ITC 4/49
Information Technology and Control
Vol. 49 / No. 4 / 2020
pp. 608-621
DOI 10.5755/j01.itc.49.4.25029

A Two-stage Strategy to Optimize Energy Consumption for Latency-critical Workload Under QoS Constraint

Received 2020/01/07 | Accepted after revision 2020/10/30

http://dx.doi.org/10.5755/j01.itc.49.4.25029

HOW TO CITE: Li, J., Teng, D., Lin, J. (2020). A Two-stage Strategy to Optimize Energy Consumption for Latency-critical Workload Under QoS Constraint. *Information Technology and Control*, 49(4), 608-621. https://doi.org/10.5755/j01.itc.49.4.25029

A Two-stage Strategy to Optimize Energy Consumption for Latency-critical Workload Under QoS Constraint

Jingwei Li, Duanyu Teng, Jinwei Lin
Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China;
e-mails: lijw66@163.com, dyteng.xjtu@gmail.com, ljw7325@stu.xjtu.edu.cn

Corresponding author: lijw66@163.com

Reducing energy consumption can bring various benefits, such as saving power, reducing operating costs and improving system reliability. Data centers afford huge energy costs. Reducing energy consumption and providing efficient quality of service (QoS) are the goals pursued by data centers. This paper aims to develop the scheduling strategy to reduce the energy consumption for latency-critical workload with the Dynamic Voltage Frequency Scaling technique. In this paper, we propose a two-stage strategy that dynamically schedules the CPU to run at the optimal frequency level and provides satisfactory QoS during the latency-critical workload execution. The two-stage strategy includes a static stage and a dynamic stage. In the static stage, a heuristic algorithm is developed to determine an optimal frequency level for different loads, at which the CPU consumes lower energy. However, due to the dynamic characteristics of the load, the determined frequency in the static stage may not guarantee QoS. Therefore, in the dynamic stage, a threshold method is proposed to fine-tune the frequency to ensure the QoS. The two stages are worked together to reduce energy consumption for latency-critical workload which has QoS constraint. A case study on the Web search application shows that our scheduling strategy is effective for energy-saving, with a reduction of more than 13% compared with the baseline strategy.

KEYWORDS: High energy cost, latency-critical workload, two-stage strategy, energy-saving, QoS constraint.

1. Introduction

Cloud computing technology and big data technology have promoted the vigorous development of the Internet. These technologies have been applied to all aspects of our modern life, such as social networking [6], media streaming [21], e-commerce [1], mobile Internet [14] and so on. With the popularity of these technologies, data centers need to process a large amount of data every day, which makes enterprises and governments have to invest in building more data centers. However, building a data center costs millions of dollars, of which the energy cost accounting for a significant proportion of total investment [9], [22]. According to 2017 statistics, about 8 million data centers around the world are processing Internet data. The power consumption of these data centers is 416.2 terawatt hours, accounting for 2% of the global total power consumption [30]. In addition, these data centers produce more than 43 million carbon dioxide annually [2]. Huge power consumption not only brings high energy costs but also damages the environment. Therefore, reduction of power consumption is significant and urgent.

Power saving is a challenging issue in data centers because of the contradiction between low power consumption and high performance. Latency-critical applications such as Web search, media streaming, etc. must meet their service-level agreement (SLA) for performance [15], [32]. To guarantee the quality of service (QoS) of these applications, data centers have to force the CPU to run at a higher frequency, which results in high power consumption. The trade-off between power consumption and performance is a hot issue of common concern in industry and academia [13][19].

The CPU is the most energy consuming component in the server, so studying the CPU power efficiency is a significant direction for energy-saving [3][27]. Dynamic Voltage and Frequency Scaling (DVFS) is a well-known technique for reducing processor power and energy. Many researchers have proposed their strategies or techniques based on DVFS for energy-saving from different perspectives. Ibrahim et al. [12] investigated the DVFS technology in the Hadoop cluster. Their work provides a useful insight into designing power-aware techniques for Hadoop systems. The work in [23] uses the DVFS technique to minimize energy consumption in parallel applications. Choi et al. [7] proposed an intra-process DVFS technique to save energy for the non-real-time applications running on an embedded system. Wang et al. [26] focus on power reduction in heterogeneous multi-tier clusters and apply Generalized Benders Decomposition (GBD) to solve the optimization problem.

Ondemand [4] is a typical DVFS based CPU frequency governor in the Linux system. It is a default strategy in Linux system and is often used as a baseline for comparison with other DVFS based strategies. Ondemand strategy scheduling the CPU frequency based on the workload running on the server. It uses the CPU utilization as an indicator of frequency scheduling. When it detects that the CPU utilization exceeds a specific threshold, it quickly schedules the processor to run at the highest frequency to ensure the performance of the load. Ondemand strategy can automatically scale the frequency according to the load while incurring almost negligible overhead. However, for latency-critical workloads, Ondemand strategy exposes two limitations. First, Ondemand aggressively schedules the frequency between almost the highest and lowest frequencies, so it cannot make full use of other intermediate frequencies to save power. Second, Ondemand only uses CPU utilization as the scheduling indicator. Sometimes a single CPU utilization does not accurately reflect the real work of the processor.

To address these limitations, we propose a two-stage strategy to dynamically schedule the CPU frequency during latency-critical workload execution to minimize energy consumption and meet the QoS constraint. The two-stage strategy includes a static frequency determination stage and a dynamic frequency adjustment stage. The static stage uses a heuristic algorithm to model the frequency-load relationship and generate a quantitative relation table. This table determines the optimal CPU frequency for a running workload, at that frequency the energy consumption is minimum when performance constraint is met. To avoid transient load surge that may damage QoS, the dynamic stage uses a threshold method to fine-grained adjust the CPU frequency. The threshold method adjusts the frequency based on the calculated Instruction per cycle (IPC) of the processor. Once the

IPC value exceeds the threshold, the CPU frequency is quickly adjusted to the highest frequency. Or else, if the IPC value does not exceed the threshold, the optimal frequency of the static stage is maintained. During the workload operation, the static stage detects the load every 1 second (abbreviated as s and later s is used) and determines the optimal frequency for it, while the dynamic stage calculates the IPC value every 1 millisecond (abbreviated as ms and later ms is used) and determines whether to adjust the frequency. The two stages worked together to minimize the energy cost and ensuring the QoS.

In summary, our paper makes the following contributions:

- A two-stage scheduling strategy is proposed to reduce the energy consumption and provide satisfactory QoS for the latency-critical workload.
- A heuristic algorithm is developed to determine an optimal CPU frequency for a running workload which is under the QoS constraint. At the optimal frequency, the CPU consumes lower energy.
- A comprehensive evaluation is conducted to demonstrate the effectiveness of the two-stage strategy for energy-saving and QoS-protection.

Paper organization: In Section 2, we briefly describe the background and clarify the motivation of this work. Section 3 elaborates on the two-stage strategy, including the heuristic algorithm and the threshold method. Section 4 describes the implementation of the two-stage strategy and Section 5 presents the case study. Section 6 reviews the related work, and Section 7 concludes the whole work.

2. Background and Motivation

In this section, we briefly introduce the frequency scheduling strategy in the Linux system and discuss its limitations through an experiment. Then we give a motivation example to clarify the insight of this work.

2.1. Ondemand Strategy and Its Limitation

Cpufreq is a kernel subsystem in the Linux power management system, and the governor in cpufreq controls the frequency scaling strategy of the processor. Cpufreq provides several frequency scaling strategies such as, performance, powersave, ondemand, and userspace etc. The governor schedules the strategy according to the user's demand. For instance, performance strategy highlights performance and keeps the CPU running at the highest frequency, while powersave emphasizes energy-saving and fix the CPU running at the lowest frequency. Ondemand strategy is the default one in the Linux system. It schedules the CPU frequency between the highest and lowest according to the CPU utilization. Once the detected CPU utilization exceeds a certain threshold, Ondemand schedules the CPU to run at the highest frequency, else it keeps the CPU to run at the lowest frequency. Usersapce provides a program interface that allows users to customize their frequency scaling strategy. Ondemand automatically scheduling the CPU frequency according to the system load. However, it is not flexible because it does not use other frequency levels and therefore limits the more space to save energy.

We evaluate the power saving effect of the Ondemand strategy on the Web search application. Web search is a typical latency-critical workload in data centers, which has a SLA requirement. To let users feel a good search experience, it is generally believed that the response time of Web search should be less than 300ms, otherwise the users rather choose to drop or leave [20][25]. Therefore, for latency-critical workloads, the scheduling strategy should reduce power consumption as well as meet their performance requirements. This paper uses the Web search application as an example of latency-critical workload to carry out the research. Our strategy is also applicable to other latency-critical workloads. We deploy the Web search benchmark on two servers, of which one works as server end, and the other works as client end. In the experiment, we set the server end works in Ondemand strategy and set the client to send different numbers of requests to the server end continuously within a period of 696s. During the workload operation, we measure the response time of the Web search and the energy consumption of the CPU of the server end. The details of how to measure the response time and CPU energy are presented in Section 4.1. In this experiment, the measured average response time is 108ms, and the total energy consumption in the whole process is 14549J. We can see that Ondemand strategy can better meet the QoS requirement (less than 300ms) but consume a lot of energy. To improve energy-saving, this paper proposes a two-stage strategy to tradeoff energy consumption and performance.

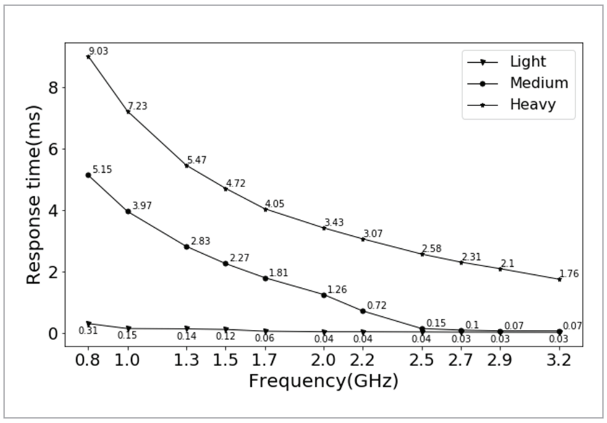
2.2. Motivation

According to the characteristic of CMOS circuit, the dynamic power of the processor is proportional to the square of the processor frequency and voltage [28] [31]. Therefore, the higher the frequency, the more of the power consumption. DVFS is a dynamic CPU voltage and frequency scaling technology enables the operating system to scale the CPU frequency up or down according to the system load. At present, most processor microarchitectures equipped with DVFS technique and it has been widely used for CPU power optimization in industry and academia. We propose a two-stage strategy based on DVFS technique, which aims to reduce energy consumption and guarantee performance for the latency-critical workload. The objective of the two-stage strategy is to determine the optimal frequency level for the given workload, at which the CPU consumes lower energy. We analyze the relationship between frequency, performance, and power through a case study on Web search. In the case study, the server is equipped with the Intel Haswell architecture processor. Table 1 lists the detail parameters of the processor and Table 2 lists the frequency levels that the processor supported.

We configure the client generates three different sizes of load (light, medium, and heavy) and set the processor to run on each of the 16 frequency levels. Each load runs for 300s on the 16 frequencies of the processor, and the response time and power consumption are measured during the load operation. It is worth noting that the response time at each frequency is the average response time of the load running for 300s, and the energy at each frequency is the total energy of 300s. Figure 1 illustrates the performance variations of the three loads at different CPU frequencies. Figure 2 shows the power consumption variations of the three loads at different CPU frequencies. It can be seen from the two figures that, the higher the frequency, the smaller the response time and the greater the energy consumption. However, the change rates are different. It can be seen from Figure 1 and Figure 2 that for the medium load, the performance improvement is very little after the frequency above 2.5, but the energy consumption is still increasing greatly. This observation hints that for different loads, choosing the appropriate frequency can both reduce the power and provide satisfactory performance.

Figure 1

Performance variations of the three loads at different CPU frequencies. The horizontal axis shows different CPU frequencies, and the vertical axis represents the performance of the load at that frequency. Light Medium and Heavy refers to three different size of loads



Therefore, our work needs to model the relationship between load and frequency in order to determine the optimal frequency level for the load which is under the QoS constraint. As can be seen from Figure 2, the relationship between power consumption and frequency approximately presents a linear trend. At a specific frequency, energy consumption is related to the load

Table 1

Information of the processor

Item	Parameter
Processor	Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
Microarchitecture	Haswell
CPU cores	4
Operating system	Ubuntu 14.04.5 LTS
Kernel version	Linux version 3.16.0-77-generic

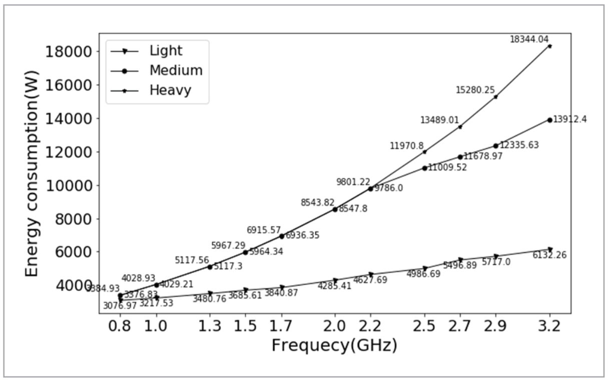
Table 2

Frequency levels of the processor supported

Frequency (GHz)	0.8	1.0	1.1	1.3	1.5	1.7	1.8	2.0
	2.2	2.3	2.5	2.7	2.9	3.0	3.2	3.201

Figure 2

Power consumption variations of the three loads at different CPU frequencies. The horizontal axis shows different CPU frequencies, and the vertical axis represents the power consumption of the CPU at that frequency. Light Medium and Heavy refers to three different size of loads



being processed. Thus, we empirically assume that there is also an approximately linear relation between load and frequency. The two-stage strategy is developed under this assumption. The evaluation results show that the proposed strategy has achieved good effect for energy-saving and QoS-protection, which indirectly illustrates the rationality of the linear assumption.

3. The Two-stage Strategy

This section elaborates on the two-stage strategy. The two stages are static stage and dynamic stage. The static stage generates a frequency-load relation table through a well-designed heuristic algorithm. To avoid transient load surge to damage the QoS, the dynamic stage adjusts the previously planned frequency by using the threshold method.

3.1. Static Stage

In Section 2.2, we assume that load and frequency are approximately linear relationship. Based on this assumption, a heuristic algorithm is designed to determine a corresponding frequency for a load, and finally generates a $[frequency, load]$ matching pairs. The response time of Web search is required less than 300ms. Thus, the Qos constraint is 300ms. The heuristic algorithm calculates the optimal frequency for the load that is under this QoS constraint. The following part will describe the heuristic algorithm and its pseudo-code.

Description of the heuristic algorithm:

Step 1: Determine the frequency of the lower limit load under QoS constraint. Specifically, for the lower limit load $load_{low}$, measure its actual response time res at the lower limit frequency $freq_{low}$. If the actual response time is less than the QoS constraint con, i.e., $res < con$, then the matching pair $[frequency, load]$ is determined, else the frequency level is increased step by step until meeting the QoS constraint, i.e., meet $res < con$, this frequency level is determined as a match to $load_{low}$.

Step 2: Determine the frequency of the upper limit load under QoS constraint. Similar to step 1, for the upper limit load $load_{upp}$, measure its actual response time res at the upper limit frequency $freq_{upp}$. If the actual response time is greater than the QoS constraint, i.e., $res > con$, then the matching pair $[frequency, load]$ is determined, else the frequency level is decreased by one level and measure the response time at this level. If the actual response time is less than the QoS constraint, i.e., $res < con$, this frequency level is determined as a match to $load_{upp}$.

Algorithm 1: The heuristic algorithm

```
1:  Input: $con, freq[N], low, upper, load_{low}, load_{upp}$
2:  Output: $[frequency, load]$
3:  $res \leftarrow measure\_response(load_{low}, freq[low])$
4:  If $res > con$ Then
5:    For $res > con$ Do
6:      $low \leftarrow low + 1$
7:      $res \leftarrow measure\_response(load_{low}, freq[low])$
8:    End For
9:  End If
10: $[frequency, load[low]] \leftarrow load_{low}$
11: $res \leftarrow measure\_response(load_{upp}, freq[upp])$
12: If $res > con$ Then
13:   $[frequency, load[upp]] \leftarrow load_{upp}$
14: Else
15:   $high \leftarrow high - 1$
16:   $res \leftarrow measure\_response(load_{upp}, freq[upp])$
17:     If $res < con$ Then
18:       $[frequency, load[upp]] \leftarrow load_{upp}$
19:     End If
20: End If
21: $generate\_med(con, freq[N], low, med, \ldots$
    $load_{low}, load_{med}, [frequency, load])$
22: $generate\_med(con, freq[N], med, upp, \ldots$
    $load_{med}, load_{upp}, [frequency, load])$
23: return $[frequency, load]$
```

Step 3: Determine the frequency of the median load under QoS constraint. Since the linear relationship between load and frequency, when the load takes the median value of the upper limit load and lower limit load, the corresponding frequency value should also be near the median of the upper limit frequency and lower limit frequency. Specifically, calculate the median load $load_{med}$ and median frequency $freq_{med}$. Measure the actual response time res of $load_{med}$ at $freq_{med}$. If the actual response time is less than the constraint, i.e., $res < con$ then the matching pair $[frequency, load]$ is determined, else the frequency level is increased step by step from median frequency $freq_{med}$ until meeting the QoS constraint, i.e., $res < con$, this frequency level is determined as a match to $load_{med}$.

Algorithm 2: The function of generate_med

```
1:  Input: $con, freq[N], low, upp, load_{low}, load_{upp}, \ldots$
           $[frequency, load]$
2:  Output: $[frequency, load]$
3:  $load_{med} \leftarrow \frac{load_{low} + load_{upp}}{2}, med \leftarrow \frac{low + upp}{2}, \ldots$
    $res \leftarrow measure\_response(load_{med}, freq[med])$
4:  If $res > con$ Then
5:    For $res > con$ Do
6:      $med \leftarrow med + 1$
7:  $res \leftarrow measure\_response(load_{med}, freq[med])$
8:    End For
9:  End If
10: $[frequency, load[med]] \leftarrow load_{med}$
11: $generate\_med(con, freq[N], low, med, \ldots$
    $load_{low}, load_{med}, [frequency, load])$
12: $generate\_med(con, freq[N], med, upp, \ldots$
    $load_{med}, load_{upp}, [frequency, load])$
```

Step 4: Iteratively determine the new median load and the new median frequency. Repeat step 3 by dichotomy method until no median frequency need be calculated.

Algorithm 1 and Algorithm 2 are pseudo-codes of the heuristic algorithm. Table 3 describes the variables in the pseudo-codes.

We apply the heuristic algorithm to the Web search application and the Haswell architecture processor (described in Table 1) to calculate the [frequency,-load] matches. We set the lower limit load is 10, the upper limit load is 440, the lower limit index is 1, the upper limit index is 16 and the QoS constraint to 300ms. The heuristic algorithm calculates the optimal frequency levels for loads that ranging from 10 to 440. Table 4 lists the relation table, where the first column is the range of the load, and the second column is the corresponding frequency level. During the workload operation, the relation table is used as the frequency scheduling rule to set the CPU run at a matching frequency. For instance, when the detection load is 80, the processor will be set to run at 1.8GHz.

Table 3
Descriptions of the variables in the pseudo-codes

Variable	Meaning
$freq[N]$	The Array of frequency levels that the processor supported
N	Number of frequency levels
con	QoS constraint
res	Actual response time
low	Lower limit index
$upper$	Upper limit index
$load_{low}$	Lower limit load
$load_{upp}$	Upper limit load
$load_{med}$	Median load

To illustrate the stability and applicability of the algorithm, we discuss the effect of variation of parameters on the system behavior. As shown in Table 3, there are nine parameters in the algorithm, of which *freq*[*N*] and *N* are parameters related to processor, and the others related to the running workload.

1 For a certain processor, if the processor supports more frequency levels (*N* becomes larger), the algorithm needs calculate more median frequencies. However, the output will be not affected by this parameter. Since the dichotomy method is applied to calculate the median frequency, the computational complexity is O (1).

2 For a workload, if the QoS constraint (*con*) changes, the output of the frequency may change. For example, the looser the constraint, the lower the matched

Table 4

Application of the heuristic algorithm

Range of load	Frequency
0-5	0.8GHz
5-32	1.0GHz
32-45	1.1GHz
45-59	1.3GHz
59-65	1.5GHz
65-72	1.7GHz
72-86	1.8GHz
86-92	2.0GHz
92-99	2.2GHz
99-112	2.3GHz
112-119	2.5GHz
119-139	2.7GHz
139-153	2.9GHz
153-159	3.0GHz
159-166	3.2GHz
166-	3.201GHz

frequency, because lower frequency can meet the QoS constraint; If the size of the load changes, the output of the frequency may also change. For example, the larger size of the load, the higher the matched frequency is. Since the larger load needs a more powerful CPU frequency.

3.2. Dynamic Stage

The static stage determines the frequency of the running load every 1s, but the sudden load surges and processor delays may damage the QoS. The dynamic stage uses a threshold method to fine-grained adjust the CPU frequency to protect QoS.

In the dynamic stage, we monitor the Instructions per Cycle (IPC) value every 1ms and scheduling the CPU frequency according to this value. Unlike Ondemand strategy, we use IPC value as the indicator rather than CPU utilization. CPU utilization sometimes does not accurately reflect the actual work of the processor. For instance, 90% of the CPU utilization is not necessarily 90% of the time is busy, it may be 20% of the time is busy and 70% of the time is stalled [10]. Stalling means the processor was not making forward progress with instructions, and usually happens because it is waiting on memory I/O. IPC shows on average how many instructions were completed for each CPU clock cycle, which is more accurately reflects the effective work of the CPU. In this paper, IPC is more suitable as an indicator for frequency scheduling than the CPU utilization. The dynamic stage determines the frequency of the running load every 1ms. This decision is based on a threshold method. The threshold is the IPC value of the load at full load operation. If the monitored IPC value is greater than the threshold, the CPU frequency is quickly scheduled to the highest one, otherwise, keep the CPU frequency unchanged.

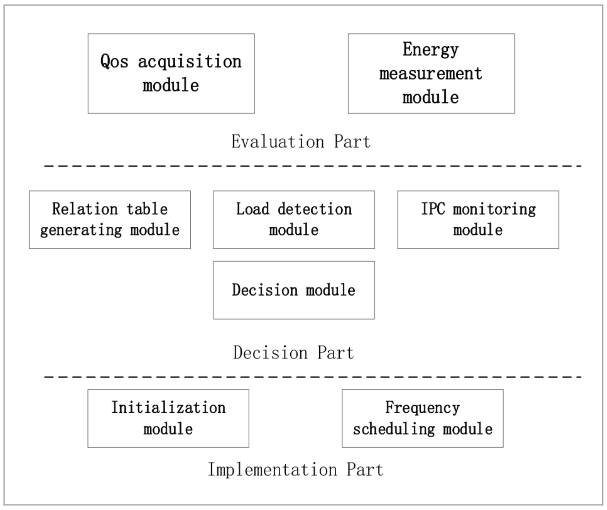
4. Frequency Scheduling with the Two-stage Strategy

Based upon the two-stage strategy, we implement a prototype for dynamic frequency scheduling. This section describes the implementation of the prototype.

4.1. Implementation of the Prototype

Figure 3 shows the architecture of the prototype that contains three parts: the evaluation part, the decision part and the implementation part. In evaluation part, QoS acquisition module responsible for evaluating the QoS of the running workload, and the power measurement module evaluates the power consumption of the CPU. The decision part and the implementation part complete the frequency scheduling through two interactions. The relation table generating module generates [*frequency, load*] matches according to the heuristic algorithm described in Section 3.1. This work is done offline. The load detection module detects the load of the running system every 1s, and the decision module queries the relation table to determine the matching frequency for the detected load. Then the [*frequency, load*] information is transmitted to the frequency scheduling module to implement the first-stage frequency scheduling. In the implementation part, the initialization module sets the governor mode, driver, and counter in the cpufreq system to prepare for frequency scheduling. The frequency scheduling module receives the information from the decision module and sets the CPU runs at the corresponding frequency. Meanwhile, IPC monitoring module continuously monitors the IPC of the proces-

Figure 3
Architecture of the prototype



sor every 1ms and judges whether IPC greater than the preset threshold, if so, the decision module determines the highest frequency for the load and transmits the information to frequency scheduling module to implement the second-stage frequency scheduling. Otherwise, the decision module determines to keep the first-stage frequency scheduling decision. In the following, we describe the implementation details of each module in Figure 3.

***Evaluation part*:**

QoS acquisition module: We obtain the QoS metric through parsing the Solr log file. Specifically, we use curl_get_req interface to send a request to the Solr server every 1s and parse out the 'QTime' metric from the log file. This metric is the response time of the server to the client requests. We take the average response time of all requests processed by the server per second as the QoS value and recorded it in the QoS log file for subsequent evaluation.

Power measurement module: We leverage the RAPL tool [8] and a power meter (Watts Up Pro Power Meter) [29] to measure the power consumption every 1s during the workload operation and recorded it in the power log file for subsequent evaluation.

***Decision part*:**

Relation table generating module: The relation table is generated in offline mode by using the historical data. We use the heuristic algorithm to generate the static frequency-load relation table, like the form described in Table 3. Details of the heuristic algorithm are described in Section 3.1.

Load detection module: For Web search, the load we get is the number of requests sent by the client. We use the interface provided by libcurl to query the Solr log file to obtain the number of requests.

IPC monitoring module: We leverage Intel PCM (Performance Counter Monitor) tool [19] to monitor the IPC parameter every 1ms.

Decision module: This module determines optimal frequencies for the running workload in two stages. In the first stage: query the relation table to determine the matching frequency for the load and transmit the information to the frequency scheduling module. In the second stage: query IPC monitoring module and check whether the IPC value is greater than the threshold (set to 3 in this paper), if so, determines the highest frequency for the load, else, keep the first-stage decision. The first decision is performed every 1s, and the second decision is performed every 1ms.

***Implementation part*:**

Initialization module: This module initializes the cpufreq kernel to support the user-defined frequency scaling. The initialization includes set the driver to apci-driver and set the governor to userspace.

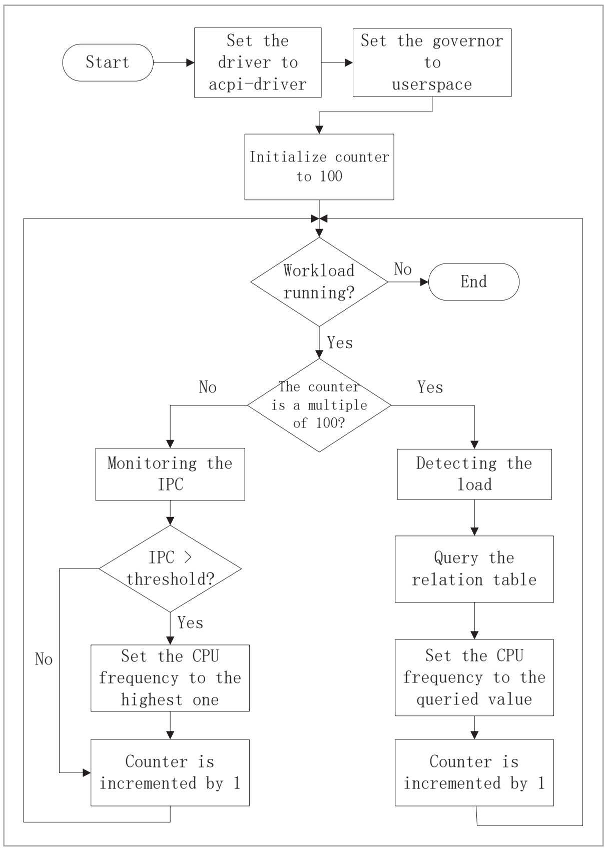
Frequency scheduling module: This module receives the decision information from the decision module and sets the frequency to the corresponding value through the interface provided by Sys.

4.2. Flow Chart of the Two-stage Strategy

We present a flowchart to illustrate how to use the two-stage strategy to accomplish optimal frequency scheduling. Figure 4 shows the flowchart. First, the initialization module sets the driver to acpi-driver and set cpufreq governor to userspace mode. Then, if the load detection module detects that there is workload running, it will continue to carry out subsequent operations, otherwise, it will end. The initialization module set a counter to determine whether the frequency scheduling is based on load or IPC. Since the load is detected every 1s and the IPC is detected every 1ms, thus if the value of the counter is a multiple of 100, the frequency scheduling strategy is determined according to the load, otherwise, it is according to IPC. The counter is initialized to 100 and increments in steps of 1 after

Figure 4

Flowchart of how to use the two-stage strategy to accomplish optimal frequency scaling


Start
Set the driver to acpi-driver
Set the governor to userspace
Initialize counter to 100
Workload running?
No
End
Yes
No
The counter is a multiple of 100?
Yes
Monitoring the IPC
Detecting the load
IPC > threshold?
Query the relation table
Yes
No
Set the CPU frequency to the highest one
Set the CPU frequency to the queried value
Counter is incremented by 1
Counter is incremented by 1

each loop. If the judgment is Yes, load detection module detects load every 1s. The decision module queries the relation table to determine the frequency for the load. Frequency scheduling module receives the decision information and conducts the corresponding frequency scheduling operation. If the judgement is No, IPC monitoring module monitors the IPC value every 1ms, and judges whether the IPC value is greater than the threshold, if so, the decision module determines to scale the frequency to the highest, and if not, keep the current frequency. The frequency scheduling module receives its decision information and conducts the corresponding frequency scheduling operation.

5. Case Study

In this section, we demonstrate the two-stage strategy in the Web search system. We use a generate load and a real Google trace load to respectively evaluate the energy-saving effect and QoS-protection capability of the two-phase strategy.

5.1. Evaluation Methodology

Experimental platform: Our experimental platform consists of two servers, one as a client and the other as a server. The two servers are connected via Ethernet. We deployed the Web search benchmark on the platform, which is one of the benchmarks in cloudsuite simulates the real-world client that sends requests to the index nodes. Taking the Web search as workload, we use the two-stage strategy and the Ondemand strategy for CPU frequency scheduling and evaluate their energy-saving effects, respectively. For Web search applications, in order to achieve a good user experience, we set its average response time to be less than 300ms. With this performance constraint, we evaluate the energy-saving effects of the two strategies.

Generating loads: In real production systems, the clients usually send a random number of requests to the server nodes. Thus, we generate two types of workload: Random workload and Google trace workload.

1 Random workload: We have designed a random load generator that is programmed by Python. The generator uses the threading library to generate a random number of threads every second, and uses the urllib2 library in Python to send threads to the server. Figure 5 shows a Random workload executed in 696s.

2 Google trace workload: In the Google trace data, the client generates a Google trace workload. Thus, we use Google trace to generate a Google trace workload. Figure 6 shows a Google trace workload executed in 696s.

5.2. Evaluation Metrics

The objective of the two-stage strategy is to reduce the power consumption of the processor while ensuring the performance of the application. Therefore, we evaluate the two-stage strategy on the power consumption metric and QoS metric.

- Power consumption: we measure the power consumption of the processor every 1s during the workload execution.
- QoS: we obtain the average response time of the application every 1s during the workload execution.

5.3. Comparison

Ondemand is the default strategy for CPU frequency scheduling in the Linux system, which is often used as a baseline to compare with other DVFS-based approaches. In our work, the proposed two-stage strategy dynamically schedules the frequency based on DVFS technique, so we compare the two-stage strategy with Ondemand strategy. We respectively evaluate the two-stage strategy and Ondemand strategy on the two evaluation metrics under two types of workloads.

Figure 5

Random workload executed in 696s. The horizontal axis shows the execution time (second), and the vertical axis denotes the number of requests generated at each second

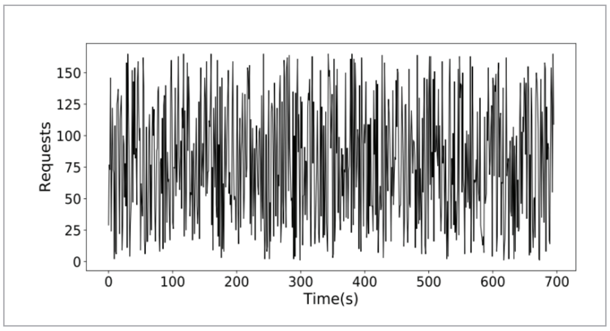
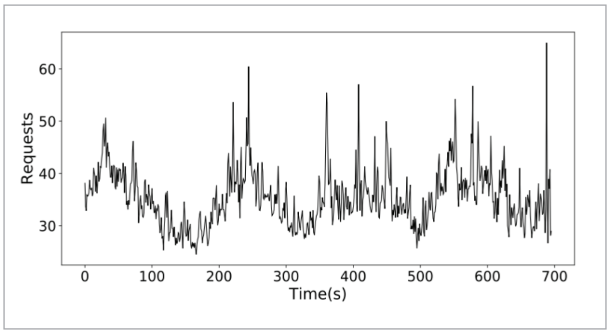


Figure 6

Google trace workload executed in 696s. The meaning of the axes is the same as Figure 5



Evaluations under Random workload: When the client submits the Random workload as shown in Figure 5, we use the two-stage strategy and Ondemand strategy, respectively, to achieve the CPU frequency scheduling. For the two strategies, the measured energy consumption and QoS are shown in Figures 7-8. In both figures, the bottom sub-figure illustrates the QoS per second, and the top sub-figure describes the power consumption per second, where the black line denotes the power measured by the RAPL tool, and the green line denotes the power measured by the Watts Up Pro Power Meter

Figure 7

Power consumption and QoS measured per second during the Random workload execution under the two-stage strategy. Rapl and Watt respectively denotes the power consumption measured by tools of RAPL and Watt

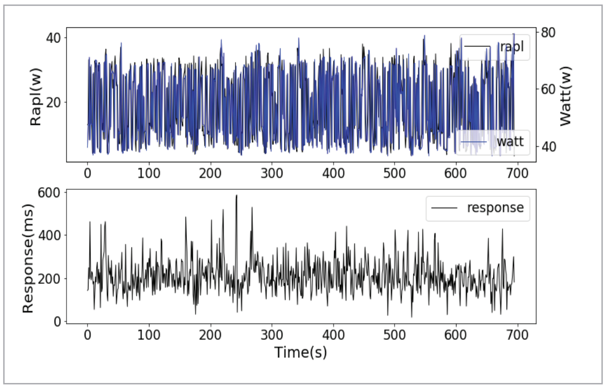
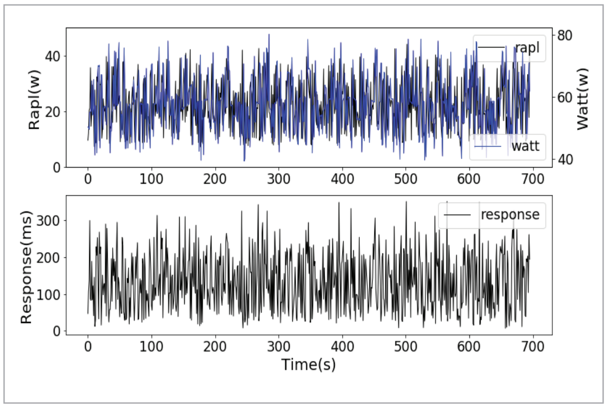


Figure 8

Power consumption and QoS measured per second during the Random workload execution under the Ondemand strategy. The meaning of Rapl and Watt are the same as Figure 7



(Watt in short). It can be seen from Figures 7-8 that under the two-stage strategy, the power consumption per second is less than 40J with the total value of 34587J (measured by RAPL) and 10649J (measure by Watt). While under the Ondemand strategy, the power consumption per second is less than 45J with the total value of 40133J (measured by RAPL) and 15478J (measure by Watt). Compared with the Ondemand strategy, the two-stage strategy reduces energy consumption by 31.2% (processor) and 13.8% (whole server). In addition, we can see from the Figures 9-10 that under the two-stage strategy, the performance per second is less than 600ms with the average value of 251ms. While under the Ondemand strategy, the performance per second is less than 350ms with the average value of 133ms. These results indicate that Ondemand is better than the two-stage strategy in terms of QoS.

Evaluations under Google trace workload: When the client submits the Google trace workload as shown in Figure 6, we use the two-stage strategy and Ondemand strategy respectively to achieve the CPU frequency scheduling. For the two strategy, the measured energy consumption and QoS are shown in Figures 9-10. In both figures, the bottom sub-figure illustrates the QoS per second and the top sub-figure describes the power consumption per second, where the black line denotes the power measured by RAPL tool, and the green line denotes the power measured by the Watt. It can be seen from Figures 9-10 that, under the two-stage strategy, the power consumption per second is less than 20J with the total value of 33529J (measured by RAPL) and 9773J (measure by Watt). While under the Ondemand strategy, the power consumption per second is less than 30J with the total value of 39169J (measured by RAPL) and 14549J (measure by Watt). Compared with the Ondemand strategy, the two-stage strategy reduces energy consumption by 32.8% (processor) and 14.3% (whole server). In addition, we can see from the Figures 9-10 that under the two-stage strategy, the performance per second is less than 400ms with the average value of 227ms. While under the Ondemand strategy, the performance per second is less than 200ms with the average value of 108ms. These results indicate that Ondemand is better than the two-stage strategy in terms of QoS.

Figure 9

Power consumption and QoS measured per second during the Google trace workload execution under the two-stage strategy. The meaning of Rapl and Watt are the same as Figure 7

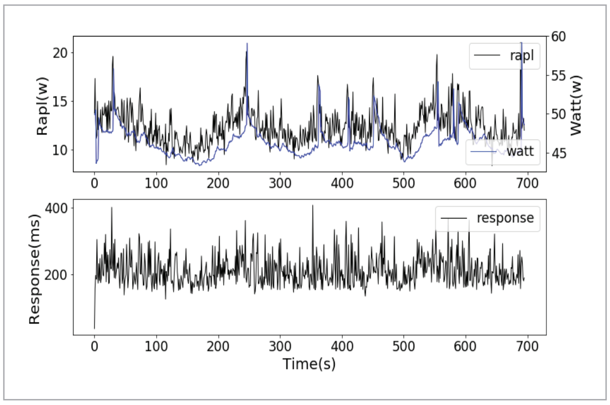
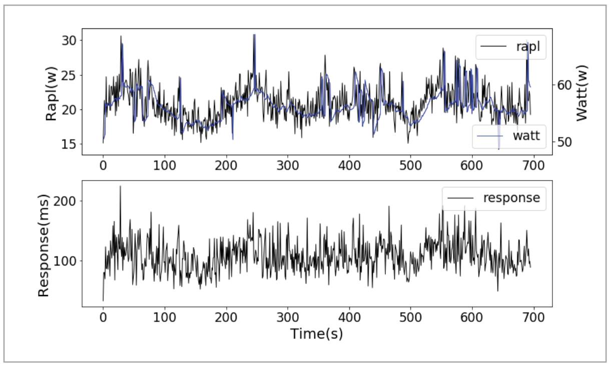


Figure 10

Power consumption and QoS measured per second during the Google trace workload execution under the Ondemand strategy. The meaning of Rapl and Watt are the same as Figure 7



Discussion: From the above two experiments, we can conclude that:

1 the two-stage strategy is more efficient than Ondemand strategy for power saving.

2 the two-stage strategy is not as good as Ondemand strategy in terms of QoS. Ondemand strategy uses a more aggressive frequency scheduling strategy, which schedules the CPU runs at the highest frequency once the CPU utilization exceeds a threshold. Since Ondemand strategy frequently keeps the CPU running at the highest frequency during the load operation, it achieves excellent performance. Nevertheless, the two-stage strategy still works well since it can meet the QoS constraint. The two-stage strategy achieves more energy-savings at the expense of performance. However, this expense is worth it. As long as the QoS constraint is met, users can hardly feel the difference between the extremely high performance and slightly poor performance.

To further compare the difference of the two strategies, we count the frequency scheduling information and analyze the difference. We collect the frequency scheduling information every 1ms on the server and calculate the proportion of all 16 frequencies to the total time in 696s. Figures 11-12, respectively, plot the frequency scheduling information for the two strate-

Figure 11

The ratio of the CPU frequency of the two strategies during a 696s period of the Random workload execution

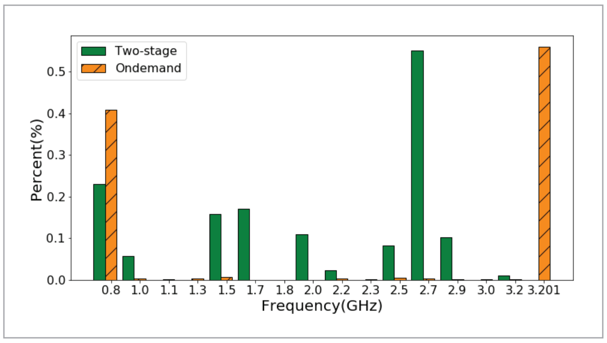
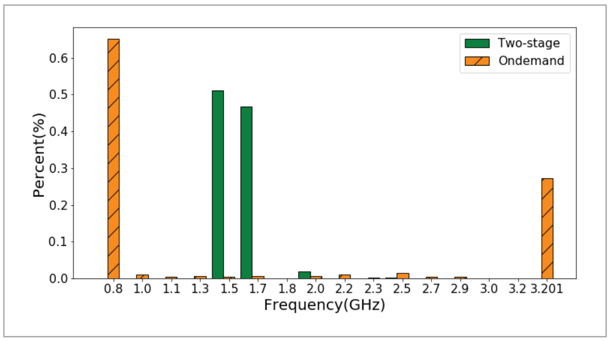


Figure 12

The ratio of the CPU frequency of the two strategies during a 696s period of the Google trace load execution



gies under the Random workload and the Google trace workload. It can be seen from the figures that Ondemand strategy is mostly at the lowest frequency and the highest frequency for both of the two workloads, while the two-stage strategy makes full use of all frequencies. The Random workload contains various size of load, and we can see from Figure 11 that the two-stage strategy controls the CPU runs at multiple frequencies. In the Google trace workload, most of the loads are between 60 to 80, and we can see from Figure 12 that the two-stage strategy schedules the CPU frequency mostly between 1.5GHz and 1.7GHz.

6. Related Work

Liu et al. [17] consider that the energy inefficiency of data centers is mainly attributed to resource under-utilization and heat recirculation. They proposed a thermal-aware and power-aware energy consumption model. Along with the constraints according to the energy consumption model, TSTD algorithm and DVFS technique are used to allocate different computational tasks to the appropriate blades and adjust the running frequency. Krzywda et al. [16] point out that DVFS, CPU pinning, horizontal, and vertical scaling, are four techniques that have been proposed to control the performance and energy consumption on data center servers. Their work investigated the utility of these four techniques for power-performance tradeoffs, such as they report that DVFS rarely reduces the power consumption of underloaded servers by more than 5%, but it can be used to limit the maximal power consumption of a saturated server by up to 20%. Guerreiro et al. [11] studied the energy saving of GPU application using DVFS technique. They proposed several classification models to identification which applications can benefit from DVFS, in terms of energy savings. Work [24] developed a learning-based DVFS framework by using reinforcement learning, makes voltage and frequency (v-f) scaling decisions for multi-core real-time systems. Casu et al. [5] investigated the trade-offs between performance and power saving in a Network-on-Chip (NoC) under three DVFS policies (rate-based, queue-based and delay-based). Through experiments they report that delay-based policy generally offers a better power-performance trade-off.

7. Conclusion and Future Work

Modern data centers run a variety of applications and process large-scale network data every day. With the high-intensity working, data centers afford huge energy costs. Latency-critical workloads have the QoS constraint, which force the server continuously runs at the high CPU frequency, this is one of the main reason for energy cost. This paper proposes a two-stage strategy to trade-off between energy-saving as well as QoS for latency-critical workloads, aims to reduce the energy consumption while guarantying the QoS. The two-stage strategy dynamically schedules the CPU frequency during the workload operation, making decisions about frequency levels that minimize energy consumption for the workload. The core of the two-stage strategy is using a heuristic algorithm to calculate the optimal frequency for the load, and uses a threshold method to fine-grained adjust the frequency to protect the QoS at runtime. The evaluation results show that the two-stage strategy is efficient for energy-saving and QoS-ensuring.

In future work, we intend to the performance and energy consumption in mobile Web applications.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61672421), the Key Research and Development Program of Shaanxi Province (No. 2018ZDXM-GY-036) and Shaanxi Key Laboratory of Intelligent Processing for Big Energy Data (No.IPBED7).

References

1. Al-Jaberi, M., Mohamed, N., Al-Jaroodi, J. E-commerce Cloud: Opportunities and Challenges. 2015 International Conference on Industrial Engineering and Operations Management, (IEOM 2015), Dubai, United Arab Emirates, March 3-5, 2015, 1-6. https://doi.org/10.1109/IEOM.2015.7093867
2. Arroba, P., Moya, J. M., Ayala, J. L., Buyya, R. Dynamic Voltage and Frequency Scaling-Aware Dynamic Consolidation of Virtual Machines for Energy Efficient Cloud Data Centers. Concurrency and Computation: Practice and Experience, 2017, 29(10), e4067. https://doi.org/10.1002/cpe.4067
3. Barroso, L. A., Clidaras, J., Hölzle, U. The Datacenter as a Computer: an Introduction to the Design of Warehouse-scale Machines. Synthesis Lectures on Computer Architecture, 2013, 8(3), 1-154. https://doi.org/10.2200/S00516ED2V01Y201306CAC024
4. Brodowski, D., Golde, N., Wysocki, R. J., Kumar, V. CPU Frequency and Voltage Scaling Code in the Linux(TM) Kernel. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt. Accessed on Nov 10, 2019.
5. Casu, M. R., Giaccone, P. Power-performance Assessment of Different DVFS Control Policies in NoCs. Journal of Parallel and Distributed Computing, 2017, 109, 193-207. https://doi.org/10.1016/j.jpdc.2017.06.004
6. Chard, K., Caton, S., Rana, O., Bubendorfer, K. Social Cloud: Cloud Computing in Social Networks. In 2010 IEEE 3rd International Conference on Cloud Computing, Miami, FL, USA, July 5-10, 2010, 99-106. https://doi.org/10.1109/CLOUD.2010.28
7. Choi, K., Soma, R., Pedram, M. Fine-grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff Based on the Ratio of Off-chip Access to on-chip Computation Times. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 2004, 24(1), 18-28. https://doi.org/10.1109/TCAD.2004.839485
8. David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R., Le, C. RAPL: Memory Power Estimation and Capping. In 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design, (ISLPED 2010), Austin, TX, USA, Aug 18-20, 2010, 189-194. https://doi.org/10.1145/1840845.1840883
9. Dou, H., Qi, Y., Wei, W., Song, H. A Two-time-scale Load Balancing Framework for Minimizing Electricity Bills of Internet Data Centers. Personal and Ubiquitous Computing, 2016, 20(5), 681-693. https://doi.org/10.1007/s00779-016-0941-9
10. Gregg, B. CPU Utilization is Wrong. http://www.brendangregg.com/blog/2017-05-09/cpu-utilization-is-wrong.html. Accessed on Nov 12, 2019.
11. Guerreiro, J., Ilic, A., Roma, N., Tomás, P. DVFS-aware Application Classification to Improve GPGPUs Energy Efficiency. Parallel Computing, 2019, 83, 93-117. https://doi.org/10.1016/j.parco.2018.02.001
12. Ibrahim, S., Phan, T. D., Carpen-Amarie, A., Chihoub, H. E., Moise, D., Antoniu, G. Governing Energy Consumption in Hadoop Through CPU Frequency Scaling: An Analysis. Future Generation Computer Systems, 2016, 54, 219-232. https://doi.org/10.1016/j.future.2015.01.005
13. Janapa Reddi, V., Lee, B. C., Chilimbi, T., Vaid, K. Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency. Proceedings of the 37th Annual International Symposium on Computer Architecture, (ISCA 2010), Saint-Malo, France, June 19-23, 2010, 314-325. https://doi.org/10.1145/1815961.1816002
14. Jararweh, Y., Doulat, A., AlQudah, O., Ahmed, E., Al-Ayyoub, M., Benkhelifa, E. The Future of Mobile Cloud Computing: Integrating Cloudlets and Mobile Edge Computing. In 2016 23rd International Conference on Telecommunications, (ICT 2016), Thessaloniki, Greece, May 16-18, 2016, 1-5. https://doi.org/10.1109/ICT.2016.7500486

15. Kanev, S., Hazelwood, K., Wei, G. Y., Brooks, D. Tradeoffs Between Power Management and Tail Latency in Warehouse-scale Applications. In 2014 IEEE International Symposium on Workload Characterization, (IISWC 2014), Raleigh, NC, USA, Oct 26-28, 2014, 31-40. https://doi.org/10.1109/IISWC.2014.6983037

16. Krzywda, J., Ali-Eldin, A., Carlson, T. E., Östberg, P. O., Elmroth, E. Power-performance Tradeoffs in Data Center Servers: DVFS, CPU Pinning, Horizontal, and Vertical Scaling. Future Generation Computer Systems, 2018, 81, 114-128. https://doi.org/10.1016/j.future.2017.10.044

17. Liu, H., Liu, B., Yang, L. T., Lin, M., Deng, Y., Bilal, K., U. Khan, S. Thermal-aware and DVFS-enabled Big Data Task Scheduling for Data Centers. IEEE Transactions on Big Data, 2018, 4(2), 177-190. https://doi.org/10.1109/TBDATA.2017.2763612

18. Lo, D., Cheng, L., Govindaraju, R., Barroso, L. A., Kozyrakis, C. towards Energy Proportionality for Large-scale Latency-critical Workloads. In 2014 ACM/IEEE 41st International Symposium on Computer Architecture, (ISCA 2014), Minnesota, USA, June 14-18, 2014, 301-312. https://doi.org/10.1145/2678373.2665718

19. PCM. https://github.com/opcm/pcm. Accessed on Nov 12, 2019.

20. Rojas-Cessa, R., Kaymak, Y., Dong, Z. Schemes for Fast Transmission of Flows in Data Center Networks. IEEE Communications Surveys & Tutorials, 2015, 17(3), 1391-1422. https://doi.org/10.1109/COMST.2015.2427199

21. Shea, R., Liu, J., Ngai, E. C. H., Cui, Y. Cloud Gaming: Architecture and Performance. IEEE Network, 2013, 27(4), 16-21. https://doi.org/10.1109/MNET.2013.6574660

22. Shehabi, A., Smith, S., Sartor, D., Brown, R., Herrlin, M. United States Data Center Energy Usage Report. https://www.osti.gov/servlets/purl/1372902/. Accessed on Nov 10, 2019.

23. Sundriyal, V., Sosonkina, M. Modeling of the CPU Frequency to Minimize Energy Consumption in Parallel Applications. Sustainable Computing: Informatics and Systems, 2018, 17, 1-8. https://doi.org/10.1016/j.suscom.2017.12.002

24. ul Islam, F. M. M., Lin, M., Yang, L. T., Choo, K. K. R. Task Aware Hybrid DVFS for Multi-core Real-time Systems Using Machine Learning. Information Sciences, 2018, 433-434, 315-332. https://doi.org/10.1016/j.ins.2017.08.042

25. Vamanan, B., Hasan, J., Vijaykumar, T. N. Deadline-aware Datacenter Tcp (d2tcp). ACM SIGCOMM Computer Communication Review, 2012, 42(4), 115-126. https://doi.org/10.1145/2342356.2342388

26. Wang, P., Qi, Y., Liu, X. Power-aware Optimization for Heterogeneous Multi-tier Clusters. Journal of Parallel and Distributed Computing, 2014, 74(1), 2005-2015. https://doi.org/10.1016/j.jpdc.2013.09.003

27. Wang, X., Qi, Y., Wang, Z., Chen, Y., Zhou, Y. Design and Implementation of SecPod, a Framework for Virtualization-based Security Systems. IEEE Transactions on Dependable and Secure Computing, 2017, 16(1), 44-57. https://doi.org/10.1109/TDSC.2017.2675991

28. Wang, L., Von Laszewski, G., Dayal, J., Wang, F. Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster With DVFS. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, (CCGRID 2010), Melbourne, VIC, Australia, May 10-17, 2010, 368-377. https://doi.org/10.1109/CCGRID.2010.19

29. Watts Up Pro Portable Power Meter. https://www.powermeterstore.com/p1206/watts_up_pro.php. Accessed on Nov 12, 2019.

30. Witham, J. Achieving Data Center Energy Efficiency. https://www.datacenterknowledge.com/industry-perspectives/achieving-data-center-energy-efficiency. Accessed on Nov 6, 2019.

31. Yang, C. Y., Chen, J. J., Kuo, T. W., Thiele, L. An Approximation Scheme for Energy-efficient Scheduling of Real-time Tasks in Heterogeneous Multiprocessor Systems. In 2009 Design, Automation & Test in Europe Conference & Exhibition, Nice, France, April 20-24, 2009: 694-699. https://doi.org/10.1109/DATE.2009.5090754

32. Zheng, P., Qi, Y., Zhou, Y., Chen, P., Zhan, J., Lyu, M. R. T. An Automatic Framework for Detecting and Characterizing Performance Degradation of Software Systems. IEEE Transactions on Reliability, 2014, 63(4), 927-943. https://doi.org/10.1109/TR.2014.2338255