This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

An Energy-Efficient Java Virtual Machine

Kuo-Yi Chen, J. Morris Chang and Ting-Wei Hou

Abstract—the power-saving opportunities of long-running application servers which execute on multi-core systems are studied in this paper. The research goal is to develop an efficient power-saving strategy of application servers with the minimum performance degradation in cloud environments. The power-saving strategy is based on the run-time information which is already available in a JVM, the base software component of application servers. Several key findings are revealed through this study. First, the particular behavior of application servers, also known as phases, can be related to the run-time information of a JVM. Thus the phases of an application server can be predicted before the applications actually execute on hardware. Secondly, some particular phases are observed in this study and used to establish the power-saving strategy, such as memory phases and execute phases. Finally, a new finding of idle phase is proposed to reduce significant energy wastage without performance degradations. Based on these findings, a set of power-saving algorithms is proposed and implemented with two widely used JVMs, Sun's Hotspot and Jikes RVM. With the experiments of five multi-threaded benchmarks and two web application benchmarks, the use of proposed power-saving strategy leads to the lowest value of EDP among the other power-saving techniques, and the performance degradation is well below six percents.

Index Terms—Energy efficiency; Java Virtual Machine; Application servers; Multi-core systems; Multi-threaded applications.

1 Introduction

In recent years, web applications have become a popular choice for service providers and already show their importance in the global marketplace. For example, a suite of web applications (e.g, E-mail, Document processing and File storages) was developed by Google has already influenced the use of the Internet [1]. The software in cloud environments that grants web applications to be served via the Internet is referred to as application servers. Industry market watchers expect the revenue of application servers could reach approximately 67 billion dollars by year 2018 [2].

With the trend of energy saving and carbon reduction, the energy wastage of long-running application servers is becoming an important issue. For example, while application servers tend to co-locate with data centers, the annual data center energy consumption in US is estimated to grow to over 140 billion kWh at a cost of $13 billion by 2020 [3]. Furthermore, the industry trend is toward integrating multiple cores on a chip [4]. As a result, multi-core processors are widely deployed on servers. The power consumption of multiple processor cores stresses the energy wastage issue of application servers.

In order to reduce power consumption of a long-running application server, the energy wastage of processors is highlighted. Since the processors consume the most of energy in a server platform [5]. The well-known power-saving technique is called the *Dynamic Voltage and Frequency Scaling* (DVFS), which is available in modern processors [6]. Many studies are based on the DVFS technique to adjust the voltage and frequency of processors to reduce CPU's power consumption [7]. These researches can be classified into two major groups, *profiling* and the use of *performance monitors*.

The profiling approach relies on the analyses of application behavior first, and then, uses this information to adjust frequencies of processors [8]. Due to the additional cost of code analyses and special instruction insertion, profiling approaches is rarely deployed on a system which requires quick responses and high performance, such as application servers.

On the other hand, the performance monitor is a set of registers in processors, which can be used to obtain hardware events. The observation of phases can be used to adjusted processors' frequency to save energy [9]. However, the use of performance monitors has limitations. First, phases only can be observed after hard-ware events are appeared in performance monitors. It is always one step behind. Secondly, the periods of a phase cannot be observed precisely. The actual start/end timing of a given phase is not known.

In order to improve the issues of profiling and the use of performance monitors, our motivation is to detect the phases of application servers precisely with the run-time information of the Java Virtual Machine (JVM), which is the base software of application servers. The instructions of Java web applications, also known as the bytecodes, have to be interpreted by the JVM, and then can be executed on the hardware. This feature leads a capability to observe the phase of application servers before the run-time behavior is actually changed.

The experimental results show that the use of proposed power-saving strategy leads to the significant energy reduction (14 to 23 percent) with the use of long-running application server benchmarks, which is better than other power-saving techniques (10 to 16 percent). It is worth noting that the slight performance degradation (four percent) is observed with the use of proposed strat-

- *Kuo-Yi Chen is with Department of CSIE, National Formosa University, Taiwan. E-mail: Kuoyichen@nfu.edu.tw.*
- *J. Morris Chang is with the Electrical Engineering Department, Iowa State University, Ames, IA, 50010. E-mail: morris@iastate.edu.*
- *Ting-Wei Hou is with Department of Engineering Science, National Cheng Kung University, Tainan, Taiwan. E-mail: houtw@mail.ncku.edu.tw.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

egy, which is also better than other power-saving techniques (8 to 12 percent). The experimental result shows can reach the goal of this study.

In the age of cloud computing, the application servers play important roles both in cloud and mobile computing. Such as weather-forecasting, IM service, map service and social network service, application servers have to performance 24x7 stably. As an application server that draws 500 W consumes 0.5 kWh actively [50]. There is 4380 kW was consumed by this server per year. It is worth noting that Amazon deployed over 1.5 to 2 million servers globally based on a conservative estimation in 2015 [51]. Assuming the proposed power-saving technology saves 15 percent energy for each server, and then 976,000 megawatts, which is equal to 81 Fort Calhoun nuclear power plants in Nebraska, could be saved per years. On the other hand, the proposed power-saving technology could be used to develop better power-performance balance algorithm for devices which is powered by virtual machines, such as Dalvik VM of Android systems.

This paper is organized as follows. First, the run-time behaviors of JVM's software components are analyzed with the use of a single core to validate its phases. Secondly, the run-time behaviors of JVM's software components are studied with the use of multiple cores to detail their interaction. Thirdly, the particular phases, such as the execution phase, memory phase and idle phase, are analyzed to observe the power-saving capability. Finally, based on the study of JVM's software components, the power-saving strategy is proposed and examined.

2 Background and Related Work

This chapter describes the terminology and definitions which relate to application servers, JVM, garbage collection and the current power saving technology. It is worth noting that the term, application servers, is referred to Java-based application servers in this study.

2.1 The Structure of Application Servers

In order to reach the requirements of security and portability, web applications are usually hosted on the application server with Java techniques. The application server can be considered as a container of various types of web applications, such as *Servlets*, *Enterprise JavaBeans* (EJB) and *JavaServer Pages* (JSP). These web applications use application servers as an interface to exploit external resources, such as the hardware, network and databases.

The structure of a Java-based application server is shown in Figure 1. The libraries of *Java 2 Enterprise Edition* (J2EE) provide the support for web applications. Moreover, the JVM of *Java 2 Stand Edition* (J2SE) is used to interpret the bytecode of web applications to machine codes, and then exececute them on hardware. It is worth noting that an application server is executed with a JVM instance. In order to exploit the run-time information which is available in a JVM to reduce the energy wastage, the JVM behavior is studied in this research.

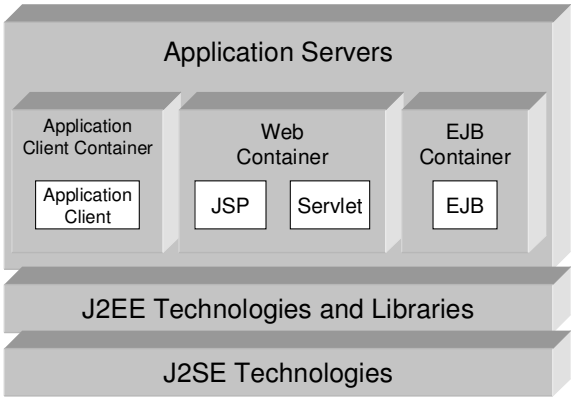


Figure 1. The structure of a Java application server

2.2 The Structure of Java Virtual Machines

In a JVM, the garbage collector is used to collect inaccessible objects, which are considered as the garbage. The high latency of memory access usually leads to CPU waiting for the results from the memory. The energy is consumed by the waiting CPU, and system performance is not improved. Thus the phase of garbage collections might be a power-saving capability of a JVM. The structure of a JVM is shown in Figure 2

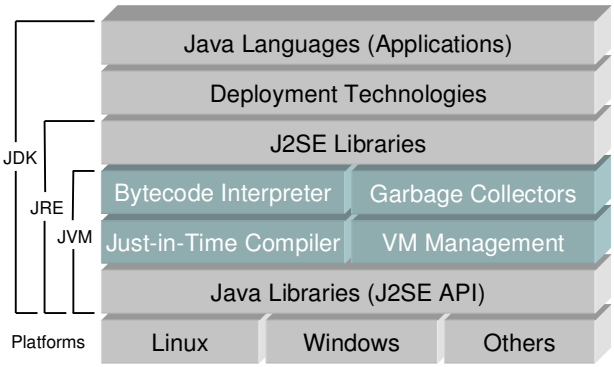


Figure 2. The structure of Java virtual machines

Based on the analysis of JVM's software components, we hypothesize that the behavior of the *vm_thread* and the *garbage collector* of the JVM could be the power-saving capability of application servers. This hypothesis will be verified with the further behavior analysis of vm_thread and the garbage collector.

2.3 Related works

As hardware components are growing into more power-hungry than ever, the power consumption of the long-running servers is becoming an interesting topic. Recent studies [10-12] have demonstrated that the power consumption could be retrenched by applying the lower power-level on the particular hardware component. Moreover, due to the significant energy demands of CPU in server systems, many studies focused on reducing power consumption of CPU [5]. Weiser *et al*. [13], have demonstrated the use of *Dynamic Voltage Scaling* (DVS) to reduce energy wastage of CPU. Further studies [14-16] explored the performance of DVS techniques in the gen-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

eral-purpose and real-time systems.

It is worth noting that there are more approaches to reduce the power-saving of a cloud/data center. Such as the studies of power-saving techniques on data centers and content delivery networks [51-52]. These studies show the distributed databases architecture has highly potential to reduce power wasting. The use of virtualization technology also leads to significant power saving [53]. With load-aware scheduling, the energy management could be more efficiency [54-56]. However, these approaches required more coordination between various servers, thus the side effect evaluation is required.

These studies put emphasis on the methodologies of phase observations. The information of applications' phases can be used to adjust the power-level of the objective system, and then reach the goal of power-saving. The methodologies of phase observations can be classified into two major groups, the *profiling* and the use of *performance monitors*.

There are many power-saving studies which are based on the profiling methodology. Delaluz *et al*. [17] and Hsu *et al*. [18, 19] use the compiler-directed profiling approach to reduce power consumption. The phases-transition instructions of particular power-levels are inserted into binary codes of applications based on the offline profiling.

However, the additional profiling work leads to the unavoidable overhead. In addition, the compiler-directed profiling only works with a single application at a time. In run-time, other processes might affect the system status, and reduce the profiling accuracy of the objective application. It leads to either less efficiency or great overheads of profiling approaches in multi-tasking and multi-cores systems. Instead of the additional work of the profiling, our approaches use the run-time information which is already available in virtual machines. Thus the overheads of phase detections could be limited.

On the other hand, the phase detection by performance monitors is widely used in industries and researches. The information of performance monitors can be observed by the run-time statistics, built-in hardware registers and external sensors. The use of run-time statistics [20-23], such as the *processor usage*, is a popular approach to reduce energy wastage. Some studies use performance counters, a set built-in registers of CPUs, to detect the phase and adjust the power-level [24, 25].

In addition, the use of external sensors, such as the thermal probe, to observe the heat generation and adjust the power-level is also a popular approach [26-28]. Some studies use a group of hardware registers, such as the numbers of stall cycles and retired instructions, to detect the phase and adjust the power-level [29-31]. Moreover, some studies use the information of hardware registers to improve task scheduling, and reach the goal of power-saving [32, 33]. Due to the variations of performance monitors only can be observed after the phase is changed, thus the possibility of energy wastage and performance degradations could be remained by this approach. In order to eliminate this possibility, the beforehand phase detection is proposed in proposed approach. The phases could be observed precisely by the run-time information which is already available in a JVM, before they actually change. The precise information of phases could be used to adjust appropriate frequencies, and then reduce the most of energy wastage with performance maintenance.

Compared with the traditional power-saving approaches which are mentioned as above, the power-saving approaches which are based on the use of virtual machines (VMs) are rarely demonstrated. Fries *et al*. [34] propose an approach to find an optimizing placement of a group of VMs on multiple hardware platforms to reduce the energy cost. An online method is proposed to configure the configuration of VMs and reduce the number of physical hosts [35]. The dynamically rescheduling is proposed to collocate processing heterogeneous workloads of VMs [36]. However, the phase detections of these approaches are based on the use of performance monitors. Thus these approaches cannot detect phases as accurately as the use of GVM approaches.

3 THE ANALYSIS OF JVM SOFTWARE COMPONENTS

In order to reach the goal of power-saving by the run-time information which is available in a JVM, the experiments are proposed to analyze the behavior of JVM's software components. The experimental steps are shown as follows. First, the setup of experiments is detailed. Secondly, JVM's software components are analyzed with the use of a single core. Thus the behavior of each JVM's software component could be observed separately. Finally, the interactions of each JVM's software component are studied with the use of multiple cores. Thus the interaction between each component is detailed. Based on experimental results, the behavior of JVM's software components is clarified to develop the power-saving strategy.

3.1 The setup of experiments

All the experiments are based on a server with the Q6600 processor, an Intel quad-core CPU. The frequency of each core is allowed to be adjusted independently. The four available frequencies of the Intel Q6600 are 2.4Ghz, 2.13Ghz, 1.87Ghz and 1.60Ghz. The Fedora Core 14 with kernel version 2.6.35 and Sun's Java System Application Server 9.1 are used as the operating system and the application server in these experiments.

In order to evaluate the performance of various power-saving approaches, the appropriate technique to measure the power consumption of processors is important. In general software approaches, such as dynamic power measurement of CMOS circuits, the power consumption of a CPU is expressed as the square value of voltage plus frequency of each core. However, only dynamic power consumption is measured by the software approach. Due to the issue of leak current, the static power consumption is becoming significant in modern processors [37]. The lack of static power measurement might result in the inaccurate evaluation of CPU power consumption [38].

In order to improve this issue, a new hardware approach is proposed [39]. Two accurate digital power meters are used to measure the power consumption of proc-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

essors by the current and voltage variation of a special electronic socket, as known as *Voltage Regulator Module* (VRM) [40], which is integrated on the motherboard to provide power to main processors. The VRM is consisted by several converters in parallel and usually has special controls that respond to signals from the processor, such as *Voltage Identification Code* (VID). It is worth noting that the output voltage of the VRM is varied based on the demand of processors.

3.2 Multi-threaded Java Benchmarks

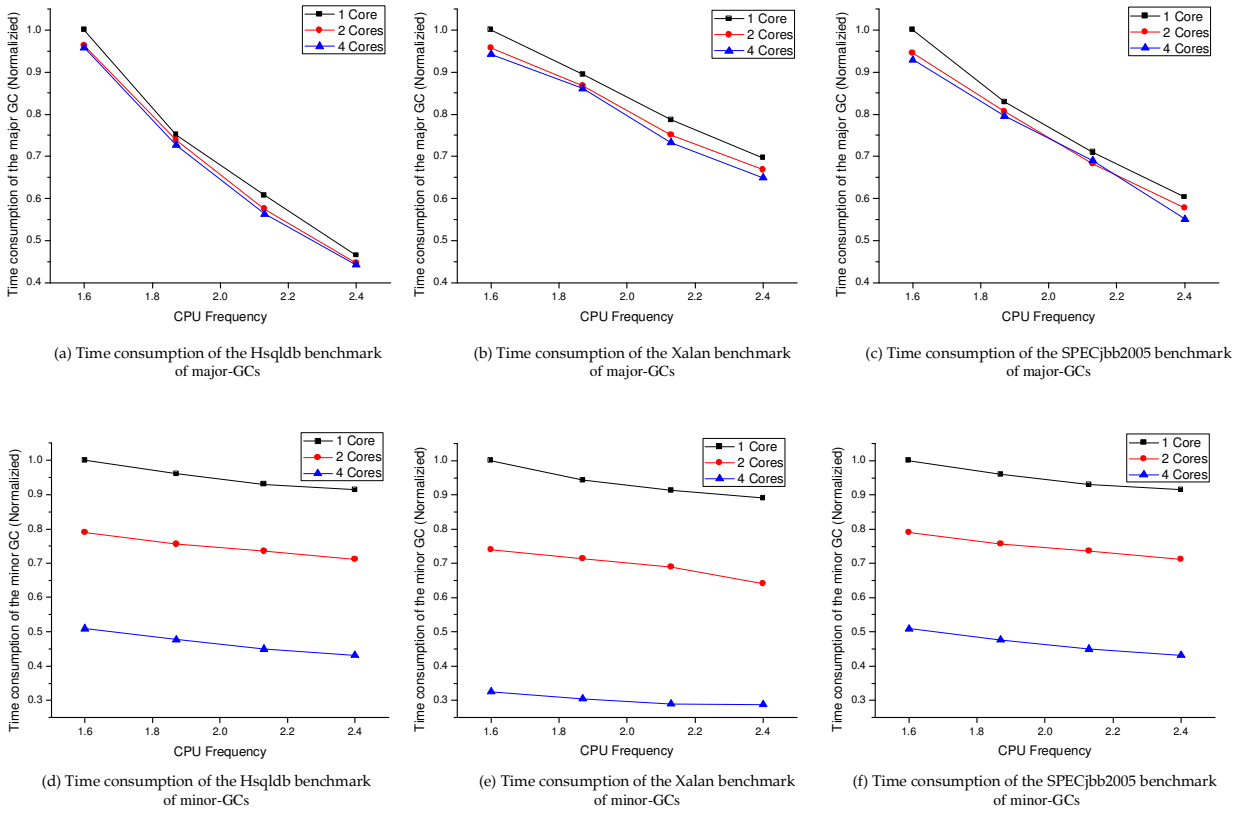
In order to approximate the multi-threading feature of web applications, five widely used multi-threaded Java benchmarks are examined in this study. They are *Eclipse*, *Hsqldb*, *Lusearch* and *Xalan* from the *Dacapo* benchmark suit [42], and *SPECjbb2005* benchmarks [43]. These multi-threaded benchmarks perform various types of workloads to present the features of web applications.

Moreover, in order to validate the performance of proposed power-saving approaches with the use of application servers, two widely used web application benchmarks, *SPECjAppServer2004* and *RUBiS*, are used in experiments. SPECjAppServer2004 [44] uses a large and representative sample of J2EE APIs to evaluate the performance of the individual system. RUBiS [45] is an online auction site modeled after e-Bay. The experimental results of SPECjAppServer2004 and RUBiS benchmarks are used to represent the performance and power consumption of the experimental application server, Sun's Java System Application Server 9.1.

3.3 The Behavior Analysis of the Garbage Collector

Due to the higest market share of Hotspot JVM, a state-of-the-art Hotspot JVM, shipped with OpenJDK 1.7 [46], is used in the experiments. With the use of Hotspot JVM, the parallel garbage collection is also referred to as the throughput collector. It uses a parallel version of the young generation collector. The old (tenured) generation is still cleaned with the default collector. On the other hand, the concurrent garbage collector is also referred to as the concurrent low pause collector. It collects garbage in the old (tenured) generation concurrently to executing the application.

It is worth noting that the use of various heap sizes could lead to significant effect when the NewRatio technique is applied. The size of the heap could determine the frequency of collection and affect the locality of both old and young objects. For example, the use of bigger heap could reduce the frequency of collection. On the other hand, the time consumption of each collection will increase due to more objects have to be collected in the heap.



(a) Time consumption of the Hsqldb benchmark of major-GCs

(b) Time consumption of the Xalan benchmark of major-GCs

(c) Time consumption of the SPECjbb2005 benchmark of major-GCs

(d) Time consumption of the Hsqldb benchmark of minor-GCs

(e) Time consumption of the Xalan benchmark of minor-GCs

(f) Time consumption of the SPECjbb2005 benchmark of minor-GCs

Figure 3. The time consumption of major/minor GCs with the use of various numbers of processor cores

The configuration of garbage collections is 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 genera-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

tions, the young generation and the tenured generation [47]. 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 [48].

In order to verify our hypothesis, the experiments are proposed to clarify the hypothesis. In this experiment, three multi-threaded benchmarks are used to examine the time consumption of two main phases of a garbage collector, the minor garbage collection (minor-GCs) and the major garbage collection (major-GCs).

In order to detail the behavior of the minor-GC and major-GC with the use of multi-core processors and different CPU frequency, the experiment is proposed to examine the time consumption of the minor-GC and major-GC. The less time consumption of GC indicates the performance improvement by the use of more cores or the higher frequency. Based on the experimental result, the power-saving capability of garbage collectors can be verified.

The minor-GC time consumption of three multi-threaded benchmarks, Hsqldb, Xalan and SPECjbb005, is shown in Figures 3 (a) to (c). It is observed that time consumption of the minor-GC is reduced significantly with the use of more processor cores. This observation shows that the gc_threads can take the advantage with all available multiple processor cores in a minor-GC. That also indicates that the gc_threads do the parallel collection. On the other hand, the slight time consumption change of the minor-GC is observed when the use of different CPU frequency. This observation shows that the processor is waiting for the memory accesses, the objecting moving from the young generation to the old generation. Since the speed of the memory accesses is much lower than the speed of the processor, the processor has to wait for the memory accesses, and results in the plenty memory stalls. Based on this observation, it can be assumed that the minor-GC is highly related to the memory phase rather than the execution phase. It is worth noting the CPU frequency can be minimized to reduce the energy wastage during a memory phases without the significant system performance degradation. Thus a minor-GC can be considered as a power-saving capability of a JVM.

The time consumption of the major-GC is shown in Figures 3 (d) to (f). Instead of the minor-GC, the slight variation of time consumption is observed when the number of processor cores is increasing. On the other hand, the significant time reduction is observed when the CPU frequency is increasing. The observations show that the performance of the major-GC is correlated to the computing power of processors. Thus the major-GC is highly related to the execution phase than the memory phase.

Furthermore, the major-GC cannot take advantage with the use of multiple cores, which can be observed by the similar time consumption with the use of various processor cores. When a major-GC is engaged, a lock is used to guarantees that no other JVM thread is in the middle of modifying the Java heap. Therefore, the global work can be accomplished correctly. Since only one JVM thread can work in the major-GC, the use of multiple processor cores cannot reduce the time consumption of the major-GC.

As a brief summary, the minor-GC takes advantage with the use of multiple cores, but not the increasing of the CPU frequency. Thus it can be hypothesized that the configuration of using all available cores and setting them to the minimum CPU frequency might lead to the performance maintenance and the energy wastage reduction for the minor-GC. On the other hand, since the major-GC takes advantage with the high CPU frequency but not multiple processor cores. It can be hypothesized that the use of a single core with the maximum CPU frequency might lead to high performance and the low power consumption for a major-GC.

In order to verify the hypotheses, the experiment which is based on various numbers of processor cores is proposed in the next section. The behavior of the minor-GC and the major-GC are analyzed by certain hardware events particularly. Thus the time consumption and the power consumption of the garbage collector, which is the power-saving capability, can be detailed further.

3.4 The Study of JVM's Software Components with the single core

In order to verify the hypotheses, this experiment is proposed with the use of a single core in this section to detail the particular behavior of each JVM's software component. With the use of the single core, the JVM works as a single thread application. Thus each JVM's software component executes sequentially. Therefore, the interaction of multi-threading and multi-cores could be avoided, and then the particular behavior of each JVM's software component can be observed and analyzed.

In order to detail the particular behavior of each JVM's software component, the hardware events are monitored to determin the phases of components. Two important hardware events, *Instr_Ret* and *IFU_Mem_Stall*, are monitored by accessing the performance counter which is built in the CPU. Instr_Ret counts the number of retired instructions and IFU_Mem_Stall counts the number of stalled cycles while a CPU waits for results of memory access.

Moreover, a special measure called *Stall Cycle Per Instruction* (SCPI) is employed (tracked) to analyze the phases of the multi-threaded benchmarks. The SCPI is defined as follows:

$$SCPI = \frac{\text{Mem\_Stall}}{\text{Instr\_Ret}}$$

where Mem_Stall and Instr_Ret are both the built into hardware events. Instr_Ret stands for the number of instructions retired. Mem_Stall represents the number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

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.

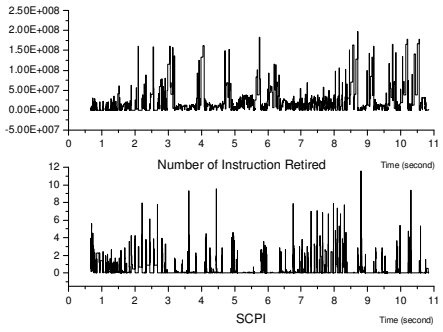


Figure 4. The global behavior of the JVM

There are four figures are used to represent the behavior of JVM's software components. The global behavior of a JVM (including all JVM's software components) is shown in figure 4. The behavior of the major JVM's software components, which are the vm_thread and gc_thread, are shown in figure 6 and 7. Two measures, the number of instruction retired and the SCPI value, are used to present the behaviors of a JVM and its software components. The time scale (X-axis) is the same in all figures. Thus the comparison between JVM's software components can be reached. Furthermore, the engaged and disengaged timing of the major-GC and minor-GC is shown with the same time scale (X-axis), thus the periods of GC can be observed and compared with the behavior of JVM's software components.

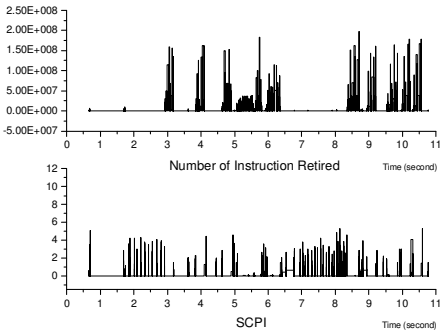


Figure 5. The global behavior of the VM_thread

Compared with the global behavior of JVM in Figure 4, it is observed that the most of Instr_Ret are generated by the vm_thread and the gc_thread in Figures 5 and 6. It is worth noting that a vm_thread and the gc_thread are both highly related with the garbage collection, the power-saving capability of a JVM. In order to detail the correlation of software components and garbage collections, the periods of garbage collections (including the minor-GC and major-GC) are shown in Figure 7.

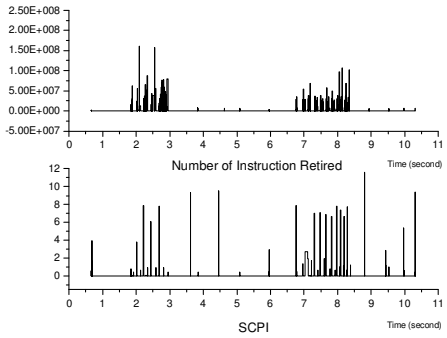


Figure 6. The behavior of the gc_thread

In a major-GC, the large numbers of Instr_Ret are observed in a vm_thread in Figure 7. However, the large Instr_Ret numbers of a vm_thread only lead a few SCPI value in a major-GC. This observation shows that the few memory access stall cycles are generated in a major-GC. The workload of a vm_thread, to visit all objects and finds inaccessible objects, leads to this observation. Thus the period of a major-GC is considered as the execute phase due to the few memory access stall cycles of the vm_thread.

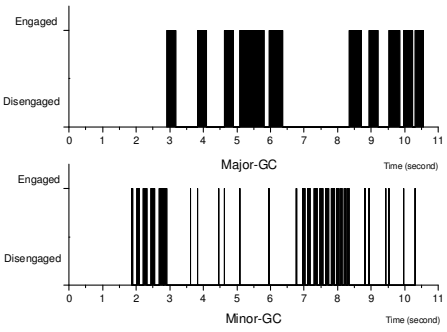


Figure 7. The engaged/disengaged timing of the major-GC and minor-GC

Moreover, the significant peaks of the SCPI value are observed at the end of each major-GC in Figure 7. In the execution round, the SCPI peaks are observed in 8.76 second, 9.43 second, 10.05 second and 10.32 second. The SCPI peaks are due to the large number of memory access stall cycles, which are generated by the garbage collection of the gc_thread. Due to the observation of the large number of memory access stall cycles, the end of a major-GC should be considered as the memory phase.

On the other hand, the significant SCPI values are observed in each minor-GC. Compared the Instr_Rets values of the vm_thread and gc_thread in Figures 7, it could be observed that the most of Instr_Rets are generated by the gc_thread. At the same time, the gc_thread also generates significant memory stall cycles. Therefore, it leads the significant SCPI value in a minor-GC. This observation verifies the hypothesis: a minor-GC is highly

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

related to the memory phase

As a brief summary, the phase of the major-GC is considered as the execution phase due to the behavior of a vm_thread. Furthermore, the end of a major-GC, which is the period for the gc_thread collects garbage, should be considered as the memory phase. On the other hand, a minor-GC should be considered as the memory phase due to the behavior of the gc_thread. It is worth noting that the engaged and disengaged timing of the major-GC and minor-GC can be observed with the run-time information which is already available in a JVM. Thus the accurate phase determination can be reached to develop the power-saving strategy of a JVM.

3.5 The Study of JVM's Software Components with multiple cores

In general, the modern multi-threaded applications can take advantage by the use of multiple core processors. However, due to the interaction of the software and hardware, such as locks, task scheduling, parallelism and cache locality. A multi-threaded application might not take advantages fully with all available cores. Moreover, the interaction might lead the to the energy wastage. For example, the poor task scheduling might lead to that one processor core waits for another core, and then results in the energy wastage due to the idle waiting of the processor core.

In order to analyze the interaction of software and hardware with the use of multiple cores, two multithreaded benchmarks, Hsqldb and Xalan, are examined with the multi-core processor (four cores) in this experiment. Due to the finding in Section 3.4, the garbage collection could be the power-saving capability of a JVM, thus the interaction of major-GC and minor-GC are detailed further in this experiment.

The experimental results of two multithreaded benchmarks are shown as follows. For Hsqldb benchmark, the number of instruction retired, the SCPI value and engage/disengage timing of GC are shown in Figures 8 (a) to (c). For Xalan benchmark, they are shown in Figures 8 (d) to (f). The time scale (X-axis) is the same in these figures. The analysis of the experimental results is as follows

First, compared with the number of Instr_Ret and the engage/disengage timing of the major-GC in Figures 8 (a) to (c), the significant number of Instr_Ret are observed in only one core. On the other hand, the very few number of Instr_Ret is observed with the other cores at the same time. Furthermore, the same experimental result of major-GC is observed with the use of Xalan in Figures (d) to (f). The reason which leads to this observation is detailed as follows

The use of locks for the JVM's global work leads to this observation in a major-GC. In a major-GC, the vm_thread uses the lock mechanism to avoid the heap modification by the other JVM threads. Thus the status of objects in a heap can be preserved. With the use of the lock, the vm_thread can visit the stacks of Java_threads and the heap to find the inaccessible objects. However, the use of a lock might lead the other cores are idle. That seems the reason which leads to the specific observation of the major-GC, only one core works and the other cores are idle

Secondly, the low values of SCPI are observed only with the core which is doing the major-GC. This observation is due to the behavior of a vm_thread in a major-GC. With marking inaccessible objects, the status of objects are changed, but not be collected. Thus the few memory access stalls are generated. That shows the core which a vm_thread executes with is highly related to execution phases. On the other hand, the extreme high SCPI values could be observed in the other idle cores. This observation details the behavior of a major-GC further with the use of multiple cores. In a major-GC, the core which a vm_thread works with is considered as the *execution phase*, and the other cores are related to *idle phases*.

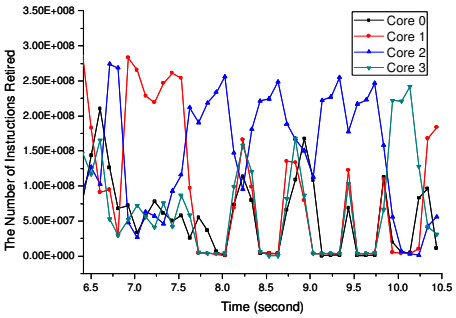
Finally, in a minor-GC, the high numbers of Instr_Ret and the high SCPI value are observed in all cores. The high number of Instr_Ret indicates that the garbage collector exploit all available cores in a minor-GC. The observation of high SCPI values verifies that the minor-GC is highly related to memory phases. Since the high number of Instr_Ret can be observed in all available cores, the observation validates that the parallel garbage collector works during a minor-GC. All cores work in a minor-GC, and all cores are related to the memory phase.

These findings of garbage collectors provide the power-saving capability of a JVM. First, the CPU frequency of the idle cores in a major-GC should be minimized to reduce energy wastage since they are idle waiting for the lock. It is worth noting that the CPU frequency tuning would not reduce the system performance since the frequency tuning is only applied on the idle cores. Secondly, the CPU frequency the specific core, which a vm_thread executes with, should be maximized to maintain the performance of garbage collection in a major-GC. Thirdly, the CPU frequency should be maximized for all available cores in the garbage collection, which is observed at the end of a major-GC. Finally, the frequency of all available cores should be minimized in a minor-GC due to that minor-GC is highly related to memory phases.

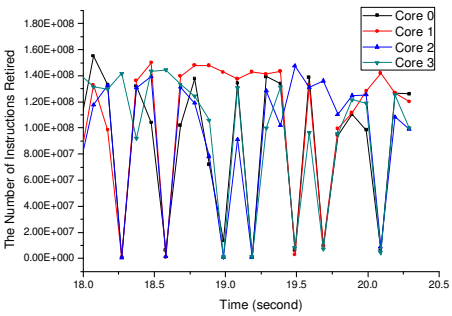
3.6 The Summary of Studies

As a brief summary, the findings of this study are shown as follows. First, only one core work in the major-GC, and its phase is the execution phase. Secondly, the other cores in a major-GC are idle phases. Finally, all cores can work in a minor-GC and their phases are memory phases. Moreover, the beginning and ending of these phases could be observed by the run-time information of a JVM before the phases are actually changed. Based on these findings, the power-saving algorithms and implementation of the Green Virtual Machine (GVM) approach are proposed in the next section.

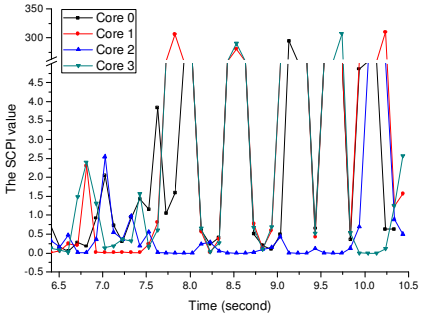
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing



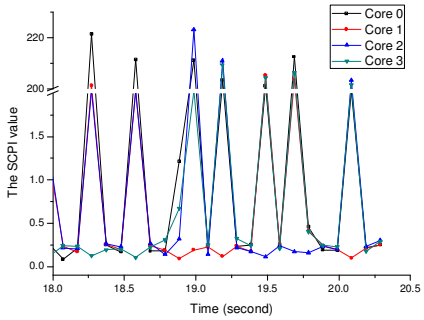
(a) The number of instruction retired of Hsqldb



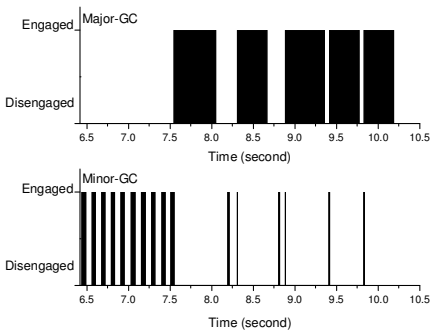
(d) The number of instruction retired of Xalan



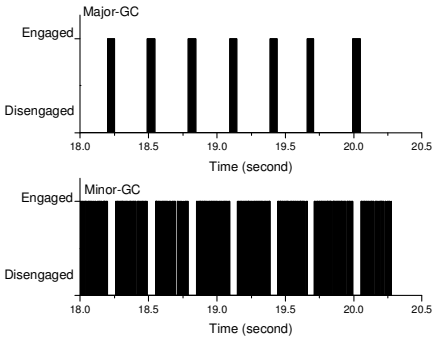
(b) The SCPI values of Hsqldb



(e) The SCPI values of Xalan



(c) The GC engaged/disengaged timing of Hsqldb



(f) The GC engaged/disengaged timing of Xalan

Figure 8. The behavior of Hsqldb and Xalan with the use of four cores

4 The power-saving strategy of a JVM

Based on the finding above, the power-saving strategy of a JVM, also known as Green Java virtual machine (GVM), is proposed in this section. The GVM power-saving approach is combined with two power-saving algorithms, major-GC and minor-GC. In addition, the advantages of GVM power-saving approach are also discussed in this section.

4.1 The algorithms of the power-saving strategy

Based the phase analysis of garbage collectors, the GVM power-saving algorithms are proposed to exploit the run-time behavior in a JVM. Due the different behavior of the minor-GC and major-GC, the GVM algorithms are constructed by two parts as follows.

First, the power-saving capability of a JVM, the major-GC, is used to develop the major-GC power-saving algorithm. Based on the experiment result, a major-GC can be considered as a combination of the execution phase, the idle phase and the memory phase. The execution phase is observed in the core which a vm_thread executes with, and the idle phases are observed in the other cores. Furthermore, in the end of a major-GC, all available cores are becoming the memory phase since the garbage is collected.

The CPU frequency of the specific core, which the vm_thread executes with, should be maximized to maintain performance of a major-GC. The CPU frequency of idle cores should be minimized to reduce energy wastage without performance degradation. In the end of a major-GC, the CPU frequency of all available cores should be minimized since the memory phase. Based on the find-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

ings, the major-GC power-saving algorithm is proposed in Algorithm 1.

The Major-GC Power-Saving Algorithm:
for each Major-GC event *Ema*, issued by the JVM process
if *Ema* = Major-GC_START **then**
for the core which vm_thread executes CPU *Cv*
C_v := *maximum frequency*;
for the other idle cores CPU *Ci*
Ci := *minimum frequency*;
else if *Ema* = GC_START **then**
for each available CPU C_a
C_a := *minimum frequency*;
else if *Ema* = Major-GC_FINISH **then**
for each available CPU C_a
C_a := *maximum frequency*;
end if

Algorithm 1. The power-saving algorithm for Major-GC

In the major-GC power-saving algorithm, the engage/disengage timing of a major-GC can be observed by the run-time information of a JVM. Moreover, the specific core (Cv), which the vm_thread executes with, also can be observed by the run-time information of a JVM. In the period of a major-GC, the CPU frequency of Cv is maximized to maintain the performance. On the other hand, the CPU frequencies of the idle cores (Ci) are minimized to reduce the energy wastage of processors. When the vm_thread finish the finding of the inaccessible objects, the CPU frequency of all available cores (Ca) should be minimized since the garbage collection is the memory phases. After a major-GC, all cores work for application normally, thus the CPU frequencies of Ca should be maximized to maintain system performance. It is worth noting that the CPU frequency tuning would not reduce the system performance since the tuning only applies on idle cores (Ci).

The Minor-GC Power-Saving Algorithm:
for each Minor-GC event *Emi*, issued by the JVM process
if *Emi* = Minor-GC_START **then**
for each available CPU C_a
C_a := *minimum frequency*;
else if *Emi* = Minor-GC_FINISH **then**
for each available CPU C_a
C_a := *maximum frequency*;
end if

Algorithm 2. The power-saving algorithm for Minor-GC

Another power-saving capability of a JVM, the minor-GC, is used to develop the minor-GC power-saving algorithm. Based on the study above, a minor-GC is considered as a memory phase for all available cores. Based on this finding, the minor-GC power-saving algorithm is proposed in Algorithm 2.

In the minor-GC power-saving algorithm, the engage/disengage timing of a minor-GC can be observed by the run-time information of a JVM. Due to the minor-GC is highly related to memory phases, the CPU frequency of all available cores should be minimized to wait the memory access. Thus the energy wastage can be reduced in a minor-GC.

Based on the observation in Figures 8(e) and 8(f), the major-GC and minor-GC would not be activated at the same time. That means the period of the major-GC and minor-GC are not overlapped. Therefore, the major-GC and minor-GC power-saving algorithms can be integrated as the GVM power-saving algorithm in Algorithm 3.

The GVM Power-Saving Algorithm:
for each Major-GC event *Ema*, issued by the JVM process
if *Ema* = Major-GC_START **then**
for the core which vm_thread executes CPU *Cv*
C_v := *maximum frequency*;
for the other cores CPU *Ci*
Ci := *minimum frequency*;
else if *Ema* = GC_START **then**
for each available CPU C_a
C_a := *minimum frequency*;
else if *Ema* = Major-GC_FINISH **then**
for each available CPU C_a
C_a := *maximum frequency*;
end if

for each Minor-GC event *Emi*, issued by the JVM process
if *Emi* = Minor-GC_START **then**
for each available CPU C_a
C_a := *minimum frequency*;
else if *Emi* = Minor-GC_FINISH **then**
for each available CPU C_a
C_a := *maximum frequency*;
end if

Algorithm 3. The power-saving algorithm

The GVM power-saving algorithm can be considered as the integration of Algorithms 1 and 2. The GVM power-saving algorithm can take advantages from the minor-GC and major-GC. All GVM power-saving algorithms, including major-GC, minor-GC and GVM, are implemented with two widely used JVMs, Sun's Hotspot and Jikes RVM. Instead of Sun's Hotspot, which is the widely deployed commercial JVM, Jikes RVM can be considered as an academic JVM for research. Based on the experimental results with these two JVMs, the power-efficiency of proposed GVM approach can be proved.

The implementation of the GVM power-saving approach is illustrated in Figure 9. This, technically, involves (1) instrumenting the JVM of the Application servers for detecting the idel, execute and memory phases, and (2) the procedure of adjustment the CPU frequencies of the device/server.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

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 major-GC or minor-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. 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.

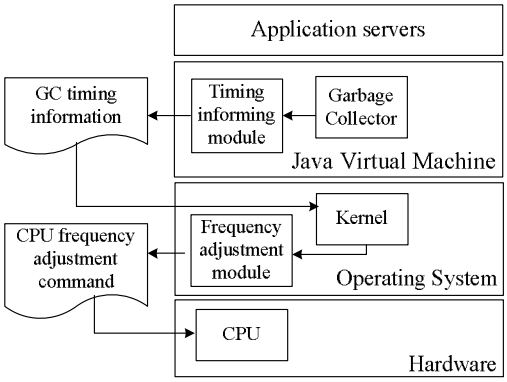


Figure 9. The implementation of proposed strategy

4.2 The advantages of proposed GVM power-saving strategy

In order to detail the advantages of the GVM power-saving approach, the further analysis are proposed. The advantages of GVM approach, which include slight overheads of phases detections, precise phases timing, accurate phase determinations and performance maintenance, are analyzed in this section.

First, phases detecting plays an important role in the most of power-saving approaches. Based on the results of phase detections, the appropriate CPU frequencies could be applied to reduce energy wastage. However, the phase detections usually lead to the overheads, and reduce the system performance. For example, the requirement of additional profiling work could be considered as the overhead in profiling approaches. Moreover, the performance monitor accesses also lead to overheads of power-saving approaches by performance monitors.

It is worth noting that the phase detections of GVM power-saving approaches are based on the run-time information which is already available in a JVM. Such as the engage/disengage timing of the major-GC and minor-GC. Moreover, the particular phases of the major-GC and minor-GC are constant and already observed. Thus the phase detections would not lead to the overhead in the GVM power-saving approach.

Secondly, the inaccurate timing of phases is a disadvantage of power-saving approaches with the use of performance monitors. The particular phase is identified only after the actual behavior already appeared in the performance monitors. It is always one step behind. The inaccurate phases timing could lead to unnecessary performance degradations and the energy wastage.

On the other hand, the period of garbage collection can be marked by the engage/disengage time of the garbage collector precisely. The timing can be used to determine the phase of the JVM. Moreover, Due to the middle-ware features of a JVM, the phases can be observed before they are arisen on hardware actually. Thus accurate phase timing can be reached by the GVM power-saving approach.

Thirdly, in the profiling power-saving approach, the observed phases could be affected by the other tasks in the run-time. It leads to that the observed phase is not identical to the profiled phase. The inaccurate information of phases might lead to inappropriate frequency tuning, and then results in the unnecessary performance degradations and energy wastage.

On the other hand, the accurate phase determination can be reached by the GVM power-saving approach. Since GVM power-saving algorithms focus on the phase of the garbage collector, which is related to a JVM itself instead of the application. The phases of garbage collectors are stable even with the use of different application. Therefore, the phase determination of GVM would not be affected by applications. The unnecessary performance degradation and energy wastage can be avoided.

Finally, the advantages of GVM power-saving approach improve the problem of the other power-saving approaches, performance degradations. Due to the unavoidable overheads of phase detections, inaccurate phases timing and inaccurate phase determinations, the system performance usually is degraded. However, the major-GC power-saving algorithm proposes the power-saving solution with the well performance maintenance. Thus the major-GC algorithm could be the power-saving solution for a web application server which highly requires performances and the quick response.

5 The experimental results

In this section, the performance of the GVM power-saving approach is examined and compared with the other power-saving techniques. Five widely used Java multi-threaded benchmarks are used to evaluate the performance of power-saving techniques with the use of Sun's Hotspot. Furthermore, two multi-tier web application benchmarks, SPECjAppServer2004 and RUBiS, are used to validate the GVM's performance with two JVMs, Sun's Hotspot and Jikes RVM. Based on these experiments results, the GVM's performance could be examined and clarified.

It is worth noting that 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. Since this measure considers both energy and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

delay simultaneously, the EDP value can better reflect the power-efficiency. In general, a lower EDP value indicates a better power-efficiency.

5.1 The Comparing of Power-saving Techniques

In order to compare the performance of GVM power-saving approach with the other power-saving techniques, six power configurations are proposed to examine the performance and power consumptions of power-saving technologies. The experimental power configurations are illustrated as follows.

First, two static CPU frequencies, the maximum and minimum frequencies, are used as the control group in experiments. The two static configurations are mapped to the performance and power-saving governors in Linux. In general, the use of maximum frequency leads to the best performance and the use of minimum frequency leads to the worst performance. Moreover, the use of minimum frequency usually leads to the significant power consumption due to the long executing time.

Secondly, two power-saving algorithms of the GVM, the major-GC power-saving algorithm (Algorithm 1) and minor-GC power-saving algorithm (Algorithm 2), are used to examine their performance. Furthermore, 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 performance monitor-based power-saving approaches.

In order to compare the combine of ondemand and conservative, a merge governor, *Smartass*, is evaluated. Smartass could be considered as a merge of the best properties of the opposite of conservative and ondemand. This governor attempts to balance performance with efficiency by focusing on an ideal frequency.

All available cores (four cores) are used in these experiments, and the CPU frequency of each core can be adjusted independently. The experimental result of system performance, power consumptions and EDP are shown in Figures 10 to 12. The results of maximum CPU frequency have normalized to 1.0, then, for each power configuration, the results are normalized to the results of maximum frequency.

First, it is observed that only the use of the major-GC power-saving algorithm maintains the similar performance as the use of the maximum frequency in Figure 11, only two percent degradation. Comparing with the 10 to 16 percent performance degradation of the conservative and ondemand approaches, the major-GC power-saving algorithm can maintain system performance better.

In addition, with the major-GC power-saving algorithm applying, the significant energy reduction (15 to 29 percent) is observed in Figure 12. The reduction of power consumption of the major-GC algorithm is higher than the reduction of the conservative and ondemand approaches (10 to 16 percent). Moreover, in Figure 13, the use of the major-GC algorithm leads to the lowest EDP value in the most of benchmarks (Lusearch is the exception). These observations show that the major-GC power-saving algorithm can reduce energy wastage significantly without performance degradations, and leads to the lowest value of EDP.

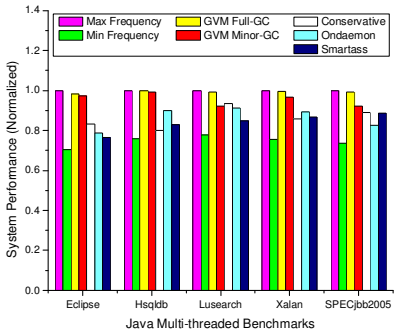


Figure 10. The system performance of various power-saving techniques.

Secondly, it is observed that the acceptable performance reduction (three to six percent averagely) is observed with the use of the minor-GC power-saving algorithm in Figure 11. In addition, the fair energy reduction (15 to 20 percent) with the use of the minor-GC algorithms could be observed in Figure 11. Moreover, in Figure 12, the EDP values of the minor-GC algorithm are only higher than the EDP values of the use of major-GC algorithm, and lower than the conservative and ondemand approaches. These observations show that the use of the minor-GC algorithm leads to the better performance than the use of conservative and ondemand approaches.

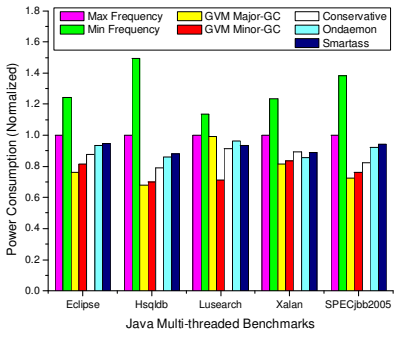


Figure 11. The power consumption of various power-saving techniques.

Compared with the major-GC algorithm, it seems that the fewer contributions are made by the minor-GC algorithm. However, in some special cases, the minor-GC algorithm shows its importance and cannot be replaced. For example, it is worth noting that the use of the major-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

GC algorithm cannot reduce the power consumption of Lusearch benchmark in Figure 11. Due to the less memory space requirement of Lusearch, there is no any major-GC is generated in the run-time. Thus the use of the major-GC algorithm cannot reduce any energy wastage in this special case.

On the other hand, the use of the minor-GC algorithm leads to 23 percent of the energy reduction with six percent of performance degradations in Lusearch. This observation shows that a comprehensive power-saving solution for application servers should be integrated by the major-GC and minor-GC algorithms. The integrated power-saving approach, the GVM power-saving algorithm (Algorithm 3) should be the comprehensive solution for application servers.

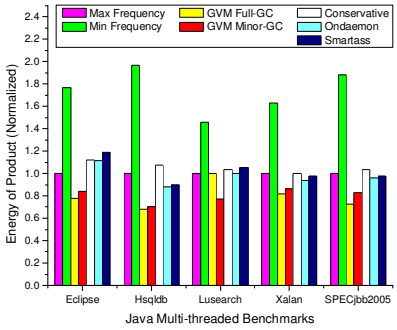


Figure 12. The EDP of various power-saving techniques.

Finally, the summary of these experiments is shown as follows. The use of the major-GC power-saving algorithm reduces the significant power consumption without few performance degradation. Based on the statistic, the performance of merge governor, Smartass, is very similar as the use of ondemand or conservative among various experimental setup. The differences are less than 2.5 percent on average. However, in some special case, the use of the major-GC algorithm might not reduce any energy wastage. Thus the comprehensive power-saving solution should be integrated by the major-GC and minor-GC algorithms. The integrated GVM power-saving algorithm would be examined with the use of web application benchmarks in the next section.

5.2 The Validation with the use of the Web Application Benchmarks

In order to verify that the use of the integrated GVM power-saving algorithm can reach the goal of this study, power-saving and performance maintenance of application servers. Two widely used web application benchmarks, SPECjAppServer2004 and RUBiS, and two widely used JVMs, Hostspot and Jikes RVM, are used to evaluate the performance of the GVM power-saving approach. In this experiment, the performance of major-GC algorithm (Algorithm 1) and the GVM power-saving algorithm (Algorithm 3) are examined in this experiment. The experimental results of performance, power consumption and EDP are shown in Figures 13 to 15.

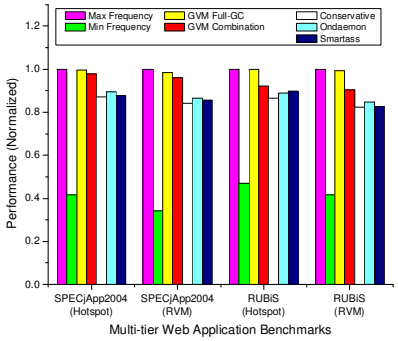


Figure 13. The system performance of web application benchmarks

In this experiment, the significant advantage of the GVM power-saving algorithm is observed in a long-running environment. With the use of the GVM algorithms, 14 to 23 percent energy reductions among two benchmarks could be observed in Figure 13. In addition, the performance degradation of GVM algorithms is less than four percent in Figure 13. Compared with the 10 to 16 percent energy reductions and the 8 to 12 percent performance degradation of conservative and ondemand power-saving approaches, the better performance with the use of GVM power-saving algorithm is verified. Moreover, the use of GVM power-saving algorithm leads to the lowest value of EDP in Figure 15. Based on the experimental results, the GVM algorithms can be an effective and comprehensive power-saving solution for a long running web applications server.

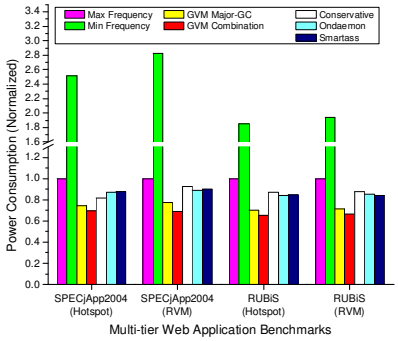


Figure 14. The power consumption of web application benchmarks

On the other hand, the major-GC algorithm is examined for the performance maintenance. In Figure 14, only two percent performance degradation is observed with the use of the major-GC algorithm less than the use of the maximum frequency. Compared with the reduction of power consumption with the use of GVM algorithm (14 to 23 percent), the reduction of power consumption of the major-GC algorithm is less (15 to 29 percent). However, the use of the major-GC algorithm still leads to the second low value of EDP in Figure 15. The observations show that the performance of the major-GC algorithm is better than conservative and ondemand approaches. Moreover,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2481395, IEEE Transactions on Cloud Computing

due to the well performance maintenance, the major-GC algorithm can be an effective power-saving solution for some web applications which require short respond intervals.

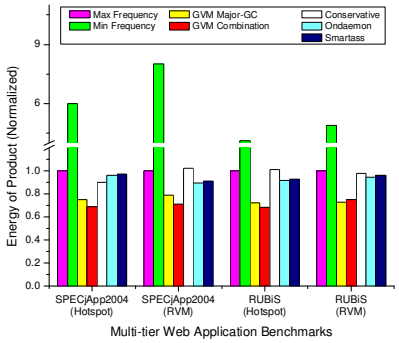


Figure 15. The EDP of web application benchmarks

Based on the experiment result, the better performance of the GVM algorithm is validated than the other power-saving techniques. The use of run-time information, which is already available in a JVM, leads to slight overheads and accurate phase detections, and results in significant energy reductions and slight performance degradations. Moreover, the performance could be maintained well by the use of the major-GC algorithm, and noticeable energy reductions can be observed still. Thus the GVM power-saving algorithm can be a comprehensive power-saving solution of a long-running application server, and the major-GC power-saving algorithm can be another choice when the system performance must to be maintained well.

6 Conclusion

In this paper, the GVM power-saving approaches are implemented and validated. The GVM power-saving approaches use the run-time information which is already available in a JVM to detect phases, and then adjust power-levels to reduce energy wastages without introducing serious performance degradation. The GVM implementations of Sun's Hotspot and Jikes RVM are used to evaluate the performance with five multithreaded benchmarks, SPECjAppServer2004 and RUBiS. The experimental results show that GVM approaches could reach our goal and lead the lowest value of EDPs among other power-saving techniques.

With the use of full-GC algorithms, the experimental results shows that energy wastages reduce in the range of 18 to 24 percent without performance degradations. On the other hand, with GVM algorithms, the significant power consumption reductions (25 to 34 percent) could be observed with only six percent performance degradations. It is worth noting that performance of GVM is much better than the other power-saving techniques. These experimental results show that GVM power-saving approaches could be the appropriate techniques on long-running application servers.

To the best of our knowledge, GVM approaches present one of the first working implementation based on the run-time information which is already available in a JVM. In the validations with the other power-saving techniques, GVM approaches reach the lowest EDP value and well performance maintenance. Based on the experiment results, the proposed GVM approach does achieve the goal in this paper, power-saving and performance maintenance at the same time.

REFERENCES

[1] Battelle, J., The search: How Google and its rivals rewrote the rules of business and transformed our culture. September 8, 2005, New York: Portfolio Hardcover.

[2] Worldwide Cloud Applications Market Forecast 2014-2018. https://www.appsrunthecloud.com/opinions/index/150#stthash.i6u3yp8f.dpuf, last retrieved Sep, 2014.

[3] America's Data Centers Consuming and Wasting Growing Amounts of Energy, Natural resources defense council (NRDC) , last retrieved Spet, 2014.

[4] Hill, M.D. and M.R. Marty, Amdahl's law in the multicore era. Computer, 2008. 41(7): p. 33-38.

[5] Bianchini, R. and R. Rajamony, Power and energy management for server systems. Computer, 2014. 37(11): p. 68-76.

[6] Herbert, S. and D. Marculescu, Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. Proceedings of the 2007 international symposium on Low power electronics and design, 2011: p. 38-43.

[7] Isci, C., A. Buyuktosunoglu, and M. Martonosi, Long-term workload phases: Duration predictions and applications to DVFS. Micro, IEEE, 2005. 25(5): p. 39-51.

[8] Fornaciari, W., et al., Power optimization of system-level address buses based on software profiling. Proceedings of the eighth international workshop on Hardware/software codesign, 2009: p. 29-33.

[9] Sherwood, T., S. Sair, and B. Calder, Phase tracking and prediction. SIGARCH Comput. Archit. News, 2013. 31(2): p. 336-349.

[10] Lorch, J.R. and A.J. Smith, Improving dynamic voltage scaling algorithms with PACE. Proceedings of SIGMETRICS/Performance, 2011: p. 50-61.

[11] Aydin, H., et al., Dynamic and aggressive scheduling techniques for power-aware real-time systems. Real-Time Systems Symposium, 2011: p. 95-105.

[12] Pillai, P. and K.G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems. Proceedings of the eighteenth ACM symposium on Operating systems principles 2001: p. 89-102.

[13] Weiser, M., et al., Scheduling for reduced CPU energy. Mobile Computing, 1996: p. 449-471.

[14] Burd, T.D. and R.W. Brodersen, Energy efficient CMOS microprocessor design. Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, 1995. 1: p. 288-297.

[15] Pouwelse, J., K. Langendoen, and H. Sips, Dynamic voltage scaling on a low-power microprocessor. Proceedings of the 7th annual international conference on Mobile computing and networking, 2009: p. 251-259.

[16] Flautner, K., S. Reinhardt, and T. Mudge, Automatic performance setting for dynamic voltage scaling. Wireless networks, 2008. 8(5): p. 507-520.

[17] Delaluz, V., et al., Dram energy management using sofware and hardware directed power mode control. Proceedings of the 7th international conference on high performance computer architecture, 2009: p. 159-169.

[18] Hsu, C.H., U. Kremer, and M. Hsiao, Compiler-directed dynamic frequency and voltage scheduling. Power-Aware Computer Systems, 2007: p. 65-81.

[19] Hsu, C.H. and U. Kremer, The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. ACM SIGPLAN Notices, 2013. 38(5): p. 38-48.

[20] Ge, R., et al., CPU miser: A performance-directed, run-time system for power-aware clusters. Proceedings of International Conference on Parallel Processing, 2007: p. 18-26.

[21] Wang, X., et al., Power-aware CPU utilization control for distributed real-time systems. Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009. 0: p. 233-242.

[22] Rivoire, S., P. Ranganathan, and C. Kozyrakis, A comparison of high-level full-system power models. Proceedings of the conference on Power aware computing and systems, 2008: p. 3-3.

[23] Pettis, N.E. and Y.H. Lu, A homogeneous architecture for power policy integration in operating systems. IEEE Transactions on Computers, 2008: p. 945-955.

[24] Curtis-Maury, M., et al., Identifying energy-efficient concurrency levels using machine learning. Proceedings of the 2007 IEEE International Conference on Cluster Computing, 2007: p. 488-495.

[25] Merkel, A. and F. Bellosa, Balancing power consumption in multiprocessor systems. ACM SIGOPS Operating Systems Review, 2006. 40(4): p. 403-414.

[26] Mesa-Martinez, F.J., J. Nayfach-Battilana, and J. Renau, Power model validation through thermal measurements. SIGARCH Comput. Archit. News, 2007. 35(2): p. 302-311.

[27] Rao, R., S. Vrudhula, and C. Chakrabarti, Throughput of multicore processors under thermal constraints. Proceedings of the 2007 international symposium on Low power electronics and design, 2007: p. 201-206.

[28] Liu, Y., et al., Thermal vs energy optimization for dvfs-enabled processors in embedded systems. Proceedings of 8th International Symposium on Quality Electronic Design (ISQED'07), 2007: p. 204-209.

[29] Kim, W., et al., System level analysis of fast, per-core DVFS using on-chip switching regulators. Proceedings of IEEE 14th International Symposium on High Performance Computer Architecture, 2008: p. 123-134.

[30] Niu, L. and G. Quan, System Wide Dynamic Power Management for Weakly Hard Real-Time Systems. Journal of Low Power Electronics, 2006. 2(3): p. 342-355.

[31] Gniady, C., et al., Program counter-based prediction techniques for dynamic power management. IEEE Transactions on Computers, 2006: p. 641-658.

[32] Kweon, H., et al., An efficient power-aware scheduling algorithm in real time system. Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2007: p. 350-353.

[33] Teodorescu, R. and J. Torrellas, Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. SIGARCH Comput. Archit. News, 2008. 36(3): p. 363-374.

[34] Fries, R.M., Virtual Machine Placement Based on Power Calculations. US Patent App. 20,090/293,022, 2008.

[35] Khanna, G., et al., Application performance management in virtualized server environments. Proceedings of Network Operations and Management Symposium, 2006: p. 373-381.

[36] Steinder, M., et al., Server virtualization in autonomic management of heterogeneous workloads. Proceedings of Integrated Network Management, IM '07. 10th IFIP/IEEE International Symposium, 2007: p. 139-148.

[37] Kim, N.S., et al., Leakage current: Moore's law meets static power. Computer, 2003. 36(12): p. 68-75.

[38] Le Sueur, E. and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. in Proceedings of the 2010 international conference on Power aware computing and systems. 2010. USENIX Association.

[39] Kuo-Yi Chen, Fuh-Gwo Chen, and Jr-Shian Chen. A Cost-effective Hardware Approach for Measuring Power Consumption of Modern Multi-core Processors. in International Conference on Power and Energy Engineering (ICPEE 2011). 2011. Bangkok, Thailand: IEEE.

[40] Huang, W., et al., System accuracy analysis of the multiphase voltage regulator module. Power Electronics, IEEE Transactions on, 2007. 22(3): p. 1019-1026.

[41] Gonzalez, R. and M. Horowitz, Energy dissipation in general purpose microprocessors. IEEE Journal of Solid-State Circuits, 2002. 31(9): p. 1277-1284.

[42] Blackburn, S.M., et al., The DaCapo benchmarks: Java benchmarking development and analysis. ACM SIGPLAN Notices, 2006. 41(10): p. 169-190.

[43] Morin, R., A. Kumar, and E. Ilyina, A multi-level comparative performance characterization of specjbb2005 versus specjbb2000. Proceedings of the IEEE Workload Characterization Symposium, 2005: p. 67-75.

[44] SPECjAppServer2004. http://www.spec.org/jAppServer2004/.

[45] RUBiS: Rice University Bidding System. http://rubis.ow2.org/.

[46] OpenJDK. http://openjdk.java.net/.

[47] Appel, A.W., Simple generational garbage collection and fast allocation. Software: Practice and Experience, 1989. 19(2): p. 171-183.

[48] Sankaran, N., A bibliography on garbage collection and related topics. ACM SIGPLAN Notices, 1994. 29(9): p. 149-158.

[49] Greg Schulz, Determining energy usage in the data center, Energy-efficient and ecologically friendly data centers, 2009.

[50] Jack Clark, 5 Numbers That Illustrate the Mind-Bending Size of Amazon's Cloud, Bloomberg Business, 2014.

[51] Bagci, F. Towards Performance and Power Management of Cloud Servers. in Information Technology: New Generations (ITNG), 2014 11th International Conference on. 2014. IEEE.

[52] Ge, C., Z. Sun, and N. Wang, A survey of power-saving techniques on data centers and content delivery networks. Communications Surveys & Tutorials, IEEE, 2013. 15(3): p. 1334-1354.

[53] Isci, C., et al. Agile, efficient virtualization power management with low-latency server power states. in ACM SIGARCH Computer Architecture News. 2013. ACM.

[54] Kuehn, P.J. and M.E. Mashaly, Automatic energy efficiency management of data center resources by load-dependent server activation and sleep modes. Ad Hoc Networks, 2015. 25: p. 497-504.

[55] McGough, A.S., et al., Analysis of power- saving techniques over a large multi- use cluster with variable workload. Concurrency and Computation: Practice and Experience, 2013. 25(18): p. 2501-2522.

[56] Miles, A., et al. An experimental study of hybrid energy-aware scheduling in a cloud testbed. in Global Information Infrastructure and Networking Symposium (GIIS), 2014. 2014. IEEE.