CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **00**:1–18
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Developing Power-aware Scheduling Mechanisms for Virtualized Environments

Shenyuan Ren[1], Ligang He[*], Huanzhou Zhu[1], Zhuoer Gu[1], Wei Song[2] and Jiandong Shang[3]

[1] *Department of Computer Science, University of Warwick,United Kingdom*
Email: S.Ren@warwick.ac.uk, zhz44@dcs.warwick.ac.uk, Zhuoer.gu@dcs.warwick.ac.uk
[2]*School of Information Engineering, Zhengzhou University, Email:iewsong@zzu.edu.cn*
[3]*Zhengzhou Research Institute of Smart City, Zhengzhou University, Email:sjd@zhengzhou.gov.cn*

SUMMARY

Cloud computing emerges as one of the most important technologies for interconnecting people and building the so-called ”Internet of People” (IoP). In such a Cloud-based IoP, the virtualization technique provides the key supporting environments for running the IoP jobs such as performing data analysis and mining personal information. Nowadays, energy consumption in such a system is a critical metric to measure the sustainability and eco-friendliness of the system. This paper develops three power-aware scheduling strategies in virtualized systems managed by Xen, which is a popular virtualization technique. These three strategies are the Least performance Loss Scheduling (LLS) strategy, the No performance Loss Scheduling (NLS) strategy and Best Frequency Match (BFM) scheduling strategy. These power-aware strategies are developed by identifying the limitation of Xen in scaling the CPU frequency and aim to reduce the energy waste without sacrificing the jobs' running performance in the computing systems virtualized by Xen. LLS works by re-arranging the execution order of the VMs. NLS works by setting a proper initial CPU frequency for running the VMs. BFM reduces energy waste and performance loss by allowing the VMs to jump the queue so that the VM that is put into execution best matches the current CPU frequency. Scheduling for both single core and multi-core processors are considered in this paper. The evaluation experiments have been conducted and the results show that compared with the original scheduling strategy in Xen, the developed power-aware scheduling algorithm is able to reduce energy consumption without reducing the performance for the jobs running in Xen.

Received . . .

KEY WORDS: Power-aware scheduling, Virtualization, Cloud computing

1. INTRODUCTION

Cloud computing emerges as one of the most important technologies for interconnecting people and building the so-called 'Internet of People' (IoP). Nowadays, energy consumption in such a system is a critical metric to measure its sustainability and eco-friendliness. Indeed, data centers used by the Cloud service providers have become one of the fastest growing sources of power consumption in industry. According to IDC (International Data Corporation), power consumption of data centers worldwide accounts for about 8% of the global electricity, which does not include the additional electricity consumed by the cooling systems equipped in the data center. In such a Cloud-based IoP, the virtualization technique, which allows multiple operating system instances (i.e., Virtual

[*]Correspondence to: Department of Computer Science, University of Warwick, United Kingdom
Email:liganghe@dcs.warwick.ac.uk

Machines) to run simultaneously in a physical machine, provides the key supporting environments for running the IoP jobs such as performing data analysis and mining personal information.

Xen [2] is a popular virtualization hypervisor used in the academic community. It has also been widely deployed in a number of industry-level Clouds, such as AWS (Amazon Web Service), Rackspace, Verizon, etc. SEDF (Simple Earliest Deadline First) is a scheduler in Xen. In SEDF, the CPU requirement for each VM is specified by a tuple (s,p,x), in which s and p designate that the VM has to run at least s in a period of p. This CPU requirement can be translated to the deadlines by which a VM has to start running (otherwise, the CPU requirement will not be met). In each scheduling round, SEDF puts the VM with the earliest deadline into execution [5].

Dynamic Voltage Frequency Scaling (DVFS) is a power management technique. DVFS can change the running frequency of CPU dynamically as required and therefore reduce power consumption when the tasks do not need to be run with the maximum CPU frequency. Xen currently has four power governors. 1) The Ondemand governor selects the CPU frequency which best fits the VM (guest domain). 2) The Userspace governor selects the frequency specified by user. 3) The Performance governor selects the highest frequency. 4) The Powersave governor selects the lowest frequency. There are twelve frequency states in the Xen power management, in which the state P0 represents the highest frequency while the state P12 represents the lowest frequency. In this paper, we consider the most complex scenario and assume that the guest domains are run under the Ondemand governor, namely, the CPU frequency is dynamically adjusted towards the best execution frequency of a guest domain.

In Xen, the CPU frequency can only be changed by one state in every interval of 10ms. The interval of 10ms is called the *frequency scaling slice*. For example, it takes Xen 40ms to change the CPU frequency from P1 to P4. Our studies show that this limitation in Xen in frequency changing may cause the energy waste and performance loss (The problem is illustrated by an example in Subsection 2), which this paper aims to reduce. In this work, we conduct the theoretical ananlysis and construct the performance model and the energy consumption model by taking into account the feature of Xen in frequency changing. Based on the analysis and the models, we derive the condition under which the best performance can be achieved, i.e., there is no performance loss caused by the limitation of Xen in frequency changing. Further, we propose a frequency-aware scheduling policy, called BFM (Best Frequency Match), by adapting the SEDF scheduling policy in Xen. Compared with SEDF, BFM is able to reduce the power consumption of running VMs without violating their CPU requirements.

The reminder of this paper is organized as follows. In Section 2, an example is presented to illustrate the problem of energy waste and performance loss due to the limitation of Xen in changing CPU frequencies. Section III reviews the related work. We present the energy consumption model and the performance model in Section IV. In Section V, we derive the situaltion where the best performance scheduling can be achieved. In Section VI, we propose the frequency-aware scheduling policies for single-core processors and multi-core processors. The experiment results are presented in Section VII. Finally, Section VIII concludes this paper and present the plan for future work.

2. A MOTIVATING EXAMPLE

As introduced in Section 1, under the OnDemand governor, DVFS is used to adjust the CPU frequency on demand. We ran an experiment on a quad-core machine to record the frequency of the CPU core which Domain0 runs on. We pinned two vcpus of Domain0 to core 3 and recorded the frequency of the core once every 300 ms for 60 times. The results are shown in Fig. 1. It can be seen that Domain0 does not always run with the highest frequency of 2301 MHz. Indeed, under the Ondemand governor, different tasks may run with different execution frequencies. For example, Fio, which is a I/O benchmarking tool, ran at the lowest scaling frequency (1200 MHz), while the computation-intensive benchmark BT (a benchmark application in the NAS Parallel benchmark) ran at the highest frequency, 2301 MHz.

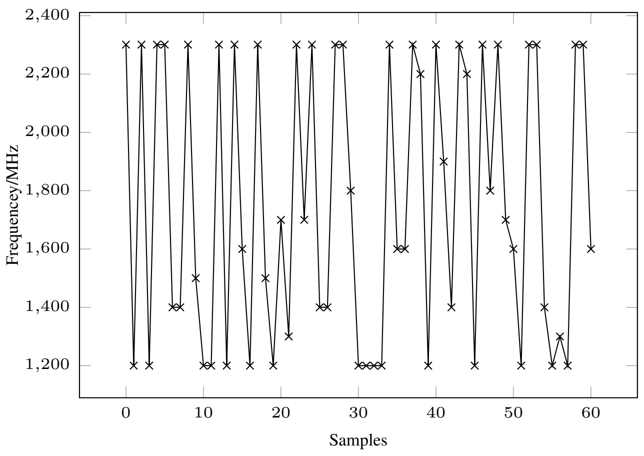


Figure 1. Frequency sampling of Domai0 for 60 times; each sampling interval is 300ms

The following illustrate the problem of energy waste and performance loss caused by the limitation of Xen in changing the CPU frequency. Fig. 2 shows the changes of Power-states (P-states) when four VMs (VM1-VM4) are running on a single core. The x axis is the elapsed time, while the y axis is the Power-state of the core at the corresponding time point.. The time slice of a VM (namely the time duration for which a VM runs continuously before the Xen hypervisor jumps in and schedule another VM into execution.) is 30ms by default. The running order of the VMs in the experiment is also labelled in the figure. The power governor checks and changes, if necessary, the CPU frequency every 10ms by at most one level [11]. Assume that the P-states demanded by VM1, VM2, VM3 and VM4 are P1, P9, P3 and P7, respectively. In this example, the initial P-state is set to be P3. When the current frequency deviates from (higher or lower than) the best frequency of a running VM, the power governor adjusts the frequency towards the best frequency. However, it can only adjust the frequency by one level every 10ms due to the feature of Xen in frequency changing. With this restriction, the actual CPU frequencies over time are those highlighted by the bold black line. When a VM runs on a frequency higher or lower than its best execution frequency, Energy waste or performance loss occurs. The difference between the best frequency and the actual frequency represents the amount of energy waste or performance loss. In this figure, the red area and the shadowed grey area represent the amount of energy waste and performance loss, respectively. For example, at time 0, the VM (VM4) only requires P5 (the best P-state), while the current actual frequency is P3 (initial P-state). Since the current P-state is higher than the best P-states of the VM, the frequency is adjusted down by one level every 10ms until it reaches the best frequency or the VM is scheduled out after its time slice of 30ms is used up. In the case of VM4, the VM is scheduled out before the actual frequency reaches the best frequency. VM3 is scheduled in after VM4. Since the best frequency of VM3 is P3, which is higher than the current running frequency, the frequency is adjusted up. Since the frequency can only be adjusted by one level every 10ms, VM3 still runs at a frequency lower than its best frequency in the first 10ms of VM3's time slice, which leads to the performance loss (indicated by the shadowed grey area) and will consequently increase the execution time of the application that is running in the VM. In the remaining time slice, VM3 runs perfectly at its best frequency.

The above example suggests that the feature of Xen in adjusting the CPU frequency may cause both energy waste and performance loss. The objective of this work aims to improve the situation. We adapt the default SEDF scheduling policy so as to minimize the energy waste and performance loss under DVFS.

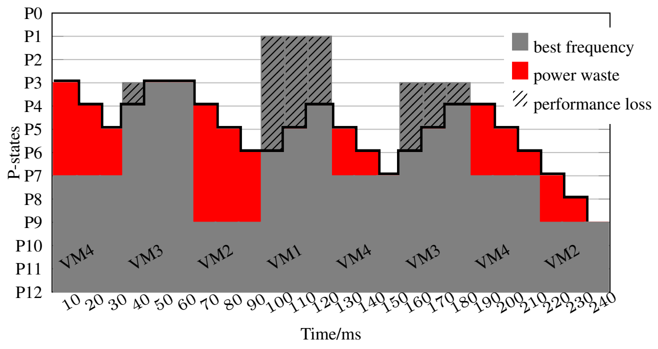


Figure 2. Energy waste and performance loss under DVFS in Xen

3. RELATED WORK

There have been many prior efforts to optimize the performance and the energy consumption for scheduling and running tasks in native and virtualized computing systems [6] [22] [1] [9] [16] [18] [20] [1] [19] [14] [15]. This section discusses the related work in different aspects, including performance and power consumption models, energy-aware scheduling and VM scheduling.

The work in [9] proposes an energy-aware strategy for scheduling stochastic tasks in heterogeneous computing environments. The work takes the performance and energy budget into account. The work in [17] [3] and [4] presents power consumption models in different computing environments.

The work in [8] considers scheduling a set of aperiodic tasks on DVFS-enabled multi-core processors. It attempts to meet the execution requirements of all the tasks and minimize the overall energy consumption on the processor. The work introduces the concept of Desired Execution Requirement for tasks and proposes an balanced allocation method to save processor energy.

A task scheduling strategy is presented in [10] to tackle the task allocation problems on DVFS-enabled CPU cores, the execution order of tasks and the CPU processing frequency of the each task. It presents the task scheduling model, the energy consumption model, the CPU frequency model and a cost function for capture the scheduling scenario. Both batch-mode and online-mode tasks are considered. The developed algorithm guarantees the minimal total cost for every time interval. The work does not consider virtualized systems, but is applied to native computing environments.

The work in [16] considers the power-aware scheduling problem in Cluster systems. The performance model is proposed to capture the task scheduling and executions in DVFS-enabled virtualized clusters tasks.

In [7], a power consumption model is proposed to estimate the energy consumption of the tasks in cloud systems. It divides the power consumption into leakage power and dynamic power and formulates the energy consumption of processors according to each in-processor event. An energy-credit scheduler is proposed to schedule the tasks according to their energy budgets instead of time credits.

The work in [13] extends the existing formulation of the power-aware job placement problem proposed in the literature, so that it can be adapted to DVFS-enabled cluster nodes. Two optimization problems are investigated in the work: (i) performance optimization given the constraint on energy consumption, and (ii) optimization of energy consumption given the constraint of job performance. In addition, it calculates the performance bound for several instantiations of the DVFS model, aiming to quantify the added benefit of the increased DVFS capabilities.

The work presented in our paper analyzes the energy consumption and designs the energy-aware scheduling strategies by taking into account the feature (limitation) of Xen in adjusting the CPU frequency, which, to our best knowledge, has not been investigated before in literature.

4. PERFORMANCE AND ENERGY MODEL FOR THE DVFS-ENABLED XEN

4.1. Performance Model

Assume a set of independent tasks T = $\{T_1, T_2, ..., T_n\}$. Task T_i runs in VM_i. f_i denotes the best frequency of VM_i. t_i denotes the execution time of task T_i when VM_i runs at the frequency of f_i (T_i is executed in VM_i). P_s denotes the scheduling time slice, which is 30ms by default, while P_f denotes the frequency scaling slice, which is 10ms. When a VM runs at a frequency, f_i', lower than its best frequency f_i, the actual work completed for the task in a time unit will be less than that when the VM runs at its best frequency. Let $c(f_i')$ denote the equivalent execution rate of T_i when VM_i runs at the frequency f_i'. $c(f_i')$ can be calculated by Eq. 1, where $F_t(f_i')$ is the function of execution time over CPU frequency.

$$c(f_i') = \begin{cases} \frac{t_i}{F_t(f_i')} & if f_i' < f_i, \\ 1 & if f_i' \geq f_i. \end{cases} \tag{1}$$

$f_i'(j)$ denotes the frequency which task T_i runs at in the jth time interval. $c(f_i'(j))$ denotes the execution rate of T_i in the jth time interval. InEq. 2 can be used to determine the number of intervals that VM_i uses to complete the execution of task T_i, which is the minimal value of m that satisfies InEq. 2.

$$\sum_{j=0}^{m} c(f_i'(j))P_f \geq t_i \tag{2}$$

Then the total execution time of task T_i when it is not always running at its best frequency f_i (denoted by t_i') can be modelled as Eq. 3, where m is the minimal value that satisfies InEq. 2.

$$t_i' = \sum_{j=0}^{m} c(f_i'(j))P_f \tag{3}$$

4.2. Power Consumption Model

When task T_i runs at the frequency of f_i', Eq. 4 is the classic equation to calculate the power consumption rate of CPU for running T_i [12] (denoted by r_i), where C is the capacitance being switched per clock cycle, V is the voltage, A is the activity factor indicating the average number of switching events undergone by the transistors in the chip and f_i' is the frequency.

$$r_i(f_i') = A \times C \times V^2 \times f_i' \tag{4}$$

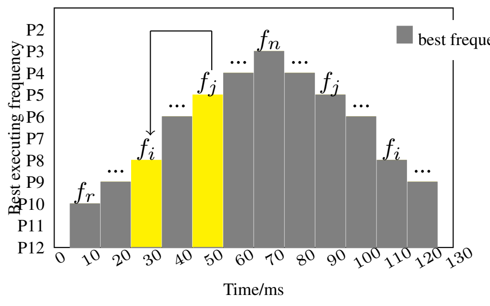
InEq. 2 has been used to determine the number of intervals that VM_i uses to complete the execution of task T_i. The total power consumption of task T_i, denoted by e_i, can be modelled by Eq. 5, where $f_i'(j)$ is the frequency which task T_i runs at in the jth time interval, same as in Eq. 2.

$$e_i = \sum_{j=0}^{m} r_i(f_i'(j))P_f \tag{5}$$

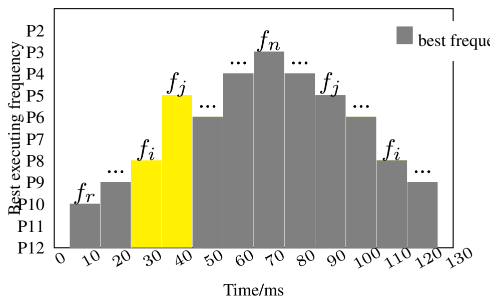
5. SCHEDULING STRATEGIES

5.1. The Scheduling Strategy with Least Performance Loss

The power management in Xen can only adjust the power state by at most one level every 10ms. The frequency gap between the current CPU frequency f and the task T_i's best (desired) executing frequency f_i will lead to either performance loss or energy waste. When the current CPU frequency f is lower (or higher) than T_i's best executing frequency, f_i, and the power management cannot increase (or reduce) f to f_i immediately, performance loss (or energy waste) occurs.



(a) "Least Performance Loss" Scheduling Strategy



(b) Illustration of changing the execution position of a randomly selected task

Figure 3. "Least Performance Loss" Scheduling Strategy & Illustration of changing the execution position of a randomly selected task

Given the current CPU frequency and a set of tasks, Theorem 1 gives the "Least Performance Loss" Scheduling strategy (LLS), namely the execution order of the tasks that leads to the least performance loss.

Theorem 1
Given a set of tasks, $T = \{ T_1, T_2, ..., T_n\}$, the best CPU frequency of T_i is f_i and the set of tasks are run in a time-sharing manner. Assume $f_1 \leq f_2 \leq ... \leq f_n$. If the current CPU frequency is f, then given the current CPU frequency, the LLS strategy (i.e., the execution order that leads to the least performance loss for the set of tasks) is to run the tasks in the following order, where T_r's frequency f_r is the highest frequency that is less than the current frequency f.

$T_r, T_{r+1}, ..., T_{n-1}, T_n, T_{n-1}, ..., T_{r+1}, T_r, T_{r-1}, ..., T_2, T_1, T_2, ..., T_n, T_{n-1}, ..., T_1$ ...

Namely, the execution order is to start from T_r, go up to T_n in the increasing order of frequency and then come down to T_1 in the decreasing order of frequency, and that the upward and downward execution pattern in terms of frequency repeat until all tasks have been completed.

Proof
The performance loss is related to the frequency gap of the tasks during the execution. Performance loss increases with the increase in the frequency gap. We prove this theorem by proving that any change in the execution order from the LLS strategy will lead to the increase of the frequency gap, thus the performance loss.

We randomly change the execution position (specified by the LLS strategy) of a randomly selected task. Assume task T_j is moved to the position after task T_i. Without the loss of generality, we assume $j > i + 1$, (i.e., we move T_j forwards as shown in Fig. 3a and Fig. 3b.

Before the change, the frequency gap among the relevant tasks (i.e. task T_i, T_{i+1}, T_{j-1}, T_j, T_{j+1}) is:

$$E = (f_{i+1} - f_i) + (f_j - f_{j-1}) + (f_{j+1} - f_j)$$
$$= f_{i+1} - f_i - f_{j-1} + f_{j+1}.$$

After the change, the frequency gap among the involved tasks is:
$E' = (f_j - f_i) + (f_{j+1} - f_{j-1})$.

Note that the gap of $f_j - f_{i+1}$ is not counted in the expression above since it does not cause performance loss (but energy waste) even if Xen cannot adjust the frequency timely from f_j to f_{i+1}.

The difference between E' and E is:
$E' - E = (f_j - f_i) + (f_{j+1} - f_{j-1}) - (f_{i+1} - f_i - f_{j-1} + f_{j+1})$
$= f_j - f_{i+1}$.

Since $f_j \geq f_{i+1}$, we get $E' \geq E$.

In the similar way, we can prove the theorem also holds when $i > j$ (i.e., moving the task backwards). Therefore, given the current CPU frequency, the LLS strategy generates the least performance loss for a set of tasks.

□

5.2. *The Scheduling Strategy with No Performance Loss*

The previous section derives LLS, the scheduling strategy with the least performance loss, given the current CPU frequency. LLS requires re-arranging the execution order of the VMs. In this section, we will derive the scheduling strategy under which there is no performance loss for a set of tasks. We call this strategy the No performance Loss Scheduling (NLS) Strategy. NLS aims to ensure all tasks run with the frequencies no less than their best frequencies. NLS does not reorder the VMs' execution. The VMs can be executed in the order of their positions in the run-queue. Rather, NLS calculates the initial CPU frequency that the CPU needs to be set with in order for all VMs to run without performance loss.

According to the Xen power management policy, the execution frequency can be modified once every 10ms, which we call the *frequency scaling slice*. The default time slice, which we call the scheduling slice, for running a VM in Xen is 30ms. After 30ms, the Xen hypervisor jumps in and schedule another VM to run. Therefore, the frequency can be changed three times at most in each scheduling slice. As shown in Fig.4, $f_k(j)$,$f_k(j+1)$,$f_k(j+2)$ indicates the three execution frequencies of task T_k in the three frequency scaling slices (indexed as j, $j+1$ and $j+2$ in the example of Fig. 4) in T_k's scheduling slice. To ensure that task T_k can execute with at least its best frequency, the frequency of the frequency scaling slice before $f_k(j)$ (i.e., the $(j-1)$-th frequency scaling slice) should be at least $f_k(j) - \Delta f$, where Δf is the frequency that can be changed at most each frequency scaling slice (which is 100 MHz, i.e., one P-state, in Xen). $(j-1)$-th frequency scaling slices falls in the scheduling slice of task T_{k-1}, which means that the execution frequency that T_{k-1} has to run with, denoted by $f_{k-1}(j-1)$, has to be $(f_k(j) - \Delta f)$ even if T_{k-1} does not need such a high running frequency. In Fig. 4, the best frequency of task T_{k-1}, i.e., f_{k-1} is shown by the yellow bar, which is lower than $f_{k-1}(j-1)$. Similarly, $f_{k-1}(j-2)$ has to be $f_{k-1}(j-1) - \Delta f$ or f_{k-1} (T_{k-1}'s best frequency), whichever is higher.

In general, assuming that T_k has the highest best frequency in the set of tasks (i.e., f_k is highest) and that j, $j+1$ and $j+2$ are the three frequency scaling slices in T_k's first scheduling slice during the running of the set of tasks, Eq. 6 can be used recursively, starting from $i = j$, to calculate the running frequency in each frequency scaling slice before j-th frequency scaling slice (it is obvious that $f_k(j)$, $f_k(j+1)$ and $f_k(j+2)$ should all be f_k), so that all VMs can run without performance loss, i.e., with the frequencies no less than their corresponding best frequencies.

The algorithm for performing the recursive calculation is outlined in Algorithm 1. The output of Algorithm 1 is the value of $f_1(1)$, i.e., the starting frequency that the CPU has to be set with in order for the set of tasks to run without performance loss.

Note that although NLS guarantees no performance loss, it may cause energy waste. The shadowed areas above the coloured bars in Fig. 4 represent the energy waste.

$$f_{k'}(i-1) = max(f_k(i) - \Delta f, f_{k'}) \tag{6}$$

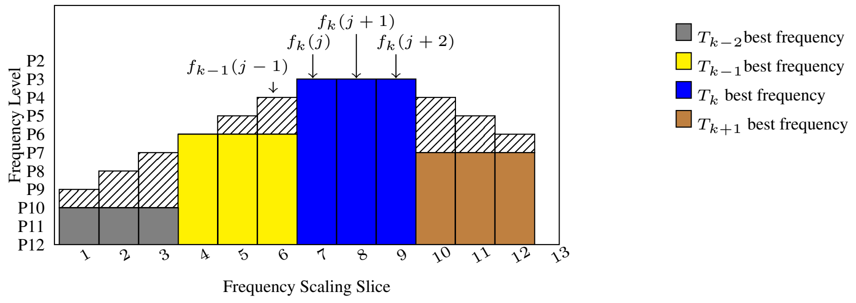


Figure 4. ”No Performance Loss” Scheduling Strategy

where,

$$k' = \begin{cases} k & \text{if } (i-1)\text{-th frequency scaling slice is in } T_k\text{'s scheduling slice} \\ k-1 & \text{if } (i-1)\text{-th frequency scaling slice is in } T_{k-1}\text{'s scheduling slice} \end{cases} \tag{7}$$

Algorithm 1 No performance loss scheduling strategy

Input: Tasks T_1,T_2,...,T_n in the run-queue, whose best running frequencies are f_1,f_2,...,f_n; T_K is the task with the highest best frequency f_K; j is the index of the first frequency scaling slice in T_K's scheduling slice
Output: $f_1(1)$;

```
$k = K, f_k(j) = f_K$;
for $(i = j; i \geq 2; i--)$ do
  if $(j-1)th$ frequency scaling slice is in $T_k$'s scheduling slice then
    $k' = k$
  else
    $k' = k - 1$;
  end if
  $f_{k'}(i-1) = max(f_k(i) - \Delta f, f_{k'})$;
end for
```

6. BFM SCHEDULER

In section IV, we presented two scheduling strategies: The Least Performance Loss Scheduling Strategy (LLS) and No Performance Loss Scheduling Strategy (NLS).

NLS will achieve the shortest execution time since every VM will execute with a frequency equal to or higher than its best frequency. Those VMs which run with a frequency higher than its best frequency (in order to guarantee that other VMs can run with their best frequencies) will cause energy waste. Therefore, NLS is designed with maximizing the performance as the only goal. In NLS, we propose the method to determine the minimal initial frequency that the CPU has to be set with in order to guarantee that no VM will experience performance loss.

Different from NLS, LLS does not reeve up CPU frequency to guarantee there is no performance loss, but makes the best effort to reduce the performance loss by manipulating the VMs' execution order. NPS does not artificially set the initial CPU frequency for running a set of VMs, but goes along with the current CPU frequency.

In order to guarantee that every VM's deadline is met, however, the SEDF scheduler in Xen requires the VMs to be run in the order of deadline (earliest deadline first). In this execution environment, meeting the deadlines is the top priority and VMs' execution order may not be able to be adjusted in the way designated by LLS. Thus, in this section we present a power-aware SEDF scheduling strategy, called the Best Frequency Match strategy (BFM). BFM aims to make the best effort to reduce performance loss subject to respecting the principle of SEDF, i.e., meeting all VMs' deadlines.

Next, we first present BFM for single-core processors and then extend it to multi-core processors.

6.1. *BFM for Single-core Processors*

BFM aims to minimize the performance loss while satisfying the VMs' CPU requirement specified in SEDF.

Assume that a set of VMs T_i ($1 \leq i \leq n$) are in the run queue of a single core with their CPU requirement expressed as (p_i, s_i, x_i) and that task T_i's best execution frequencies is f_i. The deadline of each VM is recalculated when the current scheduling slice is finished. BFM checks the deadline and the best frequency of each VM (VCPU) in the run-queue of the CPU core. If the first VM in the run-queue (i.e., the one with the earliest deadline) has the smallest gap between its the best frequency and the current executing frequency, the VM will be scheduled. However, if there are other VMs in the queue which have smaller frequency gaps than the first VM, scheduling the first VM (with the earliest deadline) will cause either performance loss (if the current frequency is less than the best frequency) or energy waste (if the current frequency is higher than the best frequency) compared with scheduling a VM with smaller frequency gaps. Under this circumstance, BFM checks if there is any better scheduling choice in the following way.

Firstly, BFM identifies the VM, for instance T_j, whose best frequency has the smallest gap with the current CPU frequency. Before allowing task T_j to jump the queue, BFM needs to make sure that all the VMs queueing before T_j satisfy InEq. 8, where t_c is the current time while L_i is the position of task T_i in the run-queue. InEq. 8 can be understood in the following way. The scheduling slice of a VM is P_s. Task T_i, whose position in the original queue is L_i, needs to wait $L_i - 1 \times P_s$ for T_i being put into execution. After the queue jump, the waiting time of T_i becomes $L_i \times P_s$. The waiting time plus the VM's running duration, which is P_s, must be no greater than VM_i's deadline d_i, which results in InEq. 8.

$$\forall L_i < L_j \quad d_i - [t_c + (L_i + 1) * P_s] \geq 0 \tag{8}$$

If InEq. 8 can be satisfied for all the VMs before T_j, BFM allows task T_j to jump the queue. If not, BFM continues to find the VM which has the second smallest gap between its best frequency and the current frequency, and applies InEq. 8 to determine whether the queue-jumping is allowed. The process repeats until BFM finds a VM that is eligible to jump the queue or all VMs have been considered (in this case, no VMs can jump the queue and BFM schedules and run the first VM in the queue, same as the SEDF scheduler).

The pseudo code of BFM on a single core is presented in Algorithm 2.

An example is used in Fig. 5 to illustrate the working of BFM. 4 VMs are considered in this example: $T_1(20, 100)$, $T_2(5, 100)$, $T_3(10, 100)$, $T_4(30, 100)$, where the first number of the pair is the time that a VM has to run in a period, which is indicated by the second number. For example, T_1 has to run at least 20 ms in every period of 100 ms. In the beginning of each time slice, all VMs are sorted by their deadlines in the run queue. d_i presents the deadline of T_i and f_i is its best frequency of T_i. At the time point of 30 ms, T_3 has the earliest deadline, while T_2 has the smallest frequency gap with the current CPU frequency P12. Under this circumstance, BFM will check if scheduling T_3 first (i.e., allowing T_2 to jump the queue) will cause the VMs before T_2 (i.e., T_3 in this example) to miss the deadlines. In this case, it will not and therefore T_2 jumps the queue successfully. At 60 ms, T_1 has the the smallest frequency gap with the current frequency. However, it is rejected for T_1 to jump the queue, since otherwise T_3, the VM before T_1 in the queue, would miss its deadline.

6.2. *BFM for Multi-core Processors*

In this section, we extend the BFM strategy for a single core to multicore processors. In this section, we denote BFM for single core by BFMS and BFM for multicore by BFMM. Compared with

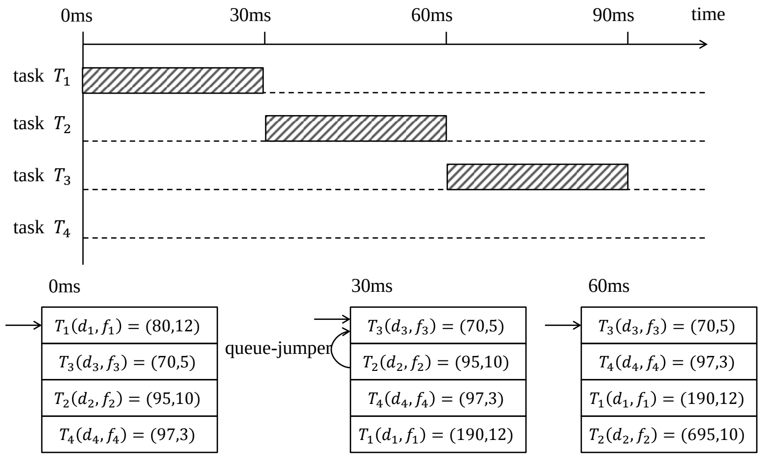


Figure 5. An example of the BFM scheduling strategy

Algorithm 2 BFMS scheduling Algorithm

Require: A set of of VMs, T_i $(1 \le i \le n)$, with their CPU requirements $(p_i,\ s_i,\ x_i)$; the best frequency of T_i, f_i; scheduling slice P_s; frequency scaling slice P_f; current time t_c; T_i's deadline d_i; T_i's location in the queue, L_i;

```
1: for The end of each scheduling slice do
2:     Calculate the deadlines of all tasks, sort the tasks in the ascending order of deadline in queue $Q$
3:     Obtain the VM with the earliest deadline, denoted by $T_e$
4:     for all Tasks in the run queue do
5:         Calculate the frequency gap $g_i$ between the best frequency of task $T_i$ and the current frequency $f$
6:     end for
7:     Sort the tasks' frequency gaps in the ascending order
8:     Get the first frequency gap, $g_k$, in the frequency-gap sorting queue and denote the corresponding task by $T_k$
9:     if $T_k$ is $T_e$, i.e., $g_e$ is the minimal gap
       then
10:        Schedule $T_e$ to run
11:    else
12:        while $g_k$ is no more than $g_e$ do
13:            if Each task $T_i$ before $T_k$ in queue $Q$ satisfies $d_i - [t_c + (L_i + 1) * P_s] \geq 0$ then
14:                Schedule $T_k$ to run next
15:            end if
16:            Get the next frequency gap, $g_k$, in the frequency-gap sorting queue and denote the corresponding task by $T_k$
17:        end while
18:        Schedule $T_e$ to run next
19:    end if
20: end for
```

BFMS, the main additional work of BFMM is to allocate a set of VMs among multiple cores in the processor. This section first presents Theorem 2, which is used as the principle for allocating the VMs, and then presents a actual allocation method. After the set of VMs are allocated, the VMs are scheduled and run using BFMS in each individual core.

Theorem 2
Assume there are n VMs and m cores in the processor (assume n can be divided by m). The following allocation method results in the least performance loss.

The n VMs are sorted in the ascending order of their best frequencies. The sequence of the VMs are denoted by $T_1, T_1, ..., T_i, ..., T_n$ (VM T_i's best frequency is f_i). The sequence of VMs are allocated evenly into m cores. Namely, assuming j denotes the index of core $1 \leq j \leq m$, VMs $T_{(j-1)\frac{n}{m}}$ to $T_{(j)\frac{n}{m}}$ are allocated to core j. In this way, tasks with the nearest best frequency will be allocated on the same core, which will lead to the least performance loss.

Proof
We prove the theorem by proving that exchanging any two VMs between different cores in the allocation method will lead to the increase in performance loss.

Assume we exchange task T_i on core C_I with task T_j on core C_J (assume $I < J$, i.e. the VMs' best frequency on C_I are lower than those on C_J). If the total frequency gap after the exchange is higher than that before the exchange, the performance loss after the exchange must be no less than that before the exchange.

Before the exchange, the total frequency gap E between the relevant VMs is:

$$E = (f_i - f_{i-1} + f_{i+1} - f_i) + (f_j - f_{j-1} + f_{j+1} - f_j) \tag{9}$$

After exchanging, the total frequency gap E' between the involved tasks can be calculated as:

$$E' = f_j - f_{i-1} + f_{j+1} - f_i \tag{10}$$

Note that the gap of $f_j - f_{i+1}$ and $f_{j-1} - f_i$ are not counted in the expression above since it does not cause performance loss (but energy waste) even if Xen cannot adjust the frequency timely from f_j to f_{i+1} and f_{j-1} to f_i. Thus,

$$E' - E = f_j + f_{j-1} - f_i - f_{i+1} \tag{11}$$

Since f_j and f_{j-1} are greater than f_{i+1} and f_i, we get $E' \geq E$.

Note the above expressions for calculating E and E' capture the general cases. There are special cases where f_i and f_j are the highest or lowest frequencies in their cores. These special cases can also be proved in a similar way. □

Theorem 2 essentially states the allocation principle that the VMs with close best frequencies should be allocated to the same core. In SEDF, however, a VM, T_i, has the CPU requirement, specified by the first two parameters of the triple (s_i, p_i, x_i). T_i's CPU requirement can be computed as $\frac{s_i}{p_i}$. When BFMM allocates the VMs, it needs to make sure that all VMs allocated to the same core can meet their CPU requirements. According to the schedulability analysis in the literature [21], if the sum of $\frac{s_i}{p_i}$ for a group of VMs allocated in a core is less than 100%, the CPU requirements of this group of VMs can be met by SEDF. Based on the consideration of the CPU requirement, we adjust the allocation method in Theorem 2 and present the allocation method used in BFMM. The fundamental idea of the adjusted allocation method is as follows.

When allocating a set of VMs to a set of cores, C = $\{C_1,C_2,...,C_m\}$, BFMM first sorts the VMs in the ascending order of their best frequencies. The sorted VM set is T = $\{T_1,T_2,...,T_n\}$ (i.e. T_1 has the lowest best frequency while T_n has the highest best frequency). VM T_i's best frequency and CPU requirement are f_i and $\frac{s_i}{p_i}$. BFMM allocates the VMs from T_1 to T_n one by one to m cores. It tries to allocate the VMs with the closest best frequencies to the same core as long as the CPU requirement of the VMs on the same core can be satisfied, i.e., the sum of $\frac{s_i}{p_i}$ on the core is less than 100%. When BFMM allocates T_i and finds that T_i's CPU requirement cannot be met in the core, it moves to the next VM and check if the next VM can be allocated the core. The process continues until all VMs are examined for the current core. BFMM then moves to the next core and tries to allocate VMs to the core. This process repeats until all VMs are allocated. The allocation method for BFMM is outlined in Algorithm 3.

Algorithm 3 The VM allocation method in BFMM

```
Require: A set of VMs $\{T_i\}(1 \leq i \leq n)$; the frequency of $T_i$, $f_i$; the CPU requirement of $T_i$, $\frac{s_i}{p_i}$); a set of cores $C_j, (1 \leq j \leq m)$;
 1: for all VMs do
 2:   Sort the VMs in the ascending order of frequency and obtain the sorted list of $\{T_1, T_2, ..., T_n\}$, i.e., $f_1 \leq f_2 \leq ... \leq f_n$
 3: end for
 4: The current core $C_c$ is initialized to be $C_1$, i.e., $c = 1$
 5: for $T_1 \rightarrow T_n$ do
 6:   if VM $T_i$ has not been allocated then
 7:     Calculate the total CPU requirement of VMs allocated to $C_c$, denoted by $\sum_c$
 8:     if $\frac{s_i}{p_i} + \sum_c \leq 100\%$
        then
 9:       allocate $T_i$ to core $C_c$
10:     else
11:       for $j = i + 1; j \leq n; j++;$ do
12:         if VM $T_j$ satisfies $\frac{s_j}{p_j} + \sum_c \leq 100\%$ then
13:           Allocate $T_j$ to core $C_c$
14:         end if
15:       end for
16:       $c++$
17:     end if
18:   end if
19: end for
```

Note that when there are no deadlines, the BFM strategy essentially becomes LLS. Another point is that BFM works by re-arranging the execution order of the jobs in the run queue (i.e., allowing queue-jumping), which is designated by SEDF, only when the deadline allows. So in terms of meeting real-time requirements, BFM is as good as SEDF. The only case where BFM is unable to guarantee a task's deadline is when SEDF is unable to meet its deadline. In this case, BFM will simply disallow the queue-jumping and the scheduling behaviour of BFM will be the same as SEDF.

7. EVALUATION

7.1. Experimental Setup

We conducted the experiments on the server with Intel(R) Core(TM) i7-3615QM CPU@2.30GHz processor, 32GB RAM and 122GB hard drives. The processor has 4 physical cores and supports 12 performance states from the minimum frequency of 1200 MHz to the maximum frequency of 2301 MHz. Xen-4.4.1 hypervisor with the kernel version of 3.13.0-32-generic was used to create the virtualized system. SEDF is selected as the vCPU scheduler and the Ondemand governor as the DVFS runtime power management. Each VM in the experiments was run with 2 VCPUs and 256M memory.

The best CPU frequency for running a task in a VM is determined in the following way. We first set the Xen governor to userspace and then set the CPU frequency using the commands: xenpm set-scaling-minfreq and xenpm set-scaling-maxfreq. We run the task with different frequencies and record the execution time and energy consumption. Fig. 6 shows the execution times of different benchmark tasks running on different frequencies.

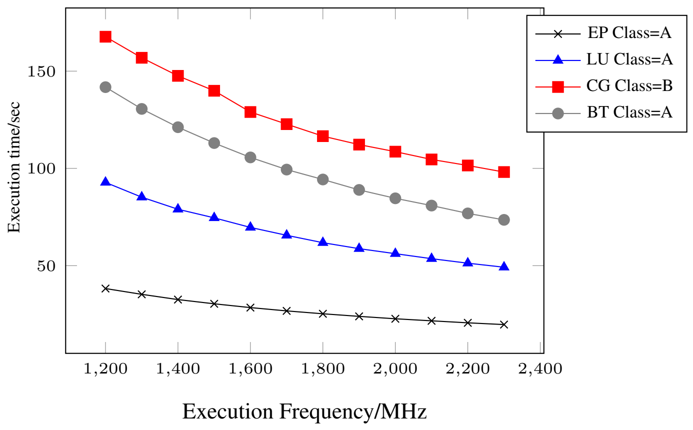


Figure 6. The execution times of different benchmark tasks running with different frequencies

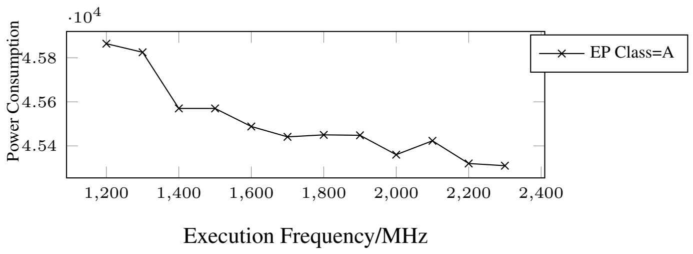
As we can see in Fig. 6, as the execution frequency increases, the decreasing trend of the execution times of all the benchmarks diminishes. The total energy consumption of completing a task can be calculated by the execution frequency times the corresponding execution time (i.e., the value on the x axis times the corresponding value on the y axis). Fig. 7 shows the power consumption of the benchmark tasks running on different frequencies. In this paper, the best CPU frequency of a task is defined as the frequency which leads to the lowest energy consumption of the task. According to Fig. 7a to Fig. 7d, we can know that the best frequencies of the four benchmarks, EP, LU, CG and BT, are 2300 MHz, 1400 MHz, 1200 MHz and 1700 MHz, respectively. A task's execution time when it is run with the best frequency is called the best frequency execution time.

7.2. Experiments on Single-core processors

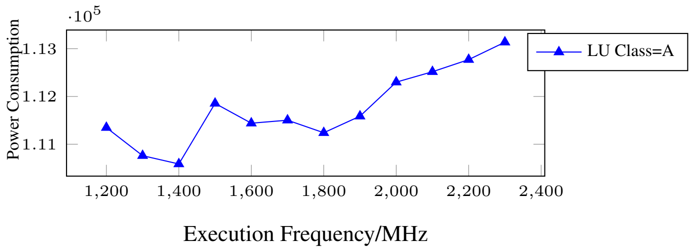
In this section, we compare the BFM scheduler with the SEDF scheduler in Xen in terms of the performance of managing the VMs in a single core. Four benchmark applications, EP, LU, CG and BT, are used, which are denoted T_1, T_2, T_3 and T_4, respectively. The best frequencies of T_1 to T_4 are 2300 MHz, 1400 MHz, 1200 MHz, 1700 MHz. Their best frequency execution times are 9850 ms, 7899 ms, 16768 ms, 9938 ms. We run these tasks, each in a separate VM, on a single core under the SEDF and the BFM schedulers. Fig. 8 compares the execution times of the tasks under these two schedulers. The best frequency execution time is also depicted in the figure for comparison.

Our experimental records for Fig. 8 show that the execution times of $T1$, $T3$ and $T4$ are reduced by 10 ms, 140 ms and 10 ms, respectively, under BFM, compared to SEDF. This can be explained as follows. Under SEDF, the tasks, especially those frequency-sensitive tasks (i.e. the executing frequency has a big influence on its execution time, for instance T_3), may run with the frequency which is lower than its best frequency, and therefore the execution time may decrease. For the tasks with high best frequencies (i.e. T_1 and T_4), the execution times under SEDF increase, comparing to their best frequency execution times. This is because they may be scheduled behind some tasks with low best frequencies and Xen cannot adjust the frequency up timely during the scheduling process. On the countrary, BFM may allow other tasks to jump the queue if they have smaller gap between the best frequency and the current frequency (subject to the deadline requirement) and therefore give Xen more time to build up the frequency to run the high frequency tasks.

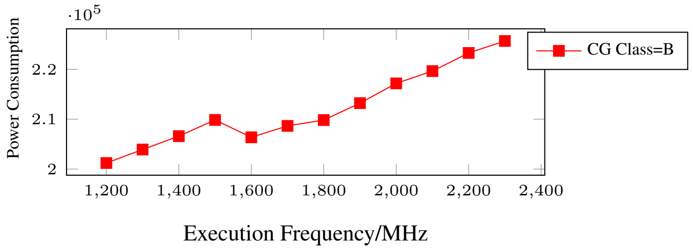
Fig. 9 compares the power consumption of the tasks running under SEDF and BFM. The power consumption of the tasks running with their best frequencies, i.e., the frequencies that result in the minimal power consumption by tasks, are also drawn in the figure for comparison. It can be seen that the power consumption under BFM is much less than that under SEDF for the tasks T_2 and T_3. This is because under SEDF, the tasks, especially those with low best frequencies, may run with the frequency higher than what they need (the best frequency), which causes energy waste. High best frequency tasks, for example, T_1 (EP), consume 2.2655×10^7, 2.2728×10^7, 2.2720×10^7,



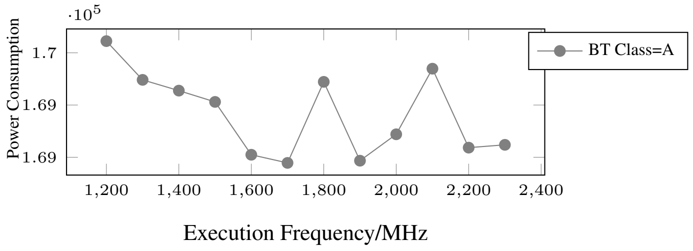
(a) The power consumption of EP running with different frequencies



(b) The power consumption of LU running with different frequencies



(c) The power consumption of CG running with different frequencies



(d) The power consumption of BT running with different frequencies

Figure 7. The power consumption of benchmarks running with different frequencies

respectively, with the best frequency, under SEDF and under BFM. SEDF scheduler leads to a power waste of 73000 (i.e. 2.2728×10^7 - 2.2655×10^7) while BFM leads to a power waste of only 65000. These results suggest that BFM can reduce energy consumption while improving performance by allowing the suitable VMs to jump the queue to fill in the gap between the current frequency and the best frequency of the VM at the head of the queue under SEDF.

7.3. Experiments on multi-core processors

We use SEDF and BFMM to schedule and run 20 tasks on a DVFS-enabled Quad-Core processor. Each task T_i has the best execution frequency and the CPU requirement, represented by a tuple $(f_i, \frac{s_i}{p_i})$. We set the tasks' execution times so that every task's execution time is 20000ms when they are run with their best frequencies.

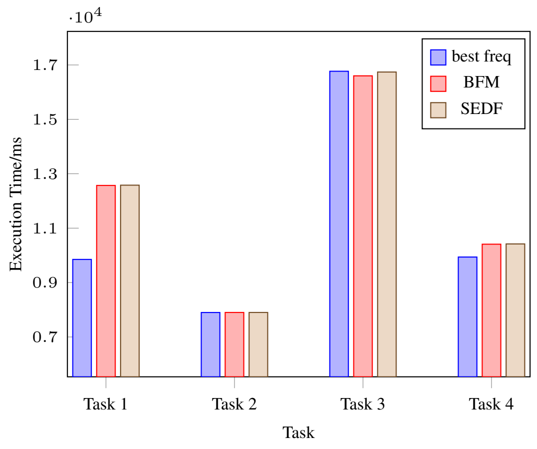


Figure 8. Execution times of tasks on a single core under SEDF and BFM

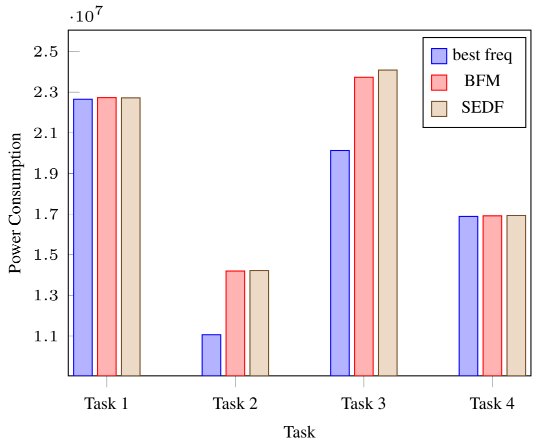


Figure 9. Power consumption of tasks on a single core under SEDF and BFM

Fig. 10 and Fig. 11 show the allocation of 20 tasks on the quad-core processor under SEDF and BFMM. The results show that BFMM allocates the VMs with closer frequencies to the same core, compared with SEDF. Fig. 12 and Fig. 13 show the performance and power consumption of these 20 tasks, respectively. As can be seen from Fig. 12, the VMs with high best frequencies (e.g., T_{13} and T_{14}) have much longer execution times under SEDF than under BFMM. The reason is similar as the reason for the performance gap shown in Fig. 8. Namely, under SEDF these tasks with high best frequencies may be scheduled to run behind the tasks with low best frequencies and therefore Xen cannot adjust the frequency up timely. AS we can see from Fig. 13, BFMM reduces power consumption of all tasks. The reason for this is also similar as the reason for the difference of the power consumption in Fig. 9.

8. CONCLUSION

This work reveals that the traditional scheduling strategies in virtualized systems managed by Xen may lead to performance loss and energy waste, due to the limitation of Xen in adjusting CPU frequency, i.e., Xen can only check and change the CPU frequency at most once every 10ms. This paper presents four scheduling strategies to remedy this situation, which are the Least performance Loss Scheduling (LLS) strategy, the No performance Loss Scheduling (NLS) strategy, the Best Frequency Match strategy for a single core (BFMS) and the Best Frequency Match

Core 1	*Core 2*	*Core 3*	*Core 4*
T_1(1200,20%)	T_2(1700,15%)	T_3(1400,5%)	T_4(2000,15%)
T_5(2000,10%)	T_6(1300,15%)	T_7(2200,35%)	T_8(2100,20%)
T_9(1300,20%)	T_{10}(1600,20%)	T_{11}(1800,20%)	T_{12}(1900,25%)
T_{13}(2100,20%)	T_{14}(2200,25%)	T_{15}(2300,20%)	T_{16}(1300,5%)
T_{17}(1200,25%)	T_{18}(1400,15%)	T_{19}(1900,15%)	T_{20}(2300,25%)

Figure 10. Task consolidation on Quad-Core using SEDF scheduler

Core1	*Core2*	*Core3*	*Core4*
T_1(1200,20%)	T_3(1400,5%)	T_4(2000,15%)	T_{14}(2200,25%)
T_{17}(1200,25%)	T_{10}(1600,20%)	T_5(2000,10%)	T_{20}(2300,25%)
T_9(1300,20%)	T_2(1700,15%)	T_8(2100,20%)	T_{15}(2300,20%)
T_6(1300,15%)	T_{11}(1800,20%)	T_{13}(2100,20%)	
T_{16}(1300,5%)	T_{12}(1900,25%)	T_7(2200,35%)	
T_{18}(1400,15%)	T_{19}(1900,15%)		

Figure 11. Task consolidation on Quad-Core using BFM scheduler

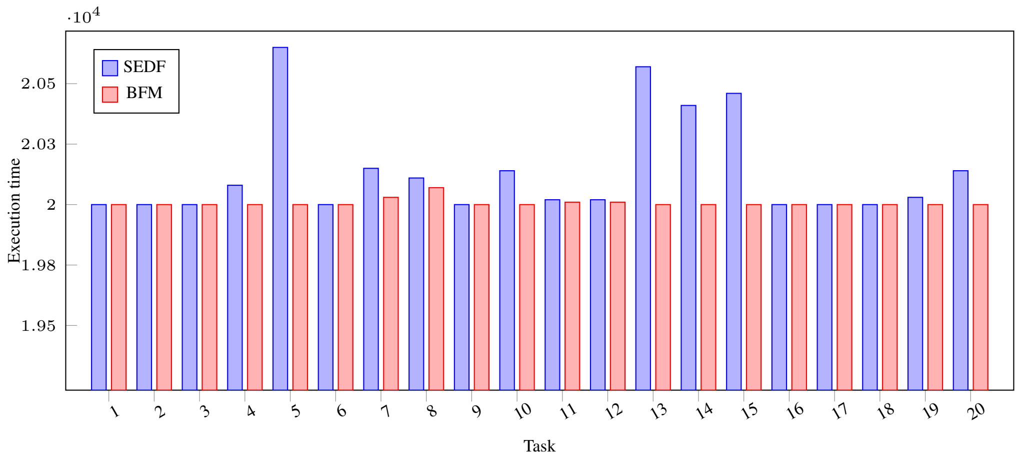


Figure 12. Execution times of tasks on a multi-core processor under SEDF and BFMM

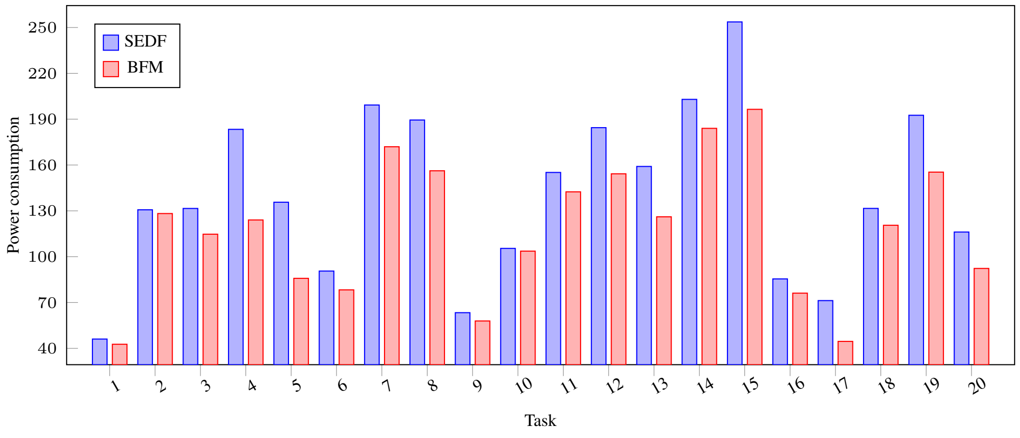


Figure 13. Power consumption of tasks on a multi-core processor under SEDF and BFMM

strategy for multiple cores (BFMM). These strategies make use of the scheduling behaviour in the Xen hypervisor and aim to reduce energy consumption while mitigating performance loss. the effectiveness of these strategies is theoretically proved and also evaluated by the experiments. The philosophy used in BFM to reduce performance loss and energy consumption may also be applied to other schedulers in Xen, such as the Credit scheduler. In the future, we plan to adapt the Credit scheduler to mitigate the performance loss and energy waste.

9. ACKNOWLEDGEMENT

This work is partially supported by the EU Horizon 2020Marie Sklodowska-Curie Actions through the project entitled Computer Vision Enabled Multimedia Forensics and People Identification (Project No. 690907, Acronym: IDENTITY).

REFERENCES

1. Jeongseob Ahn, Chang Hyun Park, and Jaehyuk Huh. Micro-sliced virtual processors to hide the effect of discontinuous cpu availability for consolidated systems. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pages 394–405. IEEE, 2014.
2. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.
3. Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755 – 768, 2012. Special Section: Energy efficiency in large-scale distributed systems.
4. Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing server energy and operational costs in hosting centers. *SIGMETRICS Perform. Eval. Rev.*, 33(1):303–314, June 2005.
5. Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Performance Evaluation Review*, 35(2):42–51, 2007.
6. Ligang He, Deqing Zou, Zhang Zhang, Chao Chen, Hai Jin, and Stephen A Jarvis. Developing resource consolidation frameworks for moldable virtual machines in clouds. *Future Generation Computer Systems*, 32:69–81, 2014.
7. Nakku Kim, Jungwook Cho, and Euiseong Seo. Energy-credit scheduler: An energy-aware virtual machine scheduler for cloud systems. *Future Generation Computer Systems*, 32(0):128 – 137, 2014. Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures.
8. Dawei Li and Jie Wu. Energy-aware scheduling for aperiodic tasks on multi-core processors. In *Parallel Processing (ICPP), 2014 43rd International Conference on*, pages 361–370. IEEE, 2014.
9. Kenli Li, Xiaoyong Tang, and Keqin Li. Energy-efficient stochastic task scheduling on heterogeneous computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 25(11):2867–2876, Nov 2014.
10. Ching-Chi Lin, Chao-Jui Chang, You-Cheng Syu, Jan-Jan Wu, Pangfeng Liu, Po-Wen Cheng, and Wei-Te Hsu. An energy-efficient task scheduler for multi-core platforms with per-core dvfs based on task characteristics. In *Parallel Processing (ICPP), 2014 43rd International Conference on*, pages 381–390. IEEE, 2014.
11. Ming Liu, Chao Li, and Tao Li. Understanding the impact of vcpu scheduling on dvfs-based power management in virtualized cloud environment. In *Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, pages 295–304. IEEE, 2014.
12. T. Mudge. Power: a first-class architectural design constraint. *Computer*, 34(4):52–58, Apr 2001.
13. Jean-Marc Pierson and Henri Casanova. On the utility of dvfs for power-aware job placement in clusters. In *Euro-Par 2011 Parallel Processing*, pages 255–266. Springer, 2011.
14. Andrei Sfrent and Florin Pop. Asymptotic scheduling for many task computing in big data platforms. *Information Sciences*, 319:71 – 91, 2015. Energy Efficient Data, Services and Memory Management in Big Data Information Systems.
15. Mihaela-Andreea Vasile, Florin Pop, Radu-Ioan Tutueanu, Valentin Cristea, and Joanna Kołodziej. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems*, 51:61 – 71, 2015. Special Section: A Note on New Trends in Data-Aware Scheduling and Resource Provisioning in Modern {HPC} Systems.
16. Gregor Von Laszewski, Lizhe Wang, Andrew J Younge, and Xi He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–10. IEEE, 2009.
17. Leping Wang and Y. Lu. Efficient power management of heterogeneous soft real-time clusters. In *Real-Time Systems Symposium, 2008*, pages 323–332, Nov 2008.
18. Sisu Xi, J. Wilson, Chenyang Lu, and C. Gill. Rt-xen: Towards real-time hypervisor scheduling in xen. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pages 39–48, Oct 2011.

19. Cong Xu, Sahan Gamage, Pawan N Rao, Ardalan Kangarlou, Ramana Rao Kompella, and Dongyan Xu. vslicer: latency-aware virtual machine scheduling via differentiated-frequency cpu slicing. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pages 3–14. ACM, 2012.
20. Chao Yu, Leihua Qin, and Jingli Zhou. A multicore periodical preemption virtual machine scheduling scheme to improve the performance of computational tasks. *The Journal of Supercomputing*, 67(1):254–276, 2014.
21. Fengxiang Zhang and Alan Burns. Schedulability analysis for real-time systems with edf scheduling. *Computers, IEEE Transactions on*, 58(9):1250–1258, 2009.
22. Huanzhou Zhu, Ligang He, Stephen Jarvis, et al. Optimizing job scheduling on multicore computers. In *Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, pages 61–70. IEEE, 2014.