A Power-Saving Technique for the OSGi Platform

Kuo-Yi CHEN(a), Chin-Yang LIN†, Tien-Yan MA†, Nonmembers, and Ting-Wei HOU†, Member

SUMMARY With more digital home appliances and network devices having OSGi as the software management platform, the power-saving capability of the OSGi platform has become a critical issue. This paper is aimed at improving the power-efficiency of the OSGi platform, i.e. reducing the energy consumption with minimum performance degradation. The key to this study is an efficient power-saving technique which exploits the runtime information already available in a Java virtual machine (JVM), the base software of the OSGi platform, to best determine the timing of performing DVFS (Dynamic Voltage and Frequency Scaling). This, technically, involves a phase detection scheme that identifies the memory phase of the OSGi-enabled device/server in a correct and almost effortless way. The overhead of the power-saving procedure is thus minimized, and the system performance is well maintained. We have implemented and evaluated the proposed power-saving approach on an OSGi server, where the Apache Felix OSGi implementation and the DaCapo benchmarks were applied. The results show that this approach can achieve real power-efficiency for the OSGi platform, in which the power consumption is significantly reduced and the performance remains highly competitive, compared with the other power-saving techniques.

key words: energy-efficient, OSGi, Java virtual machine, DVFS

1. Introduction

With the fast growth of the Internet, the use of the Internet has not been limited to the office (work environment). Instead, the home has already replaced the office to become the major place for users to use the Internet. This significant change has led to the wide deployment of home services/applications and devices, such as multimedia on demand, mobile remote control and smart home devices.

In order to provide friendly user interfaces and achieve easy management for multiple services, applications and devices, an information gateway is necessary. The Open Service Gateway initiative (OSGi) [1] platform is one of the widely used standards for such a gateway. The OSGi defines a set of standards that involves the protocols between heterogeneous services/applications and devices. As a result, different resources can communicate to each other and be well managed under the OSGi platform.

In an OSGi network, the OSGi server, also written as Open Service Gateway (OSG), is the most important component. This server normally connects to service providers over a wide area network (WAN), and connects to domestic devices over a local area network (LAN). Service providers can package their applications as bundles. These bundles can then be transferred to, and executed on, the OSGi server as requested. It is worth noting that an OSGi server is usually nonstop, i.e. a long-running server providing continuing services. Therefore, it leads to the energy wastage problem, i.e. the power-saving issue.

The Dynamic Voltage and Frequency Scaling (DVFS) technique is a common methodology for reducing the energy wastage of the server-based systems. In the simplest form, DVFS is based on dynamically changing the voltage/frequency of processors to reduce the power consumption [2]. Several related studies have been presented in the literature [3]–[7]. These studies are classified into two major groups: (1) profiling and (2) performance monitors.

The basic idea of the profiling-based approach is to analyze the runtime behaviors of the applications first, and then to adjust the frequencies of the processors based on the information gathered in the analysis (profiling) phase [8]. This method is intuitive, but it requires extra costs, such as code analyses and insertion of special instructions. Thus, the profiling approach is rarely applied to the systems which require quick response.

The alternative is to use the DVFS technique based on the information of the hardware performance monitors (PM) [9]. By reading the PM at runtime, different phases of an application, such as memory phase and execution phase, can be identified and observed during its execution. For example, the memory phase often means a higher memory stall cycles, thus implying the timing to lower the processor frequency for power-saving. However, this PM-based approach comes with two limitations. First, the result of the phase identification is always one step behind, since it can only be obtained after the program behavior information appears in the PM. Second, the actual end time of a given phase cannot be precisely determined/computed.

In this paper, we focus on how to improve the power-saving of the OSGi platform, especially for OSGi-enabled devices which are used as long-running servers. Unlike the traditional methods (i.e. profiling and PM), our approach reduces the energy wastage of the OSGi platform by exploiting the runtime information which is already available in a Java virtual machine (JVM) [10], the base software of the OSGi platform, where the runtime information serves as the basis to effectively guide the way of using the DVFS for greater power efficiency.

Manuscript received June 23, 2011.
Manuscript revised November 3, 2011.
†The authors are with the Department of Engineering Science, National Cheng-Kung University, Tainan City 701, Taiwan R.O.C.
‡This research was supported by the National Science Council of R.O.C. under contract no. NSC 99-2221-E-006-123 and 101-2220-E-001-001.
a) E-mail: kuoyichen@nc.es.ncku.edu.tw
DOI: 10.1587/transinf.95.D.1417

Copyright © 2012 The Institute of Electronics, Information and Communication Engineers
Compared with the other power-saving techniques, the proposed approach offers two benefits. First, no profiling is required, avoiding extra costs (as mentioned above). Secondly, the runtime information of a JVM, actually correlated with the application phases, can be obtained before the execution of the applications, so that the problems of the PM-based method can be avoided.

To demonstrate the feasibility of our novel approach, a series of experimental steps are performed as follows. First, the software components of the OSGi platform are studied, including the base software of the OSGi platform, i.e. the JVM. Then, the runtime information of the JVM is analyzed to identify the particular phases of OSGi applications. Finally, according to the correlation between the OSGi application phases and the analyzed runtime information, the power-saving approach is proposed and is evaluated based on an OSGi implementation [1], [16] and several benchmark programs [11]. The experimental results demonstrate the effectiveness of the proposed approach.

2. The Power-Saving Opportunity for OSGi Platform

The Open Service Gateway initiative (OSGi) [1] defines a standardized service platform for developing and deploying service-oriented applications. In OSGi, a service is described as a Java interface which is normally packaged along with its implementations into a software component called bundle. Each bundle can be dynamically and remotely installed, updated and removed. Such a dynamics provides a point-to-point transmission solution between service providers and local devices.

Figure 1 illustrates a typical OSGi infrastructure. It is easy to see that the OSGi server/gateway plays the key role in this infrastructure. The user at home can select a service to install on the OSGi server. For service providers, their OSGi applications (i.e. bundles), with the help of the operators, can be remotely deployed on the OSGi server. In this case, the OSGi server serves as a residential gateway which could be a set-top box (STB), ADSL modems, cable modems, etc.

It should be noted that, to ensure the platform independence, the OSGi platform employs a Java technology-based software framework. Each OSGi platform must be powered by a Java virtual machine (JVM). As shown in Fig. 2, the basic architecture of an OSGi-enabled device/server contains an OSGi framework that sits on top of a JVM. All the bundles (Java applications) running on the OSGi device are supposed to be run on the JVM. This means that all the instructions (i.e. Java bytecodes) have to be interpreted by a single JVM instance, and then be executed on the hardware.

Based on this fact, we have a hypothesis that there exists some relationship between the JVM runtime information and the phases of the applications. In this case, assume the memory phase of the OSGi device could be correlated to some of the JVM runtime information. Then, it would be possible to identify the correct timing to lower the CPU frequencies while the device is running. That is, under this hypothesis, the runtime information of the JVM would provide an opportunity to achieve power-saving for the OSGi device.

The remainder of this paper will be dedicated to our methodology, including how to analyze the application behavior and how the JVM runtime information can be used to identify the application phases.

In addition, since our approach is based on observing the runtime information already available in a JVM and performing DVFS in advance, the problems caused by the traditional power-saving approaches can be avoided naturally. According to our experiments in this study, the proposed power-saving approach can appropriately change CPU frequencies to reduce the energy wastage of the OSGi platform while maintaining performance.

3. Methodology

This section mainly describes the JVM runtime information useful for our study, and how the phases of the OSGi device/application can be correlated to the runtime information.

The JVM runtime information refers to the data internally managed by a running JVM, including the internal data
structures and the resource details of the JVM. In this study, the exploited runtime information would be limited to the automatic memory management of the JVM, also known as garbage collection [12]. Garbage collection (GC), an integral part of a JVM, is designed for the “memory recycling” and is essentially related to the memory access behaviors of the applications. It normally takes up a considerable portion of the Java system workload [13].

In a garbage-collected system, the involved operations/instructions can typically be divided into two parts: mutator and collector. The mutator represents the thread of computation that allocates and modifies the objects in heap, which is usually referred to as the application thread. The collector, or garbage collector, corresponds to the thread of control in charge of reclaiming the unused objects (i.e., garbage) in heap.

Since the mutator is essentially application-dependent, it is difficult to be used to identify the memory phase of the OSGi device in a precise way. Although it might work by analyzing the mutator behaviors of the applications (running on the device) at runtime, it is costly and actually can lead to performance degradation.

In contrast, the collector would provide a better opportunity for efficiently determining the memory phase of the OSGi device at runtime. First, the garbage collector, as such, is dedicated to the memory analysis and optimization. Second, the collector and the mutator are typically mutually-exclusive (i.e., non-overlapping); that is, the GC phase (i.e., the execution period of GC) would not be interfered with by the mutator behavior. Also, the OSGi device, as mentioned earlier, is based on the architecture where all applications are executed on a single JVM instance.

Based on these facts, a reasonable hypothesis would be that the GC phase of an OSGi device implies the memory phase of the device. Technically, since the JVM can identify all the GC phases at runtime, the correct timing to lower the CPU frequencies of the device can be determined naturally, thus achieving the power-saving goal. Under this concept, the mutator phase would be referred to as the execution phase.

To further support this hypothesis, we conduct the following experiment, configured as described in Sect. 5. Two Java benchmarks, Sunflow and Eclipse, from the Da-capo benchmark suite [11] are used. A special measure called Stall Cycle Per Instruction (SCPI) [20] is employed (tracked) to analyze the phases of the applications (benchmarks). The SCPI is defined as follows:

\[
SCPI = \frac{\text{Mem\_Stall}}{\text{Instr\_Ret}}
\]

where Mem\_Stall and Instr\_Ret are both the hardware events which can be monitored by the performance monitors. Instr\_Ret stands for the number of instructions retired. Mem\_Stall represents the number of the stalled cycles, which reflects the time the CPU needs to wait for the required input data from memory.

In this experiment, a low value of SCPI is used to indicate an execution phase, since it implies a high CPU workload. On the other hand, a high value of SCPI is used to signify a memory phase, which normally implies the CPU waiting for memory requests and actually the timing to lower the CPU frequencies.

Figure 3 and Fig. 4, respectively, show the experimental results for the benchmarks Eclipse and Sunflow, where the SCPI values and the GC periods are plotted. Here, each GC period (or GC phase) stands for an engaged period of GC, and the disengaged periods of GC can be viewed as the mutation phases (periods).

It could be observed that the SCPI values of the mutation phases vary greatly and are very instable. That is because the workload in the mutation phase actually refers to the application code. In contrast, the SCPI values of the GC phases are relatively high and stable. That is because the GC phase is essentially responsible for analyzing and collecting unused objects in heap memory, thus resulting in a significant number of memory stall cycles (i.e., the memory cycles that the CPU needs to wait for the required data from memory).
phase).

Clearly, the results confirm our hypothesis that the GC phase of an OSGi device implies the memory phase of that device. In this study, a novel power-saving approach is developed according to this hypothesis, which is detailed in the next section.

4. The Power-Saving Approach

This section presents our power-saving approach. The approach is based on the above hypothesis which correlates the GC phase of an OSGi device/server with the memory phase of that device/server.

Under this hypothesis, the memory phase and the execution phase, respectively, refer to the GC period and the non-GC period. Therefore, the phase detection can be achieved by directly observing the engaged/disengaged periods of the garbage collector at runtime.

Since the GC related runtime information is naturally available to a JVM and is naturally mapped to the application phases, the phase detection overhead is minimized. The phase detection procedure can be carried out by modifying the JVM. For example, when a GC trigger occurs (i.e. memory phase), a procedure of lowering the CPU frequencies would be launched for reducing the power consumption. The power-saving algorithm is outlined in Fig. 5.

Each time a start/end event of the GC occurs, the frequencies of the available CPUs would be decreased/increased. Since the frequency adjustments are confined to the engaged periods of GC, the performance of the mutators (i.e. application code) would not be affected. In the algorithm, the frequency adjustment details, e.g. the available number of CPU and levels of frequency, hinge on the underlying hardware architecture. In addition, the effectiveness of the algorithm could be related to the selection of GC strategies/algorithms. For example, in a generational GC system with our power-saving approach enabled, a major/full GC would outperform a minor GC in terms of the power-saving. That is because a major GC often involves an analysis of a wide range of memory, thus resulting in a longer period of memory phase [14].

An overview of the entire power-saving approach is illustrated in Fig. 6. This, technically, involves (1) instrumenting the JVM of the OSGi platform for detecting the memory (or GC) phase, and (2) the procedure of lowering the CPU frequencies of the device/server.

The former corresponds to the timing informing module, which is added for observing and gathering the required GC information, e.g. the timing information about the start and the end of a GC phase. The latter is realized by the frequency adjustment module, which is embedded in the kernel for efficiently adjusting the CPU frequencies (as requested). All the requests are from the timing informing module. It is also worth noting that the frequency adjustment module includes in-line assembly codes that can directly adjust CPU frequencies by accessing some particular registers of the CPU [15]. Thus, compared with the method based on user-level calls, the way we adjust CPU frequencies is more efficient; the required CPU cycles for the frequency adjustment are minimized.

In addition, the power-saving approach brings two advantages, in comparison with the traditional methods. The first is the reliable phase identification, which, as described earlier, is simply because of the nature of the garbage collection (i.e. the memory phase). As we have seen in the previous experiment (Fig. 3 and Fig. 4), each of the GC phases always reflects a memory-intensive period. Therefore, our approach can avoid the inappropriate adjustment of the CPU frequencies. It should be noted that lowering the CPU frequencies in an execution phase (i.e. mutation phase) could degrade the overall system performance. For those methods based on profiling or PM, an inappropriate frequency adjustment may occur due to an inaccurate phase detection, thereby causing unnecessary performance degradation and energy wastage.

The second advantage of the proposed approach is that the overhead of the phase identification/detection is quite low. In fact, most of the power-saving approaches require the phase identification, and this usually has a negative impact on the system performance, e.g. the additional profiling

```
Power-Saving Algorithm:
    for each GC event E, issued by the JVM process
        if E = GC_START then
            for each available CPU Ca
                Ca := minimum frequency;
        else if E = GC_FINISH then
            for each available CPU Ca
                Ca := maximum frequency;
    end if
```

Fig. 5 The power-saving algorithm.

Fig. 6 The OSGi power-saving implementation.
work. Since the phase identification of our approach is entirely based on the runtime information already available in the JVM, the overhead can be minimized.

In short, the proposed power-saving technique could provide an opportunity to achieve the power-saving goal for the OSGi platform without the overheads caused by the traditional power-saving techniques.

5. Experimental results

In order to validate the power-efficiency of the proposed OSGi power-saving approach, the following experiments are conducted. All the measurements are based on a server with the Q6800 processor, a Quad-Core Intel CPU. The frequency of each core is allowed to be adjusted independently from 1.7 to 2.9 GHz. The Fedora Core 14 with kernel version 2.6.35 and Apache Felix 3.0.9 [16], an OSGi implementation that conforms to the OSGi R4 specifications [1], are used in the experiments.

In addition, a state-of-the-art Hotspot JVM, shipped with OpenJDK 1.7 [17], is used in the experiments. The GC configuration is entirely based on the default settings in the HotSpot JVM, in which the used garbage collector is a generational garbage collector. The garbage-collected space is divided into two generations, the young generation and the tenured generation [14]. The copying collector is used in the young generation. The young generation is optimized for those objects with a short lifetime. After several collections, the survived objects are moved to the tenured generation since these objects have a longer lifetime. In the tenured generation, the mark-sweep-compact collector is used to collect the garbage concurrently [18].

The generational garbage collector would be suitable for the OSGi platform. For the OSGi server, most of the temporary objects, created by the applications in response to immediate requests, would fall into the young generation and are collected frequently and effectively. The long-lived objects, often created by long running applications, would be considered as the tenured generation, and the infrequent collections on the tenured generation can avoid unnecessary GC operations, thus improving the server performance.

In the experiments, the measurement of CPU power consumption is critical. A widely used dynamic power measurement of CMOS circuits [19] is used in this study. The dynamic power consumption of processors \( P \) is expressed as follows.

\[
P = C \times V_{dd}^2 \times f
\]

where \( C \) is the effective switching capacitance, \( V \) is the CPU supply voltage, and \( f \) is the CPU execution frequency. With this equation, the power consumption of the processors could be (approximately) measured.

However, the power consumption cannot exactly indicate the power-efficiency, because the lower frequency settings (of CPUs) might significantly decrease the system performance. Hence, another measure called energy-delay product (EDP) is used here to evaluate the power-efficiency. The EDP value is defined as the product of the power consumption and the execution time squared [5]. Since this measure considers both energy and delay simultaneously, the EDP value can better reflect the power-efficiency. In general, a lower EDP value indicates a better power-efficiency.

Since there is no standard benchmark for evaluating the power-efficiency of the OSGi platform, the widely used Java benchmark suite, Dacapo [11], is used in our experiments. The latest version of Dacapo consists of fourteen Java benchmarks that cover a various range of applications, such as image tracing, document searching, database operation and web service processing. Therefore, the effectiveness (i.e. power-efficiency) of the OSGi power-saving approach can be examined under various types of workloads. In the implementation, each of the benchmarks is simply wrapped as an OSGi bundle and is designed to be launched when the bundle is started.

Furthermore, four power configurations are used to demonstrate the power-efficiency of our OSGi power-saving approach. First, the power configurations max-frequency and min-frequency, respectively, represent the maximum and minimum frequency. Both of them use the static (fixed) CPU frequency and can be mapped to the performance and power-saving governors in Linux.

The other two configurations, respectively, refer to the Linux power-saving governors, ondemand and conservative. These two power-saving governors would adjust frequencies based on observing the CPU workload. The use of ondemand governors normally switches to the highest frequency immediately when the CPU load is high. It can thus maintain the system performance well, while it leads to more energy wastage. On the other hand, the use of conservative governors increases frequency step by step. Thus, slight performance degradation and less energy wastage would be observed. These two Linux power-saving governors are popular and actually could be used to represent the PM-based power-saving approaches.

Figure 7, Fig. 8 and Fig. 9, respectively, show the experimental results in terms of performance, power-consumption and EDP values. The configuration, OSGi power-saving, represents the proposed power-saving approach. For each of these figures, the results of the last four configurations are normalized to the results of the first configuration (i.e. the max-frequency).

Figure 7 shows the performance comparison. It is unsurprising that the maximum frequency configuration delivers better overall performance than the other four configurations. In particular, for each of the benchmarks, the performance degradation caused by the OSGi power-saving configuration is less than two percent (compared to the results of the maximum frequency configuration). This suggests that our OSGi power-saving approach is highly competitive with the maximum frequency setting in terms of the performance.

In Fig.8, it is easy to see that the use of our OSGi power-saving approach leads to the lowest power consumption. In this case, the energy wastage is reduced by about
nineteen percent on average; the use of ondemand and conservative governors leads to an average reduction of eleven and fifteen percent respectively. It should be noted that the power consumption of min-frequency always delivers the lowest performance (as in Fig. 7) and actually results in the longest execution time, so the total power consumption for the min-frequency configuration is the largest.

Fig. 7  The performance comparison.

In Fig. 9, the EDP values are measured for comparing the power-efficiency. It can be seen that, for each of the benchmarks, the OSGi power-saving configuration delivers better power-efficiency than the other four configurations. In this case, the EDP values of the OSGi power-saving approach can be reduced by ten percent on average compared with the other four settings.

Table 1 displays the garbage collection times for the benchmarks; the results vary greatly among applications. That is mainly because each application has its own memory allocation behavior and can thus influence the garbage collection behavior. Coupled with the power-efficiency comparison in Fig. 9, the results also indicate that, for most of the benchmarks, a greater percentage of GC time results in a better power-saving. For example, the percentage of GC time for the benchmark H2 is the highest (19%), so the power-efficiency for H2 is relatively higher than most of the other benchmarks (i.e. a lower EDP value).

To further evaluate the practicality of our approach, a real OSGi application/bundle based on the Home Surveillance Application (HSA), also used in [33], is included as a benchmark. The main feature of HSA is to enable a remote monitoring capability for an OSGi-based home network, as illustrated in Fig. 10. The snapshot service, launched when the bundle starts, periodically obtains the snapshots/images captured by the cameras connected to the home network (via wireless LAN); all the captured and processed images are maintained in a local storage device. The surveillance service interacts with the snapshot service and relies on the OSGi Http Service to provide a web-based interface for the
client devices (e.g. a smart phone) to request the images around the house, thus achieving the remote monitoring capability.

Figure 11 shows the power-efficiency comparison for HSA under different number of clients (1, 2, 4 and 8). The number of clients reflects the maximum number of simultaneous requests to the surveillance service. The results show that the OSGi power-saving configuration always leads to the lowest EDP values, especially for higher number of clients. That is because the more the number of requests, the higher the probability of triggering GC would be, hence benefiting more from the proposed power-saving approach.

The power-efficiency for HSA is also evaluated in a long running mode, as shown in Fig. 12, where the EDP values are measured using four clients under different periods of running time (1, 2, 4 and 8 hours). It can be seen that the OSGi power-saving configuration still outperforms the other four configurations, and that there is no clear difference among the results under different periods of running time. It should be noted that the effectiveness of the proposed approach actually depends on the application behavior; the power saving gain normally increases while the applications on the server allocate memory. In the extreme case where applications work without allocating memory, the OSGi server may not benefit from the proposed power-saving approach.

Moreover, an experiment is conducted to evaluate the effectiveness of using different GC algorithms, where an incremental GC, shipped with the HotSpot JVM, is added for comparison with the default generational GC. The incremental GC here refers to the Concurrent Mark-Sweep GC used in an incremental mode. The results, as shown in Fig. 13, are measured under different number of clients. It can be seen that the default generational GC outperforms the incremental GC in terms of the power-efficiency (i.e. lower EDP values). That is because the incremental GC performs the tenured generation collection by dividing the work into discrete phases, which normally results in low pause times but leads to longer execution times, thus having higher EDP values.

Overall, our measurements demonstrate that an OSGi device/server can significantly benefit from the novel power-saving approach in terms of power consumption and power-
efficiency, and particularly the performance remains highly competitive. The results also confirm our hypothesis that the GC phase of an OSGi device/server directly implies the memory phase of the device/server.

6. Related Work

As hardware components are more power-hungry than ever, the power consumption of long-running devices/servers has become an important issue. Recent studies have demonstrated that the energy wastage can be retrenched by operating hardware components with different power levels. Moreover, due to the significant energy demands of CPUs in server systems, many studies are aimed at reducing the power consumption of CPUs. Weiser et al suggested using the Dynamic Voltage Scaling (DVS) to save the energy wastage of CPUs [20]. More related studies were based on exploring the performance of DVFS techniques in both general-purpose and embedded systems [21]–[24].

However, few studies really focused on the power consumption issue of the long-running server based on OSGi. Most of the related studies are targeted at reducing the energy consumption for the devices connected to the server instead of for the server (system). For example, Hlacavs et al presented a distributed approach to achieve energy efficiency for future home environments/networks based on the resource sharing [25]. In [26], a home energy management system, based on a residential OSGi Gateway, is developed for reducing home energy consumption. Tompros et al proposed a network architecture that relies on an intelligent residential gateway to enable the programming of the energy consumption for the home devices [27].

In view of this, this study first attempts to explore the power-saving opportunity for the OSGi platform, and then presents a power-saving technique based on exploiting the JVM runtime information and the DVFS technique [2]. Several key findings are also revealed through this study. First, the hypothesis that the GC phase of the OSGi platform indicates the memory phase is confirmed (by the experimental results presented here). Second, the runtime behavior of the OSGi device/server can be simply observed by the underlying Java runtime, thereby minimizing the phase identification overhead, when compared with the other power-saving techniques.

The energy efficiency topic, in fact, has been investigated in many other aspects. For example, Jun shirako et al. [32] proposed a static compiler control scheme to reduce the power consumption of a multi-core processor without profiling. It involves a voltage/frequency control for processors and a power supply cut-off function for unnecessary processors. In [30], a DVFS-based power-saving approach is developed for the Android system, where a technique based on memory access rate and critical CPU speed is used to predict/suggest the suitable frequency at runtime. Since the memory access rate is obtained by performance monitors (PM), the problems of the PM-based approach, mentioned in Sect.1, are unavoidable. In contrast, our approach can avoid the problems because it uses the information already available at runtime.

There are also some studies that concentrate on the energy efficiency for Java-based execution environments. For example, the study from Shiwen Hu and Lizy K. John [31] aims at investigating the processor energy consumption on virtual execution environments (e.g. JVM) through exploring the impact of using optimized Just-in-time compiler (JIT) as well as garbage collector. Moreover, Paul Griffin et al. proposed a hybrid garbage collection scheme to appropriately lessen the number of GC triggers (i.e. the number of memory accesses), thereby resulting in a lower energy consumption [28]. In [29], Guilin Chen et al. investigated how to reduce the energy consumption by tuning the garbage collector in a multibanked memory architecture. Both of them [28], [29] achieve the energy efficiency by changing the policy of garbage collection in some way. In contrast, our approach simply exploits the GC triggering information and does not change the behavior of garbage collector.

7. Conclusions

In this paper, a novel power-saving approach for the OSGi platform is proposed. The core of this work is a key observation that correlates the memory phase of the OSGi device/server with the GC phase of the JVM in that device/server. Based on this, we develop an OSGi power-saving technique which carefully exploits the runtime information already available in the JVM to determine the appropriate timing (e.g. memory phase) of applying the DVFS (i.e. CPU frequency adjustment), thereby efficiently reducing the energy consumption for the OSGi platform. To the best of our knowledge, this is the first study focusing on applying JVM runtime information to the power-saving issue for the OSGi platform.

The proposed approach has been implemented and evaluated on an OSGi server. The results show that this approach provides an opportunity to achieve real power-efficiency for the OSGi platform, in which the power consumption is significantly reduced and the performance remains highly competitive.
Furthermore, such an approach could be applicable to other types of Java servers also powered by a single JVM, such as the J2EE application server. In future work, we plan to investigate adapting this approach to more Java-based servers/devices for the power-efficiency.

References

[1] O.S.G. Alliance, “OSGi service platform, core specification, Release 4, Version 4.2,” OSGi Specification, 2009.
[2] C. Isci, A. Buyuktosunoglu, and M. Martonosi, “Long-term workload phases: Duration predictions and applications to DVFS,” IEEE Micro, vol.25, pp.39–51, 2005.
[3] K. Choi, R. Soma, and M. Pedram, “Dynamic voltage and frequency scaling based on workload decomposition,” Proc. 2004 International Symposium on Low Power Electronics and Design, pp.174–179, 2004.
[4] K. Flautner, S. Reinhardt, and T. Mudge, “Automatic performance setting for dynamic voltage scaling,” Wireless Networks, vol.8, pp.507–520, 2002.
[5] R. Gonzalez and M. Horowitz, “Energy dissipation in general purpose microprocessors,” IEEE J. Solid-State Circuits, vol.31, pp.1277–1284, 2002.
[6] W. Kim, M.S. Gupta, G.Y. Wei, and D. Brooks, “System level analysis of fast, per-core DVFS using on-chip switching regulators,” Proc. IEEE 14th International Symposium on High Performance Computer Architecture, pp.123–134, 2008.
[7] J.R. Lorch and A.J. Smith, “Improving dynamic voltage scaling algorithms with PACE,” Proc. 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp.50–61, 2001.
[8] G. Magklis, M.L. Scott, G. Semeraro, D.H. Albonesi, and S. Dropsho, “Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor,” Proc. 30th Annual International Symposium on Computer Architecture, pp.14–25, June 2003.
[9] R. Ge, X. Feng, W.-C. Feng, and K.W. Cameron, “CPU MISER: A performance-directed, run-time system for power-aware clusters,” Proc. 2007 International Conference on Parallel Processing, pp.18–25, 2007.
[10] T. Lindholm and F. Yellin, Java virtual machine specification, Addison-Wesley Longman Publishing, Boston, MA, USA, 1999.
[11] S.M. Blackburn, R. Garner, C. Hoffmann, A.M. Khang, K.S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S.Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J.E.B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D.V. Dincklage, and B. Wiedermann, “The DaCapo benchmarks: Java benchmarking development and analysis,” Proc. 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications, 2006.
[12] R.E. Jones and R.D. Lins, Garbage Collection: Algorithms for Automatic Dynamic Memory Management, John Wiley & Sons, New York, 1996.
[13] F. Xian, W. Srisa-an, and H. Jiang, “Garbage collection: Java application servers’ Achilles heel,” Science of Computer Programming, vol.70, pp.89–110, 2008.
[14] A.W. Appel, “Simple generational garbage collection and fast allocation,” Software: Practice and Experience, vol.19, pp.171–183, 1989.
[15] T.D. Burd, T.A. Pering, A.J. Stratakis, and R.W. Brodersen, “A dynamic voltage scaled microprocessor system,” IEEE J. Solid-State Circuits, vol.35, no.11, pp.1571–1580, 2000.
[16] Apache Felix, http://felix.apache.org/.
[17] OpenJDK, http://openjdk.java.net/.
[18] N. Sankaran, “A bibliography on garbage collection and related topics,” ACM SIGPLAN Notices, vol.29, pp.149–158, 1994.
[19] M.A. Cirit, “Estimating dynamic power consumption of CMOS circuits,” Proc. IEEE International Conference on Computer-Aided Design, pp.534–537, 1987.
[20] M. Weiser, B. Welch, A. Demers, and S. Shenker, “Scheduling for reduced CPU energy,” Mobile Computing, pp.449–471, 1996.
[21] C.Y. Tsui, M. Pedram, and A.M. Despain, “Efficient estimation of dynamic power consumption under a real delay model,” Proc. 1993 IEEE/ACM International Conference on Computer-Aided Design, pp.224–228, 1993.
[22] S. Herbert and D. Marculescu, “Analysis of dynamic voltage/frequency scaling in chip-multiprocessors,” Proc. 2007 International Symposium on Low Power Electronics and Design, pp.38–43, 2007.
[23] P. Pillai and K.G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” Proc. Eighteenth ACM Symposium on Operating Systems Principles, pp.89–102, 2001.
[24] T. Seki, S. Akui, K. Seno, M. Nakai, T. Meguro, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumanu, and M. Shimagura, “Dynamic voltage and frequency management for a low-power embedded microprocessor,” IEICE Trans. Electron., vol.E88-C, no.4, pp.28–35, April 2005.
[25] H. Hlavacs, et al., “Distributed energy efficiency in future home environments,” Annals of Telecommunications, vol.63, pp.473–485, 2008.
[26] N. Kushiro, S. Suzuki, M. Nakata, H. Takahara, and M. Inoue, “Integrated residential gateway controller for home energy management system,” IEEE Trans. Consum. Electron., vol.49, no.3, pp.629–636, 2003.
[27] S. Tompros, N. Mouratidis, M. Caragiolo, H. Hrasnica, and A. Gavras, “A pervasive network architecture featuring intelligent energy management of households,” Proc. 1st International Conference on Pervasive Technologies Related to Assistive Environments, article no.75, 2008.
[28] P. Griffin, W. Srisa-an, and J.M. Chang, “An energy efficient garbage collector for java embedded devices,” SIGPLAN Not., 2005, vol.40, no.7, pp.230–238, 2005.
[29] G. Chen, R. Shetty, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, and M. Wolczko, “Tuning garbage collection for reducing memory system energy in an embedded java environment,” ACM Trans. Embed. Comput. Syst., 2002, vol.1, pp.76–95, 2002.
[30] W.Y. Liang, P.T. Lai, and C.W. Chiou, “An energy conservation DVFS algorithm for the android operating system,” JOC, 2010, vol.1, pp.93–100, 2010.
[31] S. Hu and L.K. John, “Impact of virtual execution environments on processor energy consumption and hardware adaptation,” Proc. 2nd International Conference on Virtual Execution Environments, ACM, pp.100–110, Ottawa, Ontario, Canada, 2006.
[32] J. Shirako, et al., “Compiler control power saving scheme for multi core processors,” Languages and Compilers for Parallel Computing, E. Ayyadé, et al., ed., pp.362–376, Springer Berlin Heidelberg, 2006.
[33] C.Y. Lin, C.L. Lin, and T.W. Hou, “A graph-based approach for automatic service activation and deactivation on the OSGi platform,” IEEE Trans. Consum. Electron., vol.55, no.3, pp.1271–1279, Aug. 2009.
Kuo-Yi Chen  received the B.S., M.S., and Ph.D. degrees in Engineering Science from National Cheng Kung University, Taiwan, in 1999, 2001 and 2011, respectively. He is now a Post-doctoral Fellow at Institute of Information Science, Academia Sinica in Taiwan. His major research interests include Parallel computing, Performance study of Java Virtual Machines (JVM) and Green computing.

Chin-Yang Lin  received the B.S., M.S., and Ph.D. degrees all in Engineering Science from National Cheng Kung University, Taiwan, in 1999, 2001 and 2009, respectively. He is now a Postdoctoral Fellow at National Cheng Kung University in Taiwan. His major research interests include Open Services Gateway initiative (OSGi) technology, automatic memory management (garbage collection), embedded Java platform (Virtual Machine) and software engineering.

Tien-Yan Ma  received the B.S. and M.S. degrees in Engineering Science from National Cheng Kung University, Taiwan in 1998 and 2000, respectively. He is now a Ph.D. candidate in Engineering Science, National Cheng Kung University. His major research interests include information security, indoor navigation, Java card applications, and RFID applications.

Ting-Wei Hou  received B.S., M.S., and Ph.D. degrees all in Electrical Engineering, National Cheng Kung University, Taiwan, in 1983, 1985, and 1990 separately. He is now a professor in Department of Engineering Science, National Cheng Kung University. He was an associate professor in Department of Engineering Science, National Cheng Kung University, during 1990–2010. He was a visiting scholar of CSRD of University of Illinois at Urbana-Champaign, Illinois, U.S.A, during 1993–1994. His major research is in embedded systems and system integration. He has been the project leader of the pilot project on Healthcare IC cards in Penghu, Taiwan from 1998 to 2003, which encouraged the National Healthcare IC card project. He is currently working on Java based embedded systems, such as Java Virtual Machines, Java obfuscators, MHP, OSGi, Java card applications, and RFID applications.