Research Article

Automatic Optimizer Generation Method Based on Location and Context Information to Improve Mobile Services

Yunsik Son,¹ Junho Jeong,² and Yangsun Lee³

¹Department of Computer Science and Engineering, Dongguk University, 3-26 Pil-dong, Jung-gu, Seoul 100-715, Republic of Korea

²Electronic Commerce Institute, Dongguk University, 123 Dongdae-ro, Gyeongju-si, Gyeongbuk 780-714, Republic of Korea

³Department 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; Accepted 9 February 2017; Published 14 March 2017

Academic Editor: Byeong Ho Kang

Copyright © 2017 Yunsik Son et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Several location-based services (LBSs) have been recently developed for smartphones. Among these are proactive SBSs, which provide services to smartphone users by periodically collecting background logs. However, because they consume considerable battery power, they are not widely used for various SBS-based services. Battery consumption, in particular, is a significant issue on account of the characteristics of mobile systems. This problem involves a greater service restriction when performing complex operations. Therefore, to successfully enable various services based on location, this problem must be solved. In this paper, we introduce a technique to automatically generate a customized service optimizer for each application, service type, and platform using location and situation information. By using the proposed technique, energy and computing resources can be more efficiently employed for each service. Thus, users should receive more effective SBSs on mobile devices, such as smartphones.

1. Introduction

Recently, several location-based services (LBSs) have been developed for smartphones. Of these, proactive SBSs provide services to smartphone users based on periodically collected background logs. However, since these services consume considerable battery power, it is difficult to widely use them for various SBSs on account of the characteristics of mobile devices with constrained batteries. Therefore, to successfully provide various services based on localization, the problem of overcoming the limited resources of mobile devices must be solved.

In addition, mobile devices have various platforms and require different development approaches depending on the platform. The smart cross-platform is a virtual machine-based solution that can run the same content on various smart devices. If the smart cross-platform is used to provide SBSs, it is not necessary to perform separate development for each platform. However, because the smart cross-platform has a virtual machine- (VM-) based execution environment, the performance of a program running in a VM is lower than that of a general native-code base.

In this paper, we introduce a technique to automatically generate a customized optimizer for each application, service type, and platform using location and situation information. By using the proposed technique, energy and computing resources can be more efficiently employed for each service. Thus, users can receive more effective SBSs on mobile devices, such as smartphones.

The remainder of this paper is organized as follows. In Section 2, we examine context-based optimization, characteristics of the existing smart cross-platform, the smart VM, and existing optimization techniques. Section 3 describes the code optimizer generator proposed in this paper. In Section 4, we demonstrate the performance of the optimizer. Finally, in Section 5, we present our conclusions.

2. Related Works

2.1. Context-Based Optimization Algorithms. To optimize energy consumption in SBSs, Ben Abdesslem et al. [1] proposed SensLess. This algorithm uses an accelerometer to detect motion and stasis, thereby deactivating localization. Therefore, energy consumed by unnecessary localization is

FIGURE 1: System model of smart cross-platform.

reduced. Nevertheless, the accelerometer must be operated at all times. Zhuang et al. [2] proposed sensor substitution and sensor suppression techniques to reduce energy consumption. The sensor substitution dynamically selects the localization method according to the automatically defined M-area whenever Global Positioning System coordinates are not available. In addition, sensor suppression technology requires continuous monitoring of the sensor, such as SensLess, by disabling localization using an accelerometer and a digital compass. Kim et al. [3] proposed the SenLoc algorithm. It prevents unnecessary energy consumption by more accurately detecting movement of a device with a three-axis accelerometer instead of simple movement. However, it can only be used in a stable Wi-Fi environment.

On the other hand, in the LBS, it is important to use a cloud to apply key personal information in accordance with each service. Therefore, effective retrieval of encrypted data is very important. Meanwhile, Fu et al. [4] proposed an effective technique for synonym-based multikeyword ranked searching over encrypted cloud data.

2.2. Virtual Machine Execution and Native Execution Method.

In terms of content execution, a difference exists between the VM execution method and the native execution method. First, since the VM execution method has hardware-independent characteristics, it is easy to execute contents even if the hardware is changed. Moreover, contents can be easily transplanted to various hardware platforms. In addition, even if there is an error in the content, the target system can be continuously operated. On the other hand, owing to the software execution limitation, the performance of complex algorithms is poorer than those of native methods.

2.3. Smart Cross-Platform and SVM. Smart Virtual Machine (SVM) [5, 6] is a stack-based VM solution that can be loaded on a smart device to download and run dynamic applications

on a platform-independent basis. SVM is designed to use the Smart Intermediate Language (SIL), an intermediate language that can accommodate both sequential and object-oriented languages. It has the advantage of being able to accommodate C/C++ and Java languages, as well as the Objective C language used by the iOS mobile operating system.

Figure 1 shows the overall system configuration of SVM with the smart cross-platform.

SVM and the smart cross-platform system consist of three parts: a compiler that compiles an application to generate a Standard Archive Format- (SAF-) formatted file consisting of SIL code, an assembler that converts the SAF file into an executable format, and a VM that receives and executes files in Smart Executable Format (SEF) format [7]. The smart cross-platform system is designed to be hierarchical and to minimize the burden on repurposing processes for other devices and operating environments. The SIL file from the compiler/translation process is converted into the SEF format through the assembler; SVM accepts the SEF as input and executes the program.

3. Location/Context Information-Based Automatic Optimization Model

In this paper, we introduce a technology that provides optimal execution performance by customizing various LBSs on mobile devices. The customized LBS optimization technology for mobile devices generates a pattern for each LBS using the application, context, and location information. It then optimizes the VM configuration and the application code to enable the corresponding service to be optimally executed. Figure 2 shows the LBS execution model using the proposed technique.

The service model for each mobile device using LBS is shown in Figure 3. This section introduces the service

FIGURE 2: Customized LBSs optimization model.

FIGURE 3: Configuration of the service optimizer generator.

optimizer generator (SOG) and the VM configuration for a suitable LBS algorithm on the target service with the deterministic finite automata- (DFA-) based optimization table for generating context- and application-level code optimization. The SOG proposed in this paper generates a DFA that recognizes a user-defined pattern. The optimizer optimizes

the input code using it. On account of the decisive automata, the optimizer can rapidly process the input code.

3.1. Service Optimizer Generator System Model. The service optimizer generator consists of a scanner, parser, gathered information, calculated C_0 , the created DFA, and a write

FIGURE 4: Code optimizer generation model using DFA optimization table.

optimization_pattern_description	-> '%localization_algorithm_type'	
	'%%'	
	context_patterns	
	'%%'	
	application_pattern	=> OPT_PATTERN_DESCRIPTION;
context_patterns	-> pattern_description	=> CTX_PATTERN_DESCRIPTION;
application_patterns	-> pattern_description	=> APP_PATTERN_DESCRIPTION;
pattern_description	-> patterns;	
patterns	-> pattern;	
	-> pattern patterns;	
	-> instr_code_sets	
	'::=' instr_code_sets	
	'%%' '%eval_methods' '%%'	=> PATTERN;
instr_code_sets	-> instr_code_set;	
	-> instr_code_set instr_code_sets;	
instr_code_set	-> '%instruction' opt_parameters '/'	=> INSTR_CODE_SET;
opt_parameters	-> ;	
	-> parameters;	
parameters	-> '%param';	
	-> '%param' parameters;	

Box 1: Optimization pattern description format (BNF style).

table. The scanner is the first procedure of the compiler. It creates tokens, which are syntactically significant minimum units. In other words, it receives a pattern description as an input and generates a series of tokens [8, 9]. The parser receives the token output from the scanner, checks for errors, and creates sentences according to the program syntax to generate an abstract syntax tree (AST).

The information-gathering stage involves collecting information of optimal localization methods, as well as context/application-level optimization information, to comprise the pattern description content. The content is traversed into a tree structure through the scanner and parser. The data structure is a two-dimensional layout with an integrated linked list. It is converted into a data structure wherein the pattern information and look-ahead assertion

are integrated to calculate the C_0 . The stage of “making” the DFA turns the calculated C_0 into a DFA format.

The DFA normalization process involves repetitions of “Calculating C_0 ” and “Making DFA” stages. Since the resultant DFA is normalized, optimization actions are easily performed. In other words, the DFA-type backtracking task is not necessary. Figure 4 shows the model of the code optimizer generation model using the DFA table from SOG.

3.2. Code Optimizer Pattern Description. A description of the optimization pattern is comprised of replace and pattern parts in the BNF style shown in Box 1. There are no length restrictions for the replace and pattern parts, and a single pattern comprises a single line.

```

addiv $1 $2 $3 $4 ::= 'lod.i' $5 $6 / 'lod.i' $7 $8 / 'add.i' %% $1 = $5; $2 = $6; $3 = $7; $4 = $8 %%
subiv $1 $2 $3 $4 ::= 'lod.i' $5 $6 / 'lod.i' $7 $8 / 'sub.i' %% $1 = $5; $2 = $6; $3 = $7; $4 = $8 %%
...
ldc.i.m1          ::= 'ldc.i' $1 %% $1 == -1 %%
ldc.i.0           ::= 'ldc.i' $1 %% $1 == 0 %%
...
incv.i $1 $2 $3   ::= 'lod.i' $5 $6 / 'ldc.i' $7 / 'add.i' / 'str.i' $8 $9
                  %% $5 == $8; $6 == $9; $1 = $5; $2 = $6; $3 = 7 %%
decv.i $1 $2 $3   ::= 'lod.i' $5 $6 / 'ldc.i' $7 / 'sub.i' / 'str.i' $8 $9
                  %% $5 == $8; $6 == $9; $1 = $5; $2 = $6; $3 = 7 %%
...
jgt.i $1 $2 $3 $4 $5 ::= 'lod.i' $6 $7 / 'lod.i' $8 $9 / 'le.i' / 'fjp' $10
                  %% $1 = $6; $2 = $7; $3 = $8; $4 = $9; $5 = 10 %%
jle.i $1 $2 $3     ::= 'lod.i' $6 $7 / 'lod.i' $8 $9 / 'gt.i' / 'fjp' $10
                  %% $1 = $6; $2 = $7; $3 = $8; $4 = $9; $5 = 10 %%
...

```

Box 2: Application-level pattern description example for SOG.

To distinguish the instruction code sets, the replace parts, pattern parts, and pattern descriptions are described with the separator “::=.” Accordingly, the instruction code sets of the pattern part are described. The instruction code sets are described with instructions, parameters, and the separator “/.” In addition, the localization algorithm type can be SensLess, SenLoc, Zhuang’s localization algorithm, and so forth. These have been demonstrated in related studies [2–4].

3.3. Code Optimizer Using DFA Optimization Table. Figure 4 shows the structure of the code optimizer using the DFA optimization table, as shown in Box 2. The structure is generated from SOG through an optimization pattern description created with application, location, and context information. The code optimizer consists of five modules: a DFA optimization table, driving routine, code reader, code writer, and an instruction list.

First, when the user creates a pattern by employing the pattern specification grammar introduced in Box 1, the SOG generates a DFA-based optimization table corresponding to the pattern.

The DFA table consists of respective state and instruction sets for the code. The program input by the optimizer attempts to match with optimization instruction pattern using the instruction read by the predefined driving routine and the information recorded in the DFA optimization table. The code input unit analyzes the instruction of the inputted SAF code and converts it into a list of instructions comprised of a double connected list. In the instruction list, the optimized code table and instruction partial substitution matched by the driving routine are replaced with optimization instructions. It thus generates an optimized SAF code.

4. Experimental Results

In an experiment, the optimization code pattern for the HTML5-based smart cross-platform was entered into the

```

...
int parsingTable[NO_STATES][NO_SYMBOLS+1] = {
    /** state 0 **/
    0, 0, 0, 7, 0, 0, 6, 0, 5, 4, ...
    3, 2, 1},
    /** state 1 **/
    -3, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
    0, 0, 0},
    /** state 2 **/
    -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
    0, 0, 0},
    /** state 3 **/
    -4, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
    0, 0, 0},
    ...

```

Box 3: Example of the generated DFA table.

implemented SOG and the generated optimizer was evaluated. Box 2 shows the optimization pattern of the SIL instruction defined according to the pattern specification grammar outlined in Box 1. The matching information of the optimization code for the parameter reduction, increase/decrease operation, and control instruction and the related conditions are provided in [8, 9].

Box 3 presents part of the DFA optimization table generated by SOG. As shown in the source code of the generated DFA table in Box 3, the next states to transit according to the input instruction are recorded for each target state.

Table 1 outlines data of the SIL code converted through the implemented optimizer. The input programs are a hash algorithm and tower of Hanoi algorithm source. The data show that the operator, parameter abbreviation, and control instruction for referencing the array variable were changed to the optimized SIL instruction.

The results of comparing the pattern matching performance with the string-pattern matching and tree-pattern

TABLE 1: Optimization results.

Hash.saf	Hash.saf (optimized)	hanoi.saf	hanoi.saf (optimized)
%Line 143: scanf("%s", operation);	%Line 143: scanf("%s", operation);	%Line 25: if (num == 1) move(from, to);	%Line 25: if (num == 1) move(from, to);
ldp	ldp	lod.i 1 12	lod.i 1 12
ldc.p @0x0247	ldc.p @0x0247	ldc.i 1	ldc.i.1
lda 1 0	lda 1 0	eq.i \$0	jne.i \$0
call scanf	call scanf	fjp	
%Line 146: operation[0] == 'c'	%Line 146: operation[0] == 'c'	%Line 25: if (num == 1) move(from, to);	%Line 25: if (num == 1) move(from, to);
\$\$22: nop	\$\$22: nop	ldp	ldp
lda 1 0	ldc.c 1 0 0	lod.i 1 0	lod.i 1 0
ldc.i 0	cvc.i	lod.i 1 4	lod.i 1 4
cvi.ui	ldc.c 97	call move	call move
cvui.p	cvc.i	ujp \$1	ujp \$1
add.p	eq.i	nop	nop
ldi.c	dup	\$0: \$0: %Line 26: else {transfer(from, spare, to, num-1);	\$0: \$0: %Line 26: else {transfer(from, spare, to, num-1);
cvc.i 97	tjp		
ldc.c	pop		
cvc.i	ldc.c 1 0 0	ldp	ldp
eq.i	cvc.i	lod.i 1 0	lod.i 1 0
dup	ldc.c	lod.i 1 8	lod.i 1 8
tjp	cvc.i	lod.i 1 4	lod.i 1 4
pop	eq.i	lod.i 1 12	lod.i 1 12
lda 1 0		ldc.i.1	ldc.i.1
ldc.i 0		sub.i	sub.i
cvi.ui		call	call
cvui.p			
add.p			
ldi.c			
cvc.i			
ldc.c 98			
cvc.i			
eq.i			
			transfer

matching-based optimizer showed the same pattern as the DFA-based optimizer generated through SOG. Since the generated code optimizer performed pattern matching based on DFA, the input program could be rapidly analyzed by converting the SIL code optimization pattern into a deterministic finite automata.

Furthermore, the search time was decreased by 18% and 6%, respectively, compared with the conventional string-pattern matching and tree-pattern matching [10–12]. Additionally, because the DFA table was generated using the SOG, an optimizer for performing the optimization code pattern matching according to the pattern definition could be automatically generated.

5. Conclusions

In this paper, we proposed a method to generate an automatic optimizer from user-defined pattern information based on location and context information. It can effectively support LBS and improve the performance of smart cross-platform VMs. Because the pattern is represented by DFA, it shows faster optimization performance compared to existing string-pattern matching and tree-pattern matching. This makes it easy to modify or add the pattern information of the optimizer.

Moreover, the optimizer is automatically generated by the SOG; therefore, a separate optimizer correction due to the pattern change does not occur. In this study, we used DFA for fast pattern matching, which means that the expression level of the pattern had a limit that could not extend beyond the regular language. To solve this problem, we defined an area for describing a separate evaluation method in the optimization pattern.

In this paper, we introduced a technique for automatically generating a customized optimizer for each application, service type, and platform using location and context information. The proposed technique can more efficiently use more energy and computing resources for each service. Thus, users can obtain more effective LBSs on mobile devices, such as smartphones.

Our future studies will be based on the optimization pattern of the peephole level, which was limited in this study. In addition, the pattern technique that can apply the data flow will be expanded and developed. Using these studies, we will apply higher-level optimization techniques to a novel optimizer.

Disclosure

This paper was extended from the previous research paper “A Study on the Code Optimizer Generator for the Smart Cross Platform”, in *Advanced Science and Technology Letters*, 2016 [13].

Competing Interests

The authors declare that there is no conflict of interests 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. 2013RIA2A2A01067205 and no. 2016RIA2B4008392).

References

- [1] F. Ben Abdesslem, A. Phillips, and T. Henderson, “Less is more: energy-efficient mobile sensing with senseless,” in *Proceedings of the 1st ACM Workshops on Networking, Systems, and Applications for Mobile Handhelds (MobiHeld '09)*, pp. 61–62, Barcelona, Spain, August 2009.
- [2] Z. Zhuang, K.-H. Kim, and J. P. Singh, “Improving energy efficiency of location sensing on smartphones,” in *Proceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '10)*, pp. 315–330, New York, NY, USA, June 2010.
- [3] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava, “SensLoc: sensing everyday places and paths using less energy,” in *Proceedings of the 8th ACM International Conference on Embedded Networked Sensor Systems (SenSys '10)*, pp. 43–56, Zurich, Switzerland, November 2010.
- [4] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, “Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing,” *IEICE Transactions on Communications*, vol. E98B, no. 1, pp. 190–200, 2015.
- [5] Y. Son and Y. S. Lee, “A study on the smart virtual machine for smart devices,” *Information*, vol. 16, no. 2, pp. 1465–1472, 2013.
- [6] 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.
- [7] Y. S. Lee, J. Jeong, and Y. Son, “Design and implementation of the secure compiler and virtual machine for developing secure IoT services,” *Future Generation Computer Systems*, 2016.
- [8] N. Kumar and S. Hiranwal, “Current trends in the field of code optimization,” in *Proceedings of the International Conference on Emerging Trends in Engineering & Management for Sustainable Development*, pp. 1–6, 2016.
- [9] S. Cherubin, M. Scandale, and G. Agosta, “Stack size estimation on machine-independent intermediate code for OpenCL kernels,” in *Proceedings of the the 7th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and the 5th Workshop on Design Tools and Architectures For Multicore Embedded Computing Platforms (PARMA-DITAM '16)*, pp. 1–6, Prague, Czech Republic, January 2016.
- [10] A. S. Tanenbaum, H. van Staveren, and J. W. Stevenson, “Using peephole optimization on intermediate code,” *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 1, pp. 21–36, 1982.
- [11] W. M. McKeeman, “Peephole optimization,” *Communications of the ACM*, vol. 8, no. 7, pp. 443–444, 1965.
- [12] C. M. Hoffmann and M. J. O'Donnell, “Pattern matching in trees,” *Journal of the Association for Computing Machinery*, vol. 29, no. 1, pp. 68–95, 1982.
- [13] Y. Son and Y. S. Lee, “A study on the code optimizer generator for the smart cross platform,” *Advanced Science and Technology Letters*, vol. 133, pp. 138–143, 2016.

Hindawi

Submit your manuscripts at
<https://www.hindawi.com>