Cross-Layer Real-Time Support for JVM-based Smartphone Systems

Young Joo Woo[1], Jungwook Cho[1], Donghyouk Lim[2] and Euiseong Seo[1]
[1]Ulsan National Institute Science and Technology
[2]Electronics and Telecommunications Research Institute

***Abstract*-- The existence of the JVM layer hinders applications from notifying the operating system scheduler about their timeliness requirements and, therefore, the applications sometimes fail to respond on time. This research proposes a cross-layer real-time support by which applications notify operating systems about their timeliness requirements. Our prototype shows significant improvements in the response times and throughputs of prioritized applications.**

I. INTRODUCTION

The Java virtual machine (JVM) architecture fundamentally prevents applications from affecting the stability and security of the whole system by monitoring and strictly controlling the behaviors of applications that run through it. Smartphones, which may run uncertified third-party applications, can protect the system from unstable or malicious third-party applications by employing the JVM architecture. Therefore, some smartphone systems including Android utilize the JVM architecture, and its use is expected to increase in future smartphone operating systems.

Most smartphone applications are interactive or multimedia-related such as games, media players and web browsers. The quality of those applications commonly relate to the scheduling latency. In order to fulfill the timeliness requirements, most embedded OS kernels offer real-time schedulers in one way or another.

Real-time schedulers rely on the information about the timeliness requirements of applications, which is provided by the applications to the kernel. However, because the existence of the JVM layer prohibits applications from directly accessing the kernel interfaces, applications in JVMs cannot benefit from the real-time scheduler. Running JVMs through the real-time scheduler is fundamental to applications` timely responses [1].

This paper aims to suggest and evaluate the cross-layer real-time support framework that enables Java-based applications to utilize the kernel-level real-time scheduler in smartphone operating systems. The suggested framework is implemented as a Java library with Java application programming interface (API) methods that can be called externally to deliver the timeliness requirements of Java applications to OS kernels.

II. PROPOSED CROSS-LAYER FRAMEWORK

In Java-based smartphone operating systems, a JVM runs as a user-level task and an application runs inside the JVM. Generally, an application runs in a separate virtual machine. Therefore, the operating system can manage the responsiveness of the application only by manipulating the scheduling policy and the priority of its corresponding virtual machine [2].

The Linux kernel, which is popularly used as the kernel of many smartphone operating systems, currently employs the Completely-Fair Scheduler (CFS) [3] as its scheduler. The CFS has multiple scheduling queues including Normal, Batch, First-In-First-Out (FIFO) and Round-Robin (RR), and uses a different scheduling policy for each scheduling queue.

Both FIFO and RR are real-time class scheduling queues. When runnable tasks are in those queues, they will be scheduled ahead of other tasks in the Normal or Batch queues. Furthermore, they will never be preempted by other tasks that have lower priorities as long as they are runnable.

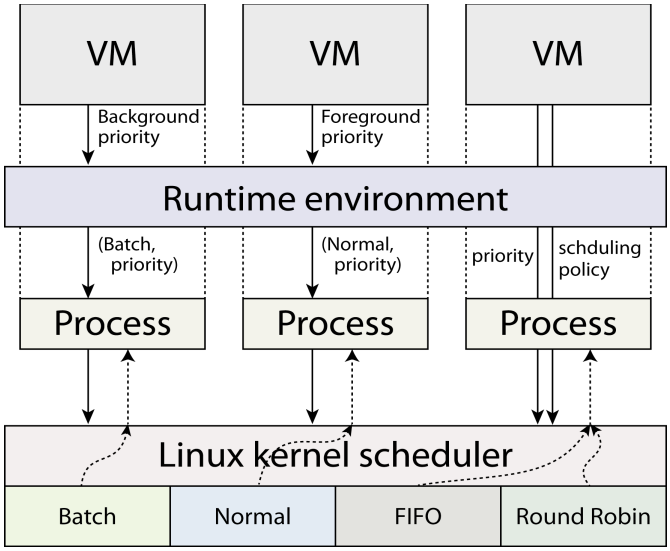

Fig. 1. The proposed real-time support framework directly passes the timeliness requirements of applications to the kernel-level scheduler

Android, which is an open source smartphone operating systems that uses the JVM architecture, categorizes applications tasks into two groups; foreground tasks and background tasks. Foreground tasks are placed in the Normal queue, whereas background tasks are run from the Batch queue. Currently, Android neither uses the real-time queues nor provides an interface for applications to set their scheduling priorities or policies.

We resolve this issue by proposing a cross-layer real-time support framework. The proposed framework is apparently a

This work was supported by the IT R&D program of MKE(Ministry of Knowledge Economy), Korea

Java library. Applications invoke the framework methods like other methods offered by the Android system framework. As shown in Fig. 1, through the Java methods of the suggested framework, applications can notify the kernel of their desired scheduling policies and priorities. When applications invoke a method of the framework, it calls the corresponding system call through the Java native interface (JNI).

In many cases, applications rely on other applications or system services [4]. For example, a music player charges the media server, which is a system service provided by the Android framework, with decoding music files and playing them. In these cases, prioritizing only the application does not guarantee a certain quality of music. Therefore, we prioritize both applications and their dependent services at the same time. However, identifying these dependency relationships among applications and services is difficult, and our scheme requires applications to explicitly notify the kernel of their dependent services or applications.

III. EVALUATION

The proposed scheme is implemented in the Android 2.2 operating system that runs on HTC Nexus One. We execute applications in the foreground and observe the quality of their services while running the hackbench benchmark in the background in order to impose a heavy load on the system.

First, we measure the buffer write delay of the media player while it plays music. If the media player or its related services are not properly scheduled on time, the music will play slower than normal, and it can be interrupted intermittently. Although delayed buffer writing does not always result in the retarded play or sound jitters, it has a strong relationship with such poor user experiences.

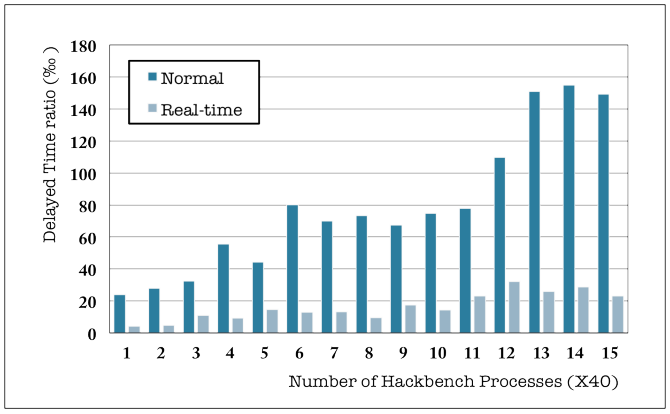

Fig. 2. Cumulative buffer writing delay of the music payer under varying number of Hackbench process groups

As shown in Fig. 2, more frequent and longer buffer write delays occur as more number of hackbench task groups run together when applications are not prioritized to the real-time class. This delay is reduced by approximately 85% under heavy loads by employing the proposed scheme. Thus emotional quality and quantitative results improve. Neither sound jitters nor retarded play occurred until we had run 15 task groups.

One limitation of the proposed approach is that priority inversion may occur when the scheduler fails to prioritize all dependent applications or services. To verify the effects of this limitation, we conducted an experiment by using a web browser that was set to load a web page periodically. As shown in Fig. 3, there is no significant difference between the conventional system and our scheme. The web browser uses its binder thread to communicate with other services. Because applications have their own binder threads and all binder threads are scheduled as a group, prioritizing only a single binder thread is not possible in the current system. Consequently, the scheduling delay of the web browser was not improved because of the scheduling delay of its dependent binder thread.

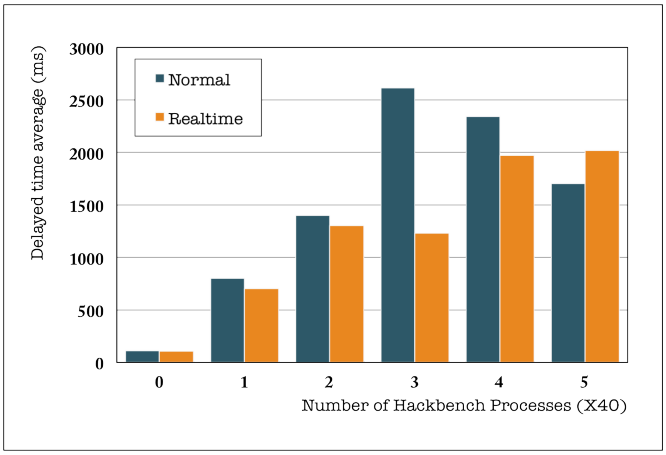

Fig. 3. Average loading delay of the Web browser under varying number of Hackbench process groups

IV. CONCLUSION AND FUTURE WORK

We suggested and evaluated a real-time requirement notification scheme for smartphone operating systems using JVMs for third-party applications. The suggested scheme reduced the scheduling latency by 85% under heavy loads in our evaluation.

However, in spite of this benefit, the present scheme cannot always provide real-time responsiveness because it does not automatically detect dependency relationships among applications and services and some threads cannot be prioritized to the real-time class. We are conducting further research to resolve these issues.

REFERENCES

[1] G. Bollella and J. Gosling, "The real-time specification for Java," Computer, vol. 33, pp. 47 -54, June 2000.

[2] J. Auerbach, *et al.*, "Design and implementation of a comprehensive real-time java virtual machine," in *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, ed. Salzburg, Austria: ACM, 2007, pp. 249-258.

[3] C. S. Pabla, "Completely fair scheduler," Linux Journal, vol. 2009, August 2009.

[4] C.-t. Man, et al., "Study of Priority Inversion in Embedded Linux," in Proceedings of the First International Conference on Innovative Computing, Information and Control - Volume 3, ed. Washington, DC, USA: IEEE Computer Society, 2006, pp. 217-219.