Hindawi Publishing Corporation
Journal of Sensors
Volume 2016, Article ID 8085407, 8 pages
http://dx.doi.org/10.1155/2016/8085407

Research Article

Light-Weight and Versatile Monitor for a Self-Adaptive Software Framework for IoT Systems

Young-Joo Kim,[1] Jong-Soo Seok,[1] YungJoon Jung,[1] and Ok-Kyoon Ha[2]

[1] *Electronics and Telecommunications Research Institute, Embedded SW Platform Research Section, Deajeon 34129, Republic of Korea*
[2] *Department of Aeronautics & Software Engineering, Kyungwoon University, Gumi 39160, Republic of Korea*

Correspondence should be addressed to Ok-Kyoon Ha; okha@ikw.ac.kr

Received 15 April 2016; Accepted 7 November 2016

Academic Editor: Antonio Fernández-Caballero

Today, various Internet of Things (IoT) devices and applications are being developed. Such IoT devices have different hardware (HW) and software (SW) capabilities; therefore, most applications require customization when IoT devices are changed or new applications are created. However, the applications executed on these devices are not optimized for power and performance because IoT device systems do not provide suitable static and dynamic information about fast-changing system resources and applications. Therefore, this paper proposes a light-weight and versatile monitor for a self-adaptive software framework to automatically control system resources according to the system status. The monitor helps running applications guarantee low power consumption and high performance for an optimal environment. The proposed monitor has two components: a monitoring component, which provides real-time static and dynamic information about system resources and applications, and a controlling component, which supports real-time control of system resources. For the experimental verification, we created a video transport system based on IoT devices and measured the CPU utilization by dynamic voltage and frequency scaling (DVFS) for the monitor. The results demonstrate that, for up to 50 monitored processes, the monitor shows an average CPU utilization of approximately 4% in the three DVFS modes and demonstrates maximum optimization in the Performance mode of DVFS.

1. Introduction

Rapid growth in information and communications technology (ICT) has resulted in the development of various types of Internet of Things (IoT) devices and applications for the industry, home, and other sectors. However, such IoT devices have different hardware (HW) and software (SW) capabilities. The HW capability is mainly influenced by the number of CPU cores or the CPU clock speed. Further, battery capacity is important because IoT devices do not generally use external power. Therefore, many researchers have considered the relation between performance and power. For example, if a system allocates many CPU cores to a program, the program has high performance but its power consumption is not efficient. The SW capability of an IoT device is mainly determined by the number of running applications because running applications can affect system performance, power, and so forth. Hence, these running applications must be customized when the devices are changed or new applications are executed on the device. However, the applications are not optimized with respect to performance, power, and so forth because IoT device systems do not suitably provide static/dynamic information for fast-changing system resources and applications.

Therefore, in this manuscript, we propose a light-weight and versatile monitor for a self-adaptive software framework; the proposed monitor and the framework can automatically control system resources according to the system status. The proposed monitor can function with small-scale systems (e.g., IoT devices and embedded devices) and large-scale systems (e.g., PC and rich systems); the monitor has a light-weight design. In order to support the self-adaptive software framework, the monitor helps running applications to guarantee low power and high performance, thus creating an optimal environment. The proposed monitor has two components: a monitoring component and a controlling component. The monitoring component provides static and dynamic information about the systems and applications

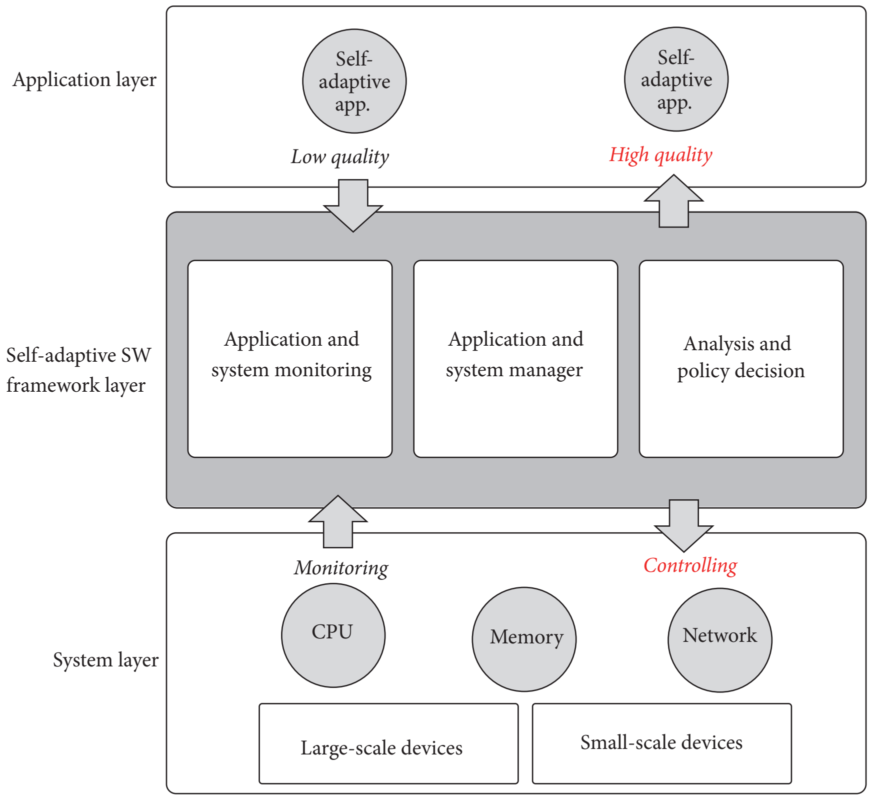

FIGURE 1: Concept of self-adaptive software framework.

in real-time. Static information is meaningful data that are already fixed in systems and applications, and dynamic information is meaningful data that change during the execution of systems and applications. The controlling component helps in the control of system resources (e.g., CPU, memory, network, etc.) in real-time. The control functions defined are CPU on/off, CPU frequency control, and network bandwidth control. For the experimental verification of the proposed monitor, we created a video transport system based on IoT devices and measured the CPU utilization for the monitor. The results showed that, for up to 50 monitored processes, the average CPU utilization of the monitor is approximately 4% in the three DVFS modes. Further, we observed that the monitor shows maximum optimization in the performance DVFS.

The remainder of this manuscript is structured as follows: In Section 2, we introduce the concept of a self-adaptive software framework. In Section 3, we describe the lightweight and versatile monitor for the self-adaptive software framework. In Section 4, we demonstrate the potential of the monitor through a self-developed QoS guarantee system. Finally, we state our conclusion and outline our directions for future research.

2. Self-Adaptive Software Framework

A self-adaptive software framework [1, 2], which is a middleware to guarantee optimal QoS for each application executing in a system, can manage and control running applications during their life cycle in a real-time and dynamic manner. In order to manage and control these applications, the framework provides monitoring functions. These functions are typically called adaptive applications. These adaptive applications include at least two modules: a QoS generator such as heartbeat [3] and a performance container with various algorithms. The QoS generator is inserted into a monitoring point of an application before the execution of the application; then, during the execution of the application, the QoS generator periodically reports the QoS for the application. The performance container uses one of two approaches: the first approach is to change the input parameters that influence the performance of the application, and the second approach is to create one or more algorithms that can change according to the performance. The self-adaptive software framework examines the reported QoS and then controls the adaptive applications. The framework uses the static and dynamic information about the application and the system resources to adjust the optimal QoS performance.

Generally, the self-adaptive software framework is composed of modules that monitor application or system information and control system resources. Figure 1 shows the overall structure of the self-adaptive software framework. As shown in this figure, the self-adaptive software framework layer is located between the application layer and the system layer. The framework consists of three modules: *an application and system monitoring* module, *an application and system*

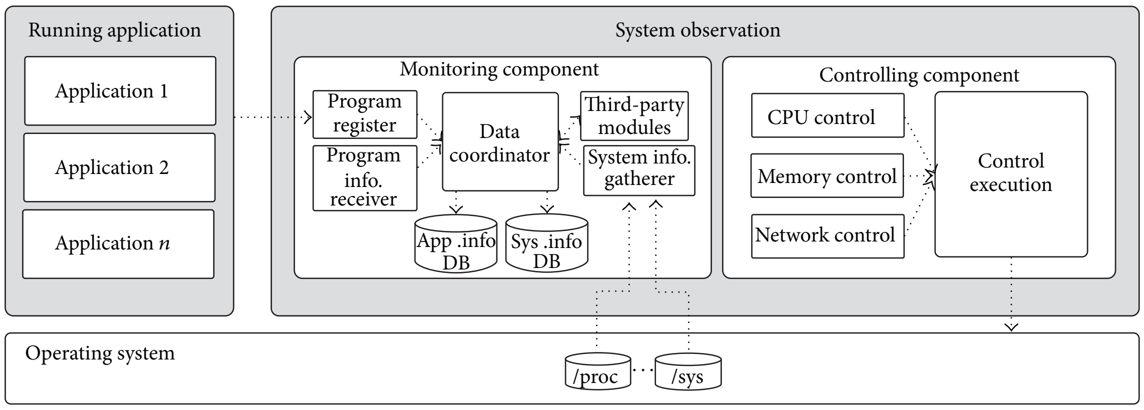

Figure 2: Block diagram of the light-weight and versatile monitor consisting of monitoring component.

manager module, and *an analysis and policy decision* module. In the application layer, when a self-adaptive application is running with low quality and the user requirement for the application is high quality, it is not easy for a general operating system to change the low-quality service to a high-quality service. However, the self-adaptive software framework can adjust the service quality by using the three modules. The application and system monitoring module gathers information from the system layer through monitoring, the application and system manager module provides the gathered information to the other modules, and the analysis and policy decision module controls the system resources or the flow of the application with the assistance of the manager. Our proposed monitor corresponds to the application and system monitoring module and the application and system manager module.

3. Self-Adaptive System Observation

In this manuscript, we present the *light-weight and versatile monitor*, which consists of the monitoring component and the controlling component. First, the monitor gathers static/dynamic information about the applications and systems; this information is required for the self-adaptive decision. Then, this information is provided to external modules and external devices. The proposed monitor is light-weight with respect to CPU utilization; therefore, it can be ported to diverse IoT devices such as embedded systems.

Figure 2 shows the block diagram of the light-weight and versatile monitor. In this figure, when applications are executed on a system, the monitor observes the status of these applications and the system in real-time and records this information in the *"application information DB"* and *"system information DB,"* respectively. The recorded information will be utilized to determine the optimization of the application to obtain high performance and low power consumption.

3.1. Monitoring Component. The monitoring component consists of five modules: *program register*, *program information receiver*, *system information gatherer*, *data coordinator*, and *third-party modules*. The program register and the program information receiver are responsible for collecting information about the application. During the execution of an application, two modules are connected to applications in TCP/IP. For example, when an adaptive application executes on a system, the application is registered in the light-weight and versatile monitor of the self-adaptive software framework by the program register; then, the program information receiver gathers static/dynamic information about the registered application. Table 1 provides a summary of the static/dynamic information for a running application.

Table 1: Application information (static/dynamic).

Static information	Dynamic information
Program ID	Allocated core
Program name	The number of threads
Program path	Thread list
Max QoS	Program status
Min QoS	Program time
Target QoS	Program/memory Utilization
Application sampling time	Network information (Tx, Rx)
Log file	Heartbeat rate

The system information gatherer collects static/dynamic information about the system resources in real-time. Further, the third-party modules receive information such as power, program characteristics, and internal kernel information from external modules or external devices. For example, system power and application power must be measured by external power equipment, and the measured power is transferred to the third-party modules of the self-adaptive system observation monitor. System static information represents the fixed values corresponding to the HW resources (e.g., CPU, memory, and network), and it is configured only once when a system functions. Generally, system static information has a unique value for the system when HW specifics remain unchanged. The value is determined by one-time data collection. System dynamic information represents values that change according to the system status. Most of these values can vary according to the system overhead, and they are updated periodically during the setup for information

Table 2: System information (static/dynamic).

Static information	Dynamic information
The number of cores	Core activity status
Max freq. of core	Core current freq.
Min freq. of core	The number of threads
Core MIPS	The number of processes
Available frequency	Core utilization
System memory	System memory
Network interface	Network packet (Tx, Rx)
DVFS	Power (CPU, GPU, and memory)

collection. The system static information shown in Table 2 includes CPU core, CPU frequency, memory, network, and DVFS. The system static information is the fixed information of the HW. If the HW capability is changed, the system static information is also updated. For this information, a data structure corresponding to each HW system is maintained separately. The system static information is updated only once when the light-weight and versatile monitor is executed. The system dynamic information shown in Table 2 includes the current state of the CPU core, CPU utilization, memory, network, and power. The dynamic information can be changed according to the current state of the system; further, users can change the resource values of the system, and thus, dynamic information is changed in real-time.

The data coordinator reorganizes the information collected from the program information receiver, the system information gatherer, and the third-party modules; then, this information is saved in the application information DB and the system information DB. The dynamic information about the application and systems is categorized into two data structures: instant data and calculation data. The instant data (e.g., CPU activity) can be used immediately in other modules, whereas the calculation data (e.g., utilization) cannot be used immediately owing to the need for additional operations. The static information about the applications and systems has a single data structure, which corresponds to instant data. These data are classified in real-time as application, system, or third-party data.

3.2. Controlling Component. The controlling component consists of four modules: *cpu control*, *memory control*, *network control*, and *control execution*. This component provides an environment to control system resources such as CPU, memory, and network. The various parameters of these system resources are as follows:

(i) CPU: core on/off, core frequency, thread affinity, DVFS (dynamic voltage frequency scaling) [4]

(ii) Memory: cache drop, minimum memory set

(iii) Network: bandwidth, packet drop

Control execution controls the CPU and memory by interacting with the proc file system. Control execution controls the network by using network control commands

Table 3: Monitoring interfaces and controlling interfaces.

Monitoring interface	Controlling interface
The number of cores	Core activity status
Max freq. of core	Core current freq.
Min freq. of core	The number of threads
Core MIPS	The number of processes
Available frequency	Core utilization
System memory	System memory
Network interface	Network packet

(e.g., tc command) or kernel model programs (e.g., network stack). Further, the controlling component provides monitoring interfaces and controlling interfaces such as libraries. Table 3 shows the monitoring interfaces and controlling interfaces. Thus, self-adaptive system observation uses the static/dynamic information about the applications and systems to enable optimal execution of applications.

4. Experimental Verification

In this section, we present the experimental verification of the proposed monitor. We introduce a video transport system based on IoT devices; the monitor is applied to this system and the experimental results for the proposed light-weight monitor are presented.

4.1. Implementation. Figure 3 shows a video transport system with a self-adaptive SW framework including the light-weight and versatile monitor. The system consists of a three-tier structure (IoT devices ↔ set-top box (STB) ↔ host system). The IoT device that performs video capture and encode consists of an Intel Edison Board [5–8] with CAM; the STB, which performs video streaming, consists of an Embedded Board with Exynos 5422 [9] and contains the proposed monitor and a self-adaptive policy manager. This manuscript does not focus on the policy manager. The host system, which includes a user interface, consists of a mobile device with wireless communication such as Wi-Fi. The functioning of this system can be described as follows: video sources are generated from the Intel Edison Board with CAM in real-time; the generated sources are transferred to the STB through Wi-Fi; a video streaming server on the STB receives these video sources and decodes them in real-time; finally, the decoded sources are transferred to the mobile device through Wi-Fi.

In Figure 3, if a self-adaptive SW framework is not present in the STB, the STB cannot guarantee the transfer of a high bitrate video generated in the Intel Edison Board to the host system because the STB may convert the video into a low bitrate video owing to overhead. In typical systems, this behavior is observed. However, even in the presence of a self-adaptive SW framework in the STB, the high bitrate generated from the Intel Edison Board may not reach the IoT devices and the host system. For example, if the number of video sources increases steadily, it is not easy for a typical video transport system to provide the high bitrate videos generated

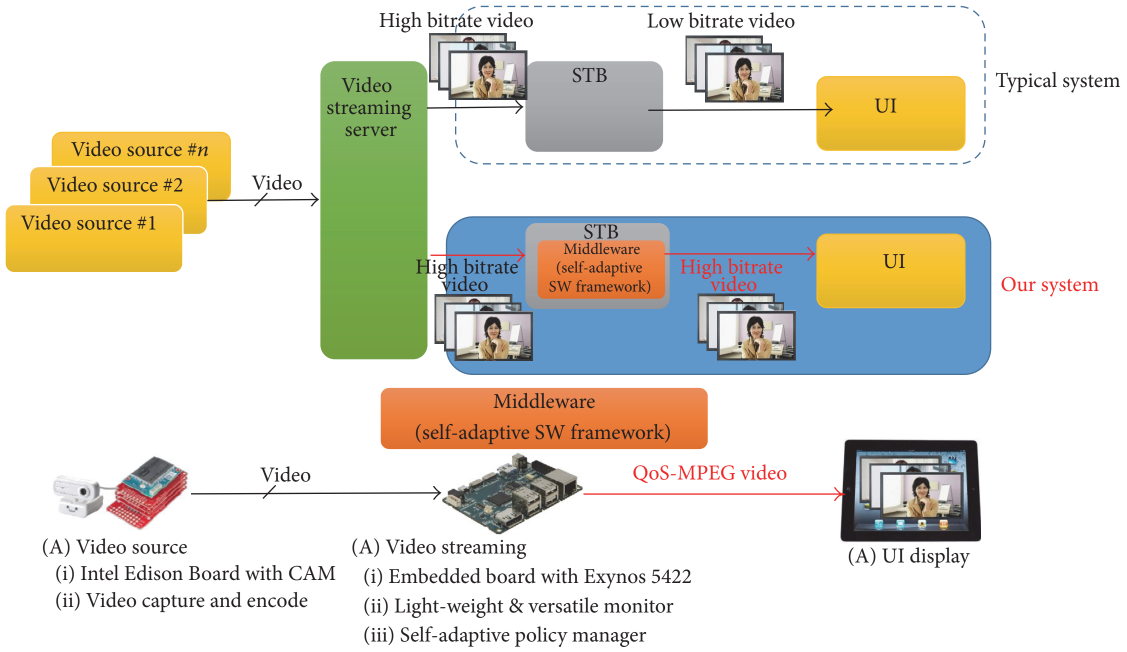

FIGURE 3: Video transport system with self-adaptive SW framework including light-weight and versatile monitor.

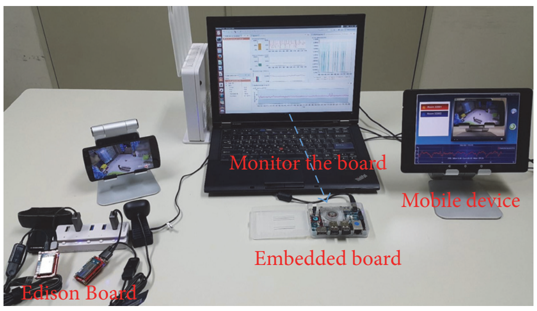

FIGURE 4: The screenshot of the implementation of the system in Figure 3.

in the Edison Board to the mobile device. The reason is that system resources (e.g., the number of cores, memory, and network bandwidth) allocated in the streaming server are insufficient; hence, the STB assigns low performance to running applications in a fair manner. However, the proposed video transport system can provide the high bitrate videos to the mobile device because our system has a middleware—the self-adaptive SW framework including the light-weight and versatile monitor. This middleware can handle system resources by using the application and system information collected by the monitor. Therefore, the proposed system always maintains the QoS defined by the user even in the case of many overheads.

Figure 4 demonstrates a system based on the implementation of Figure 3. In Figure 4, two Edison boards capture and encode video images. The embedded board receives these images and processes them. The light-weight and versatile monitor is ported to the board. The tool that runs in the notebook shows the information collected by the monitor in real-time; this information represents the static/dynamic information about the video streaming server and embedded board. The mobile device is provided with the original images of the target systems without loss of images. The tool and UI (user interface) show a regular QoS, indicated by the red color, in the graph.

4.2. Results and Analysis. In order to verify the light-weight characteristic of the proposed monitor, which functions on the Exynos 5422 embedded system, we measure the CPU utilization of the monitor by using our homebrew experimental application. The application automatically creates processes according to the input values and then initiates the execution of the processes. The input values are 1, 7, 15, 30, 50, 80, 100, 130, 170, and 200. CPU utilization is measured by DVFS governor (Interactive, Performance, and Ondemand). Figures 5, 6, and 7 show the CPU utilization corresponding to the number of running processes in the monitor for Interactive, Performance, and Ondemand modes of DVFS, respectively. The top graphs in each figure show the results corresponding to the conversion of CPU utilization into a percentage for 200 running processes. The CPU utilization is measured 500 times while the processes are executing. The bottom graphs in these figures show the average and error range of CPU utilization according to the number of monitored processes.

For the Interactive mode, the measured CPU utilization is shown as a percentage in the top graph of Figure 5. As shown in the dotted red rectangle of this graph, the percentage is approximately 20% for up to 50 processes. However, more than 80 processes exceed 30%, so that the suggested monitor can affect an embedded system (e.g., Exynos 5422) due to

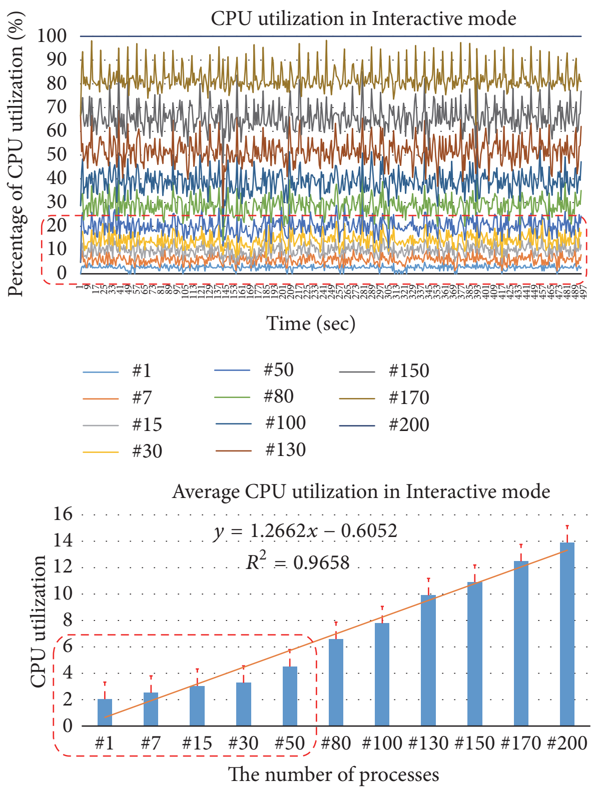

FIGURE 5: CPU utilization in Interactive mode.

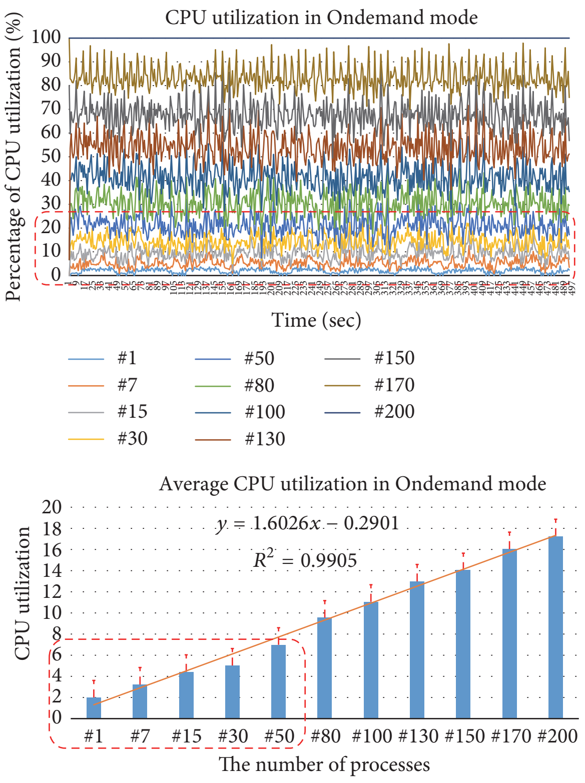

FIGURE 7: CPU utilization in Ondemand mode.

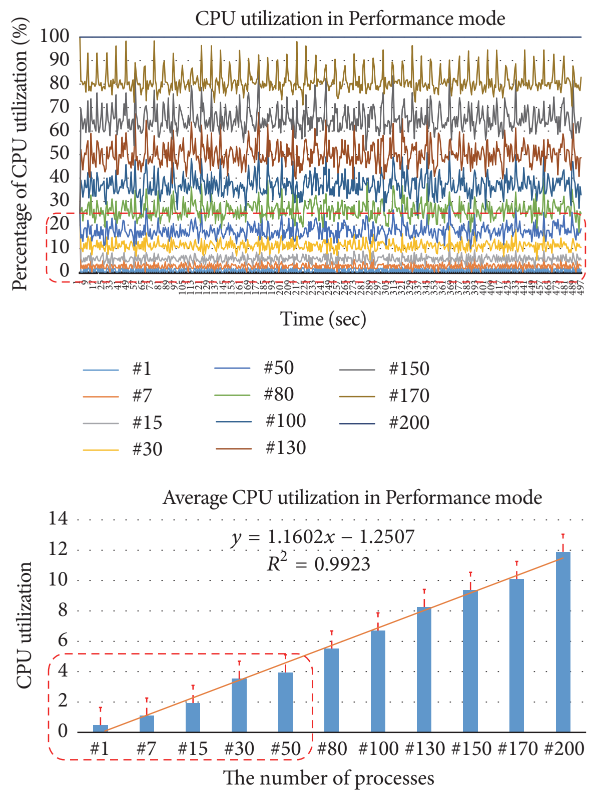

FIGURE 6: CPU utilization in Performance mode.

its monitoring overheads. The bottom graph of Figure 5 is the detailed experimental results for the aforementioned fact. The graph shows that the CPU utilization coefficient of the monitor working in the Interactive mode is 1.2662 ($y = 1.2662x - 0.6052$). The coefficient is calculated by the regression analysis result for the measured CPU utilization based on the number of processes. This result is correct because R^2, the reliability of the measured data, is 0.9658. In the bottom graph of Figure 5, the CPU utilization is approximately 4% for up to 50 processes as shown in the dotted red rectangle. Therefore, we can state that the proposed monitor has the light-weight characteristic because the monitor based on a self-adaptive SW framework is not necessary for a large number of processes (greater than 50). The results in Figures 6 and 7 are similar to the result of Figure 5. The CPU utilization coefficient is 1.1602 ($y = 1.1602x - 1.2507$) and R^2 is 0.9923. The CPU utilization is also approximately 4% for up to 50 processes. The CPU utilization coefficient in Figure 7 is 1.6026 ($y = 1.6026x - 0.2901$) and R^2 is 0.9905. However, the CPU utilization is approximately 5% for up to 50 processes. Through experimental results, the proposed monitor shows the best CPU utilization in the Performance mode and the second-best CPU utilization in the Interactive mode. The Ondemand mode has more overhead than the Performance and Interactive modes because the mode controls CPU frequency by checking CPU utilization periodically.

Figure 8 shows the CPU utilization corresponding to the number of processes (1, 7, 15, and 50) for the Interactive, Performance, and Ondemand modes. In all four graphs,

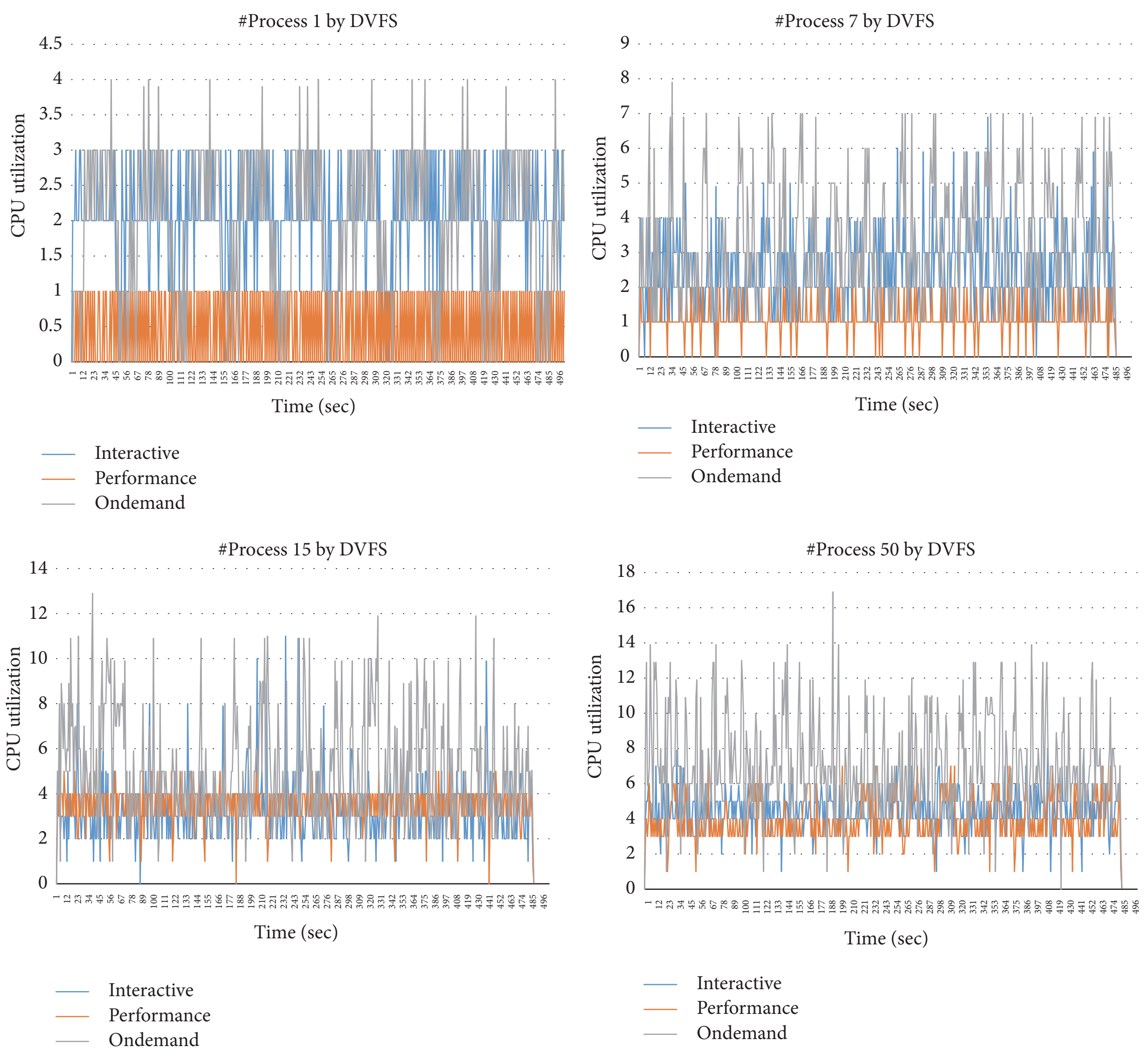

Figure 8: CPU utilization by DVFS (number of processes: 1, 7, 15, and 50).

the CPU utilization of the Ondemand mode fluctuates very widely and the utilization is also higher than that of Interactive and Performance modes. As the number of processes increases, the suggested monitor has much more overhead in the Ondemand mode. (e.g., "#Process 50 by DVFS" graph of Figure 8 indicates that the CPU utilization of the Ondemand mode is over 2 times that of the Interactive and Performance modes). That is, the monitor efficiently works in the Interactive and Performance modes. Overall, the Ondemand mode has greater overhead than the Interactive and Performance modes. Figure 9 explains the aforementioned logical reason well. In the Performance, the CPU utilization is 0.49 when the number of process is 1. In this case, the suggested monitor has the most performance. The CPU utilization is 3.95 when the number of processes is 50. The increase rate of the utilization is about 8.06. The increase gap is the largest among the CPU utilization of DVFS governor. In the Interactive mode, the CPU utilization is 2.04 when the number of process is 1 and the CPU utilization is 4.50 when the number of processes is 50. The increase rate of the utilization is about 2.21. The increase gap is the smallest, so that the suggested monitor has the most stable performance. In the Ondemand mode, the CPU utilization is 1.99 when the number of process is 1, and the CPU utilization is 7.03 when the number of processes is 50. In this case, the suggested monitor has the worst performance. The increase rate of the utilization is about 3.53. Therefore, the monitor has the light-weight overhead in the Performance mode and the monitor has the most stable overhead in the Interactive mode.

5. Conclusion

In this manuscript, we propose a light-weight and versatile monitor that can be used in a self-adaptive SW framework. This monitor can be used for large-scale devices (gateway and STB) and small-scale devices (Intel Edison Board and

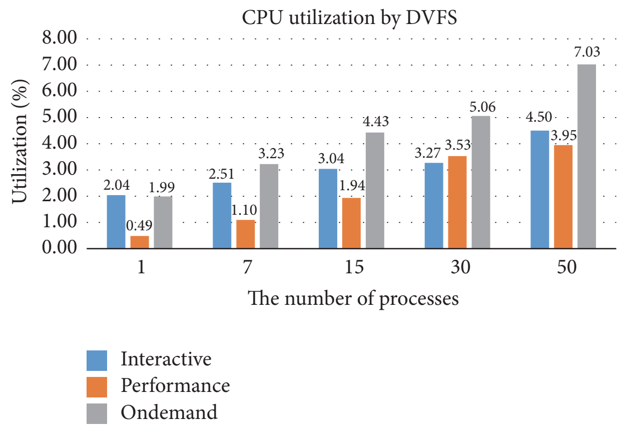

Figure 9: Average CPU utilization by number of processes.

IoT devices). The proposed monitor provides static/dynamic information about system resources and running applications to users in real-time; thus, the monitor helps a self-adaptive policy manager of the self-adaptive SW framework to optimally control system resources or applications. From our experiments, we determined that the monitor shows maximum optimization in the Performance mode of DVFS. The monitor shows 3.95% CPU utilization for up to 50 monitored processes. In future work, we will apply the proposed monitor to various hardware platforms and will demonstrate the superiority of the proposed monitor.

Disclosure

This paper is a revised and expanded version of the paper entitled "Design of Self-Adaptive System Observation over Internet of Things" presented at International Conference on Control and Automation (CA 2015), November 25, 2015, Jeju, Korea.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by ICT R&D program of MSIP/IITP (B0101-16-0661, the Research and Development of the Self-adaptive Software Framework for various IoT Devices) and also was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2014R1A1A2060082).

References

[1] C. Bolchini, M. Carminati, A. Miele, and E. Quintarelli, "A framework to model self-adaptive computing systems," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '13)*, pp. 71–78, Torino, Italy, June 2013.

[2] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "A generalized software framework for accurate and efficient management of performance goals," in *Proceedings of the 13th International Conference on Embedded Software (EMSOFT '13)*, pp. 1–10, IEEE, October 2013.

[3] H. Hoffmann, J. Eastep, and M. D. Santambrogio, "Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments," in *Proceedings of the International Conference on Autonomic Computing (ICAC '10)*, pp. 79–88, June 2010.

[4] H. Hong, J. Lim, H. Lim, and S. Kang, "New thermal-aware voltage Island formation for 3D many-core processors," *ETRI Journal*, vol. 37, no. 1, pp. 118–127, 2015.

[5] https://www.sparkfun.com/categories/272.

[6] http://www.intel.com/content/www/us/en/do-it-yourself/edison .html.

[7] Intel Edison Boards, Intel Edison Breakout Board Hardware Guide, 2015 .

[8] Intel Edison Boards, Intel Edison Board Support Package—User Guide, Revision 001, 2014.

[9] http://www.samsung.com/semiconductor/minisite/Exynos/w/solution/mobile_ap/5422/.