, 4, 1

Table 4. Context Information Transmitted from the IoT Device to the Server

Context Data	Value before improvement	Value after improvement
gPc	6	6
DisVector	1	1
sp	4	4
ep	3	3
SpDisplay	0, 2	2
ArDisplay	0, 2	2
Operation Stack	−1, 1, 4, 1, 3000	4, 1, 3000
Activation Record	−1, 1, 4, 1	4, 1

Table 5 shows context information after synchronization in the server to check whether the context information transmitted from the IoT device is normally synchronized.

Table 5. Context Information after Synchronization in the Server

Context Data	Value before improvement	Value after improvement
gPc	6	6
DisVector	1	1
sp	4	4
ep	3	3
SpDisplay	0, 2	2
ArDisplay	0, 2	2
Operation Stack	−1, 1, 4, 1, 3000	−1, 1, 4, 1, 3000
Activation Record	−1, 1, 4, 1	−1, 1, 4, 1

After context information synchronization in the server, the context information is the same as the context information of the IoT device at the offloading request point, so that the context information transmitted from the IoT device can be confirmed to be normally synchronized.

We compare the amount of context information transmitted from existing virtual machines to confirm that the amount of context information to be transmitted is reduced when applying the context information synchronization scheme. Figure 7 compares the amount of context information transmitted from the virtual machine before and after applying the context information synchronization scheme proposed in the prime number algorithm example.

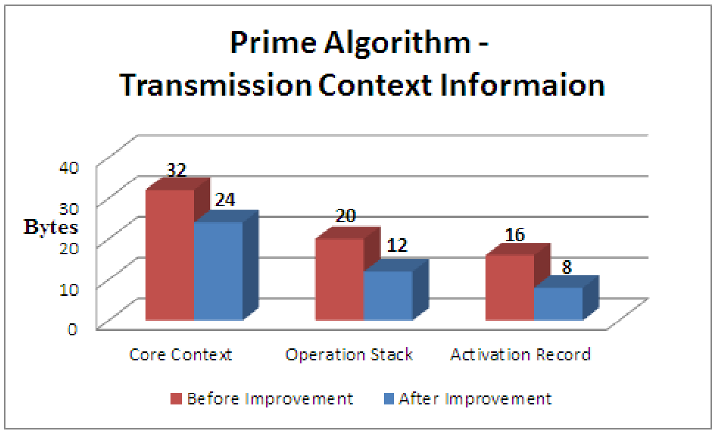


Figure 7. Synchronized Area of the Activation Record

The amount of context information to be transmitted before and after application of the context information synchronization technique can be confirmed to be reduced by two times in the case of the activated record and to be 1.6 times in the case of the operation stack, and 1.3 times in the case of the essential record information. Therefore, it can be confirmed that the amount of context information to be synchronized is reduced before applying the proposed context information synchronization method.

5. Conclusions and Further Researches

The virtual machine to which the conventional offloading technique was applied needs to synchronize all the context information when the context information is synchronized for communication between the server and the local. Since the amount of context information to be transmitted is large, the resource consumption rate of context information synchronization is increased in low performance IoT devices.

In order to solve this problem, this paper classifies cases of offloading function according to differentiated information by distinguishing context information required to perform offloading function, and context information synchronization method which synchronizes only the context information necessary for execution function Respectively. For this purpose, we have studied the essential context information, the operation stack, and the activation record synchronization, which are essential context information in the context information synchronization technique. In the case of the required context information, the data size of the pointer that manages the start frame to be transmitted is reduced, and the operation stack and the activation record use the common point of managing the resources in the function frame unit, thereby reducing the size of each data by synchronizing only the frame region necessary for the function execution. The virtual machine using the proposed context information synchronization scheme can reduce the resource consumption rate of IoT equipment because the amount of data to be transmitted is smaller than that of the existing virtual machine when the context information is synchronized.

Currently, the context information synchronization technique synchronizes only context information that is essential for function execution. Offloading function In the case of using global variables in the function to perform offloading, IoT equipment needs synchronization every time global variable value is assigned, and when reference variable such as pointer, array, variable address is used, referenced memory between server and local must be shared. In order to solve this problem, we plan to study the synchronization method of context information in which global variables are managed based on analyzed information by analyzing the use of global variables of functions through the static profiler. Also, we will study the reference area synchronization method to refer to the same area in the server when loading, and to minimize the overhead caused by context information synchronization in low performance IoT devices to maximize the offloading efficiency.

Acknowledgments

This research was supported by Seokyeong University in 2016.

References

[1] S. M, Han, Y. S. Son, Y. S. Lee, "Design and Implementation of the Smart Virtual Machine for Smart Cross Platform," Journal of Korea Multimedia Society, Vol. 16, No. 2, pp. 190-197, 2013.
[2] Y.S. Lee, Y.S. Son, "A Study on the Smart Virtual Machine for Executing Virtual Machine Codes on Smart Platforms", International Journal of Smart Home, SERSC, Vol. 6, No. 4, 2012, Australia, pp. 93-105.
[3] Y.S. Lee, Y.S. Son, "A Study on the Smart Virtual Machine for Smart Devices", Information -an International Interdisciplinary Journal, Vol. 16, No. 2, International Information Institute, 2013, Japan, pp.1465-1472.
[4] Y.S. Lee, S.M. Oh, Y.S. Son, "Design and Implementation of HTML5 based SVM for Integrating Runtime of Smart Devices and Web Environments", International Journal of Smart Home, SERSC, Vol.8, No.3, 2014, Australia, pp.223-234.
[5] K. Yang, S. Ou, and H.H. Chen, "On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications," IEEE Comm. Magazine, Vol. 46, No. 1, pp. 56-63, 2008.
[6] Kumar, Karthik, "A Survey of Computation Offloading for Mobile Systems," Mobile Networks and Applications Vol. 18, No. 1, pp. 129-140, 2013.
[7] Shi, Cong, "Cosmos: Computation Offloading as a Service for Mobile Devices," Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing. ACM, pp. 287-296, 2014.
[8] B. G. Chun, "Clonecloud: Elastic Execution between Mobile device and Cloud," Proceedings of the sixth Conference on Computer Systems. ACM, pp. 301-314, 2011.
[9] K. Kumar and Y. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," Computer, vol. 43, no. 4, pp. 51–56, 2010.
[10] H. Min, J.M. Jung, J.Y. Heo. "A Function level Static Offloading Scheme for Saving Energy of Mobile Devices in Mobile Cloud Computing." Journal of Korea Information Science Society, Vol. 42, No. 6, pp. 707-712. 2015.

Authors

JaeHyun Kim, he received the B.S. degree from the Dept. of Mathematics, Hanyang University, Seoul, Korea, in 1986, and M.S. and Ph.D. degrees from Dept. of Statistics, Dongguk University, Seoul, Korea in 1989 and 1996, respectively. He was a chairman of Dept. of Internet Information 2002-2007. Currently, he is a member of the Korean Data & Information Science Society and a Professor of Dept. of Computer Engineering, Seokyeong University, Seoul, Korea. His research areas include mobile programming, cloud system and big data analysis.

Yangsun Lee, he received the B.S. degree from the Dept. of Computer Science, Dongguk University, Seoul, Korea, in 1985, and M.S. and Ph.D. degrees from Dept. of Computer Engineering, Dongguk University, Seoul, Korea in 1987 and 2003, respectively. He was a Manager of the Computer Center, Seokyeong University from 1996-2000, a Director of Korea Multimedia Society from 2004-2005, a General Director of Korea Multimedia Society from 2005-2006, a Vice President of Korea Multimedia Society in 2009, and a Senior Vice President of Korea Multimedia Society in 2015-2016. Also, he was a Director of Korea Information Processing Society from 2006-2014 and a President of a Society for the Study of Game at Korea Information Processing Society from 2006-2010. And, he was a Director of HSST from 2014-2017. Currently, he is a Professor of Dept. of Computer Engineering, Seokyeong University, Seoul, Korea. His research areas include smart system solutions, programming languages, and embedded systems.