Received May 31, 2017, accepted June 26, 2017, date of publication July 11, 2017, date of current version July 31, 2017.

*Digital Object Identifier 10.1109/ACCESS.2017.2724598*

# A DVFS Based Energy-Efficient Tasks Scheduling in a Data Center

**SONGYUN WANG[1], ZHUZHONG QIAN[2], (Member, IEEE), JIABIN YUAN[1], AND ILSUN YOU[3], (Senior Member, IEEE)**
[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
[2]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 211106, China
[3]Department of Information Security Engineering, Soonchunhyang University, Asan 31538, South Korea

Corresponding author: Ilsun You (isyou@sch.ac.kr)

This work was supported in part by the National Natural Science Foundation of China under Grant 61472181 and in part by the Soonchunhyang University Research Fund.

**ABSTRACT** With the popularity of cloud computing and high performance computing, the size and the amount of the datacenter develop rapidly, which also causes the serious challenges on energy consumption. Dynamic voltage and frequency scaling (DVFS) is an effective technique for energy saving. Many previous works addressed energy-officiate task scheduling based on DVFS. However, these works need to know the total workload (execution time) of tasks, which is difficult for some real-time tasks requests. In this paper, we propose a new task model that describes the QoS requirements of tasks with the minimum frequency. In addition, we define energy consumption ratio (ECR) to evaluate the efficiency of different frequencies under which to execute a take. Thus, it is possible to convert the energy-efficient task scheduling problem into minimizing the total ECR. By transforming the problem to the variable size bin packing, we prove that the minimization of ECR is NP-hard in this paper. Because of the difficulty of this problem, we propose task allocation and scheduling methods based on the feature of this problem. The proposed methods dispatch the coming tasks to the active servers by using servers as less as possible and adjust the execution frequencies of relative cores to save energy. When a task is finished, we propose a processor-level migration algorithm to reschedule remaining tasks among processors on an individual server and dynamically balance the workloads and lower the total ECR on this server. The experiments in the real test-bed system and simulation show that our strategy outperforms other ones, which verifies the good performance of our strategy on energy saving.

**INDEX TERMS** Data center, energy aware, optimal scheduling.

## I. INTRODUCTION

Nowadays, cloud computing provides solutions for scientific and engineering applications while bringing a very large number of electricity energy cost and significant carbon footprints at the same time. The computing resources consume about 0.5% of worldï£¡ï£¡s total power usage [1] and the economic cost of energy in data centers is about $11.5 billion in 2010 [2]. In recent years, instead of the previous focus on system performance, cloud platform designers have begun to concentrate about the issue of power management due to the huge power consumption. Reducing the energy consumption of data centers has even become a primary issue in the design of modern data center [3].

Dynamic Voltage and Frequency Scaling (DVFS), being widely applied in modern processors, is recognized as an effective technique for achieving the tradeoff between system performance and energy saving. With DVFS, the processor could dynamically adjust the working frequency, which leads to different energy consumption. However, "lower frequency fewer energy consumption" is not always true because low frequency increases the task execution time as well as energy depends both on power and execution time. Pervious works [4]–[6] indicate that there is a certain optimal frequency for a given processor, under which the energy consumption of executing a task is minimized.

Generally, letting each processor in a data center works under the certain optimal frequency will result in an overall minimum energy consumption. However, many real-time tasks have the QoS (Quality of Service) requirements. For a batch of coming tasks, if we execute all the tasks under the optimal frequency, several tasks may miss the deadline. Thus, it is a challenge how to allocate tasks to processors and set a suitable frequency for each processor to optimize the total energy consumption. A good DVFS based task allocation strategy should minimize overall energy consumption, as long as meeting the QoS requirement, i.e. achieving tasks in time.

To apply effective task allocation, we need to estimate the workload of each task beforehand, based on which the energy-aware schedulers set the proper frequencies for all the working processors. If the systems could complete all the tasks before their deadlines, each processor should work under the optimal frequency; otherwise, the processors should be set to a higher working frequency to meet their deadlines [4], [7]. Previous works generally adopt the Worst Case Execution Time/Cycle (WCET/WCEC) [4], [6]–[10], the upper bound of execution time under maximum frequency, as the workload of a task. But WCET usually does not match the real running workload of a task, which results in a gap between theoretical results and real energy savings. Therefore, some works [9], [11], [12] make use of probability-based WCET as a workload to improve effectiveness of the task scheduling. However, it is still difficult to get WCET without source code. Even worse, for some long-time running tasks, WCET is unbounded.

Motivated by this, we define a new task model that uses the minimum frequency to represent QoS requirement instead of deadline. That is, if the task is running on the processor that works upon the required frequency, the QoS requirement is satisfied. In addition, we propose Energy Consumption Ratio (ECR) to evaluate energy consumption under different working frequencies compared with the maximum frequency, which indicates the energy efficiency in different frequencies. Thus, the energy-aware task scheduling is converted into minimizing the overall ECR, which is proved analogous to but more difficult than Variable Size Bin Packing problem.

In our previous work [13], we proposed two task-to-processor allocation algorithms to minimize the total ECR while ensuring frequency requirements of tasks when a set of tasks arrive. In this paper, we extends these energy-aware task allocation algorithms. Firstly, we formally prove the hardness of this allocation problem and analyze the performance of the two task allocation algorithms in detail. Secondly, we investigate the runtime task migration problem. During the running time, when a task finishes, the remaining tasks may not be running in an optimal situation. We propose a local task migration algorithm and accordingly adjust frequencies of related processors to reduce the energy consumption, since a running task migrating among processors within a server is quick and low cost. Finally, We improve the real testbed to evaluate our energy-efficient task scheduling mechanism, combining with task allocation and migration. Both real testbed and simulation experiments show that our mechanism may significantly reduce energy consumption while meeting the QoS requirement of tasks.

The main contributions of this paper are as follows:

- This paper proposes a general model to express QoS requirements of real-time tasks, which is more common than workload-deadline model as well as could easily represent tasks with unknown workload and deadline.
- This paper proves that the energy-efficient tasks allocation problem (i.e. minimizing ECR) is NP-Hard, and proposes two effective heuristics to achieve tasks allocation.
- This paper also investigates the runtime optimization, and presents a task migration and frequency readjustment scheme to further reduce energy consumption of the server, on which a task is just finished.
- This paper proposes a prototype system in a real testbed, and conducts some experiments in different cases compared to other algorithms. Results prove that our method is more effective.

The rest of this paper is organized as follows. Section II introduces some related works. Section III presents task model and ECR model. Section IV presents the task allocation algorithms to achieve effective task-to-CPU allocation and local tasks migration schema. In the section V, we evaluate the energy-efficient task scheduling mechanism through simulation and experiments in the real system. Section VI is the conclusion of the paper.

## II. RELATED WORK

DVFS based Energy-efficient task scheduling is widely studied in recent years. The goal of this task scheduling is to optimize the energy consumption while meeting the QoS requirement. Generally, there are two phases, including task-to-processor allocation and frequency scaling. That is, we need to assign each task to a processor and set a proper working frequency to the processor, so that all the tasks could catch the deadline and the overall energy is minimized. In [8], this optimal problem is shown to be NP-hard in the strong sense, if each task has a fixed deadline and the workload is known beforehand. The authors suggested load balance strategy for energy saving, and proposed a Worst-Fit Decreasing (WFD) task allocation scheme to balance the workload and reduce energy consumption. In [9], the authors studied various task partitions and DFS schemes to analyze the effectiveness on energy saving. It is shown that the WFD has good performance for off-line task scheduling, while it does not work well for online task partition.

Actually, when a task is submitted, it is difficult to precisely estimate the total workload before execution. Previous works usually utilized deterministic Worst-Case Execution Time (WCET) to express workload. WCET is the upper bound execution time that the task is running on the processor with maximum frequency [14]. Reference [4] proposed an approximation algorithm with polynomial bounded running time, which has a 1.283 approximation ratio if the cost of turning on/off processor is minor and has a 2 approximation ratio if the cost is non-negligible. Reference [7] presented an approximate scheme in polynomial complexity on the basis of the assumption that the higher workload leads to larger energy consumption in comparison with the lower workload. Reference [6] proposed 2 algorithms to optimize the dynamic energy consumption based on DVFS platform, where processors share the same frequency. Reference [15] presented a lightweight energy-aware task allocation algorithm for multi-core processor. Although it does not have a bounded approximation ratio, it has a good performance according to the experiment results.

WCET is the upper bound of the workload and it sometimes has a big gap with the actual task execution. Some research works regarded the real workload as a random variable and follows some probabilistic distributions based on WCET. Then, the task execution time is divided into bins that has related probabilities to be consumed. Reference [11] investigated the task scheduling with uncertain workload and suggested a new algorithm to unify the intra- and inter-task voltage scheduling into a single optimization problem. In [12], the authors studied energy-aware real-time tasks scheduling with uncertain execution time. Based on probability, this energy-aware task scheduling is converted into a load-balance problem. They designed an algorithm that minimizes the energy consumption as well as guarantees the performance. Reference [16] addressed the Processor and Voltage Assignment with Probability problem with a probabilistic algorithm. These works are more realistic, however, we still need to get the WCET of each task, which actually is not easy without the source code of the task to be performed.

Most of the energy-aware scheduling algorithm based on DVFS technique are designed for single server with multi-processors such as [10] and [17]. Some other works have been proposed for cluster level energy-aware task scheduling. In [18], the authors estimate the required frequency to determine the optimal number of servers to provide services. The idle node will be turned off and the frequency of each server will be scaled to the estimated value for energy saving. In [19], the authors investigated the affect of frequency of server's power and solved the optimal power allocation problem under power budget in server farms. In [20], the authors researched energy saving strategies of real-time tasks on cluster-based multi-cores. They showed that the minimum energy on each island is dependent on the the core number and leakage power. Based on the observation, they gave a polynomial algorithm to obtain the minimum energy.

## III. MODELS AND SYSTEM OVERVIEW

### A. REAL-TIME TASK

Nowadays, tasks with unknown execution time and deadlines are common in data centers, such as web service, Hadoop task scheduler, this paper proposes a DVFS-based scheduling solution which is different to existing works which using Worst Case Execution Time (WCET) as workload. The energy-aware scheduling for this type of tasks makes it significant for service providers to reduce power costs. A task is represented as a two-tuple $(R_i, F_i)$, where $R_i$ is the task release time, $F_i$ is the minimum proportion of frequency to highest profiling frequency. $F_i$ is the frequency requirement, which is the QoS requirements of tasks. That is, if the task is allocated to a processor, the working frequency of the processor should upon this minimum frequency, so that the task could be finished in time. The frequency of a processor indicates the number of clock cycles in a unit time. System should guarantee the clock cycles allocated to $\tau_i$ in a unit time are larger than $F_i$ of maximum frequency, in order to meet the QoS requirement.

We believe that the minimum frequency is a reasonable, feasible and more general model for expressing QoS requirements. Firstly, a task has a fixed number of instructions when it is allocated to a processor. And the number of instructions that can be processed per unit time is different, which results in different execution time at different frequencies. Thus, this is why the frequency can represent the relative task execution time. Secondly, While the total execution time of some long-running tasks is difficult to measure, the minimum frequency used to ensure the execution of these tasks can be easily measured. Finally, resource providers mainly meet the requirement of the volume of CPU, memory, and bandwidth. Actually the bottleneck of some tasks is not the computing, users may need a lower frequency to match the bottleneck of other resources, and avoid wasting the economic cost of computing resources. Service providers can dynamically or statically recommend prices for different frequencies so that the system can use DVFS-based energy-saving strategies to reduce the economic cost of electricity. Users could require tasks' running speed based on the price of services and the property of tasks. Service providers and users have reached an agreement on energy-aware scheduling services.

### B. ECR MODEL

As we known, most processors are structured by Complementary Metal-Oxide-Semiconductor Transistor (CMOS). The energy consumption of CMOS includes dynamic one $P_{dynamic}$ and static one $P_{static}$. According to the previous studies [7], [21], dynamic part of energy consumption is approximately proportional to the square of the voltage $V$ and frequency $f$. Generally speaking, the voltage is linear with the frequency [18]. Consequently, the dynamic energy consumption has a cubic relationship to applied frequency, i.e. $P_{dynamic}(f) = \alpha \cdot f^3$, where $\alpha$ is a coefficient that is related to different processor. The static part $P_{static}$ is basic energy consumption if the processor is running. Therefore, the processor's power consumption can be expressed by the following formula:

$$P(f) = \alpha \cdot f^3 + P_{static}. \tag{1}$$

Under a fix frequency $f$ in $t$ seconds, the energy consumed of a processor $E$ is $P(f) \cdot t$. If one instruction is executed in a clock cycle, then one unit of energy is consumed; otherwise, no energy costs. Let Clock cycle Per Instruction (CPI) is one, i.e. one clock cycle executes one instruction averagely. Then, the energy consumption of one instruction at frequency $f$ is: $E^j = P(f) \cdot \frac{1}{f}$, where $\frac{1}{f}$ is the duration of a clock cycle with frequency $f$. Let $I_t$ is the total instruction set of task $\tau_i$ and the frequency of executing $j^{th}$ instruction be $f^j$, the entire energy consumption of $\tau_i$ is: $E_i(f^j) = \sum_{j \in I_t} P(f^j) \cdot \frac{1}{f^j}$.

According to [15], the total energy can be minimized by Lagrange Multiplier Method. And it shows that if all the instructions are running at the same frequency, the total energy consumption is minimized. Therefore, we set the same frequency $f$ for each instruction, and the total energy consumption of execute $\tau_i$ is $E_i(f) = |I_t| \cdot P(f) \cdot \frac{1}{f}$.

The energy consumption of the task is determined by the applied frequency and the total instruction number. For a given task, the instructions are usually fixed. Although the number of instructions is unknown in our task model, we may compare the energy consumption of running a certain task between the two frequencies. Therefore, we apply the *energy consumption ratio* (ECR) to express the relative energy consumption of task execution at different frequencies in comparison with the maximum frequencies, which is:

$$\begin{aligned} r(f) &= \frac{E_i(f)}{E_i(f_{max})} \\ &= \frac{f_{max}}{P(f_{max})} \cdot (\alpha \cdot f^2 + \frac{P_{static}}{f}). \end{aligned} \tag{2}$$

The $f_{max}$, $P(f_{max})$, $\alpha$, $P_{static}$ are constants to a certain processor, so the ECR are fixed for a given processor. As shown in equation (2), function $r(f)$ is convex, there is an optimal frequency that brings minimum ECR. In real systems, the frequencies that a processor runs are discrete. That is, a processor could choose a set of working frequencies $f_{min} = f_1 < f_2 < \cdots < f_k = f_{max}$, where $f_k$ represent $k^{th}$ available frequency. For simplicity, $f_k$ represent the relative frequency compared to the maximum frequency. Different frequency leads to different ECR, there exists an optimal frequency $f_{opt}$ that has the minimum ECR. For a given processor, it is easy to get the $r(f)$ and $f_{opt}$ through offline energy tests, which are used to achieve energy-aware scheduling.

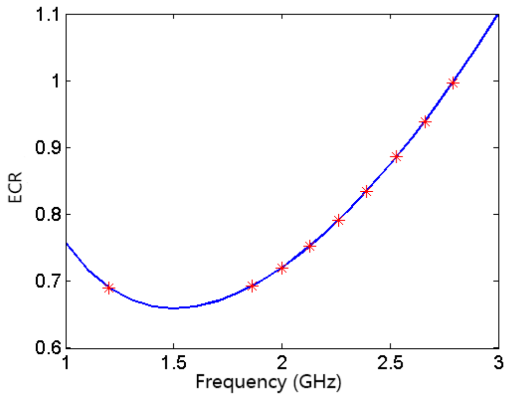

**FIGURE 1.** **ECR function example.**

For clarity, Fig. 1 shows the energy consumption ratio of a processor under different frequencies, and the frequency set is {1.20GHz, 1.86GHz, 2.00GHz, 2.13GHz, 2.26GHz, 2.39GHz, 2.53GHz, 2.66GHz, 2.79GHz}. We can see the ECR of different frequencies is different. When $f = 1.5GHz$ the energy consumption ratio is minimum if frequencies are continuous, while in the case of non-continuous frequencies, the energy consumption ratio can reach a minimum value under the frequency of 1.86 GHz.

### C. SYSTEM OVERVIEW

Our goal is to reduce the energy consumption by energy-aware task scheduling. The ECR cannot represent how much energy consumption working at different frequencies could save, it indicates the energy consumption compared to that of the maximum frequency, which shows how efficiency this frequency is. If the task runs at an optimal frequency, it consume minimum energy to complete this task. Consequently, we should make the tasks working at an optimal frequency as possible as we can. When there is opportunity to change frequency i.e. a task is finished, we will apply task migration within a single server to balance the workload among processors, so that it may achieve a better performance for the remain tasks. Although we do not know the execution time to predict the total energy consumption of the tasks, we could reduce the energy consumption by setting more energy-efficient frequencies in different time segments during the tasks running time. Here, we briefly introduce the basic architecture of energy-efficient tasks scheduling in a data center, which includes the following two parts.

#### 1) ENERGY-AWARE TASK ALLOCATION

Task-to-Processor allocation aims to assign tasks to the available servers to minimize the energy of the cluster to execute these tasks, meanwhile satisfying the requirements of tasks when a set of tasks are submitted to a set of multi-core servers in a data center. In order to predict the energy consumption of different configurations, ECR is used to evaluate the energy consumption of each task and task schedular selects a task allocation with the lowest energy consumption. However, to meet the QoS requirement, the frequency to execute a task should be upon its minimum frequency requirement, so we need to adjust the frequency of the processor considering both new coming and existing tasks. Through comparing all the possible allocations, the incoming task will be assign to one processor with the minimum overall energy cost.

#### 2) LOCAL TASK MIGRATION

When a task is completed on a processor, the workload on this processor will decrease. Therefore, we should adjust the frequency to a more energy-efficient level. On the other hand, if we migrate some tasks to the processor with decreased workload, it is possible to scale down the frequency of multiple processors which will lead to more energy saving. Our main idea is migrating tasks to another processors that have leaving tasks, if the total ECR of these two processors decreases. Through task migration, the total ECR of the related server may reduce.

## IV. ENERGY-EFFICIENT REAL-TIME TASKS SCHEDULING

In this section, we first introduce the task scheduling and frequency scaling on individual processor based on ECR model. Then we explain the energy-aware Task-to-Processor allocation. Finally, we present the local task migration algorithm.

### A. FEASIBILITY TEST OF TASK ALLOCATION

In this section, we introduce 1) the scheduling method on each servers, and 2) which processors can load the coming tasks according to the scheduling method, i.e., feasibility test of task allocation.

Nowadays, the multi-processor architecture is applied wildly. There are two main scheduling methods in multi-processor system: *global* and *partitioning-based* scheduling. There is a single ready queue of tasks in the global scheduling method, tasks will be scheduled to run in the idle processor, which means the task migration among processors is allowed. The method can take the full advantage of each processor, but it is difficult to apply in energy-aware system [22]. The partitioning-based method binds the tasks to only one processor and the tasks will be executed on this processor until finishing. However, this method cannot fully utilize the processor and will causes the wastage of some computing resource [22], [23]. Therefore, we applied compromised scheduling method in multi-processor servers. When tasks come, we allocate the to a processor using energy-aware task allocation method. When a task finishes on a server, the scheduler will make a decision to migrate tasks (Task migration method will be introduced in Section IV-C). This compromised method is easy to deploy and can take the full utilization of each processor and let each processor work under a more energy-efficient frequency.

According to our scheduling method, a processor may have a set of tasks need to be executed. Generally, the real-time task scheduling on single-processor can be divided into 2 categories: priority-driven and share-driven. The priority-driven scheduling is not suitable for the task allocation without task execution time. That is because the long time running task will starve a coexisting task on the same processor. Therefore, we utilize the weighted sharing scheduling, so that the processor have chance to run any task assigned to it. The share-driven task scheduling gives each task $\tau_i$ a weight $w_i$, which expresses the clock cycles in each scheduling period, by which the execution of each task is proportional to the weight $w_i$. To improve the resource utilization, the entire weight of all tasks on the same processor is 1, i.e. $\sum_{\tau_i \in T_c} w_i = 1$ where $T_c$ is a set of tasks that running on this processor $c$. To optimize the energy consumption, the weight of each task will be adjusted when a new task is assigned to this processor.

We will then describe the method how to minimize the energy cost by weight setting. Assume a task $\tau_i$ is running with the weight $w_i$ on a processor at the frequency $f_i$, this task essentially is running at the frequency of $w_i \cdot f_i$. To guarantee the QoS requirement, we need to ensure the clock cycles of each task is more than the clock cycles under required frequency in a scheduling period, i.e.,

$$w_i \cdot f_i \geq F_i$$

Therefore, for a given processor and related assigned tasks, the objective of energy-aware task scheduling within a processor is to optimize the total ECR, which can be formalized as:

$$\begin{aligned} \min \; & \sum_{\tau_i \in T_c} r(f_i) \\ \text{s.t. } & w_i \cdot f_i \geq F_i \quad \forall i \in |T_c| \\ & \sum_{\tau_i \in T_c} w_i \leq 1.0 \\ & w_i \in [0, 1] \\ & f_i \in \{f_{min}, \ldots, f_{max}\} \end{aligned} \tag{3}$$

where $r(f)$ is the energy consumption ratio (ECR), $f_i$ is the frequency to execute task $\tau_i$, $T_c$ is the set of tasks assigned to processor $c$ and $|T_c|$ indicates the task number of the set. To optimize the overall energy consumption, both weight and frequency should be carefully selected, which is non-trivial. In addition, although modern processors may switch working frequency quickly, there still some performance loss and energy cost of such frequency switch. Consequently, we never change the working frequency of a processor if no incoming and departure tasks, we only adjust the corresponding weights of each task to satisfy their QoS requirements. Since each task should meet the constraint $w_i \cdot f_i \geq F_i$, so we have:

$$\sum_{\tau_i \in T_c} w_i \cdot f_i \geq \sum_{\tau_i \in T_c} F_i \tag{4}$$

Given a set of tasks running on processor $c$ with same frequency $f^c$, i.e., $f_i = f_j$, $\forall i, j \in |T_c|$, then working frequency $f^c$ should satisfy the following equation:

$$f^c \geq \sum_{\tau_i \in T_c} F_i \quad \text{s.t.} \, f^c \in \{f_{min}, \ldots, f_{max}\} \tag{5}$$

After determining the frequency on the processor, the minimum required weight of each tasks can also be determined by equation (4), which means that $\tau_j$'s weight is $\frac{F_j}{\sum_{\tau_j \in T_c} F_i}$.

*Theorem 1: Given a task with known WCEC $w_j$ and deadline $d_j$, let the utilization $u_j$ be $u_j = \frac{w_j}{f_{max} \cdot d_j}$. If we set tasks' minimum requirement frequency as $F_j = u_j \cdot f_{max}$, these tasks can be finished before their deadline under our scheduling.*

*Proof:* Assume that all the tasks arrive to a processor $c$ at the same time 0. Let tasks be scheduled by our scheduling method on this processor. According to the equation (6), we have $f^c \geq \sum_{\tau_i \in T_c} F_i = \sum_{\tau_i \in T_c} (u_i \cdot f_{max})$. In a unit time, each task $\tau_j$ can get $f^c \cdot w_j$, where $w_i = \frac{F_j}{\sum_{\tau_j \in T_c} F_i} = \frac{u_j}{\sum_{\tau_j \in T_c} u_i}$ cycles. So the unit time consumed from the arrival of task $\tau_j$ to the end of $tau_j$ is:

$$\begin{aligned} e_j &= \frac{w_j}{f^c \cdot w_j} \\ &\leq \frac{w_j}{(\sum_{\tau_i \in T_c} u_i \cdot f_{max}) \cdot \frac{u_j}{\sum_{\tau_i \in T_c} u_i}} \\ &= \frac{w_j}{f_{max} \cdot u_j} \\ &= \frac{w_j}{f_{max} \cdot \frac{w_j}{f_{max} \cdot d_j}} \\ &= d_j \end{aligned}$$

Therefore, the execution time of task $\tau_i$ is less than the deadline, i.e., tasks can be finished before deadline (arrival time is 0). ∎

Beside, we want to let the processor work under a more energy-efficient frequencies. We already know that if we run a task using the frequencies with lower ECR, the task will consume less energy. On the promise of ensuring the requirement of each task on the same processor, we want to let processor work under a more energy-efficient frequencies, i.e., with lower ECR. According to the equation (2), the function of ECR is convex, there is an optimal frequency $f_{opt}$ that bring minimum ECR. When the frequency is lower than $f_{opt}$, the ECR declines with the increase of frequency. When the frequency is larger than $f_{opt}$, the ECR ascends with the increase of frequency. Therefore, when the minimum required frequency according to equation (6) is smaller than $f_{opt}$, we let the processor run under optimal frequency $f_{opt}$. While the minimum required frequency is larger than $f_{opt}$, we set the running frequency to be the smallest frequency that larger than the minimum required frequency. Thus, the running frequency on each processor is set to be:

$$f = \max(f_{opt}, \sum_{\tau_i \in T_c} F_i), \quad \forall f \in \{f_{min}, \dots, f_{max}\} \tag{6}$$

Assume a task can be allocated to a processor $c$, the frequency of this processor should satisfy the equation (6) i.e. $f \geq \sum_{\tau_i \in T_c} F_i + F_n$, where $\tau_n$ is the new task and $T_c$ represents the set of tasks allocated on processor $c$. The equation can be changed as follow:

$$f - \sum_{\tau_i \in T_c} F_i \geq F_n \tag{7}$$

As long as the frequency $f (f \leq f_{max})$ of processor $c$ satisfies the above equation, the new task can be allocated to this processor and executed under frequency $f$. We call the value of $F_i$ as *Capacity Requirement* for task $\tau_i$. To describe the value of remain capacity of a processor to satisfy the capacity requirements, we define the *Remain Capacity* of processor $c$ at frequency $f$ as:

$$cap(f, c) = f - \sum_{\tau_i \in T_c} F_i \tag{8}$$

When a task comes, these processors whose remain capacities of max frequency are lager than task's capacity requirement can load the coming task.

Based on the above inter-processor task scheduling, we now present the tasks allocation strategy among all processors to optimize the overall energy consumption.

## *B. ENERGY-AWARE TASK ALLOCATION*

### 1) PROBLEM DEFINITION

The problem is to find a feasible task-to-CPU allocation for coming tasks that brings minimum energy consumption to complete these tasks. Because the more energy efficient frequencies a task runs, the less energy it will consume. From this point, we minimize the total ECR to reduce the energy consumption, so we call energy-aware method Optimal Energy Consumption Ratio (*OECR*). We want to address the online task allocation problem based on the Energy Consumption Ratio, which is describe as follow:

Given a set $T$ with $N$ tasks, whose frequencies is represented by $(F_1, F_2, \dots, F_N)$, and a set of homogeneous servers, and each server has $c$ processors whose frequencies can be adjusted independently in $l$ discrete frequencies levels. Each frequency level $f$ has different remain capacity and Energy Consumption Ratio (ECR) $r(f)$. The objective of task allocation is find a task-to-processor assignment in the server set, such that: 1) the active number of servers is minimum, and 2) the total energy consumption ratio of the system is minimized.

*Theorem 2: The online task allocation problem based on the Energy Consumption Ratio is NP-hard.*

*Proof:* If we only minimize the total ECR of tasks and the number of processors are unlimited, the problem can be transformed into the following form: there is unlimited number of processors which have the different size (remain capacity), and the size set is finite. Different remain capacities of processors have different cost. The objective of the problem is to allocate a set of items (tasks) to minimize the total cost. This problem is equivalent to the Variable Sized Bin Packing, which is NP-hard [24].

If we only minimize the number of active servers, we can let all the servers run at the maximum frequency to load the tasks. Therefore, the size of each server is fixed, the objective is to minimize used number of bins (active servers), this problem is equivalent to the bin packing problem, which is NP-hard [25].

Therefore, the online task allocation problem based on ECR is NP-hard. ∎

### 2) TASK ALLOCATION ALGORITHM

The random release time of tasks need us to develop effective online algorithms to schedule the tasks for energy saving instead of offline algorithms. System should rapidly deal with the requests for avoiding influencing the task allocation in next period, and the time complexity of allocation algorithms should be bounded. However, as shown above, the minimization of ECR is NP-hard in the strong sense. So we design a lightweight algorithm to allocate tasks efficiently. In our settings, we allocate a batch of incoming tasks in a unit time. We present two heuristics for a single task allocation and a batch of tasks allocation respectively. The main idea of these two algorithms is to use less active servers and to balance the workload among active servers.

If one task submitted, our allocation algorithm checks whether we could load it at the optimal frequency. The increment of total ECR caused by this task is minimized, if the task could be loaded at optimal frequency. We fully use the optimal frequency $f_{opt}$ of active server to execute tasks, which will minimize the energy consumption of running these tasks. The algorithm for one task allocation is shown in the Alg. 1.

**Algorithm 1** Minimum Energy Consumption Ratio Increment (MECRI)

```
Input:
    Incoming task $(R_i, F_i)$
Output:
    The processor and the optimal frequency loading that new task $(c, f_{opt})$;
1:  if $\max_{c \in C} cap(f_{opt}, c) >= F_i$ then
2:      find the first fit processor $c$
3:      return $(c, f_{opt})$
4:  end if
5:  if $F_i > \max_{c \in C} cap(f_{opt}, c)$ and $F_i <= \max_{c \in C} cap(f_{max}, c)$ then
6:      $C_m = \{c | F_i < cap(f_{max}, c)\}$
7:      for c in $C_m$ do
8:          $\theta_r(c) = (|T_c \cup \tau_i|) \cdot r(f') - |T_c| \cdot r(f)$
9:      end for
10:     The processor with least ECR variation $\theta_r(c)$ is selected
11:     return $(c, f')$
12: end if
13: if $F_i >= \max_{c \in C} cap(f_{max}, c)$ then
14:     The task is allocated to processor $c$ on a new server
15:     return $(c, f_{opt})$
16: end if
```

For an incoming task, the Alg. 1 first compares its capacity requirement to the maximum remaining capacity of the optimal frequency $f_{opt}$ in the processor set $C$. If this maximum remaining capacity is larger than the task's capacity requirement, we use the first fit strategy to select a processor of $\tau_i$ (line 1–4). Otherwise, if the maximum remaining capacity of $f_{max}$ satisfies $\tau_i$'s capacity requirement, the algorithm calculates the each corresponding increment of ECR when the task is allocated to a processor. The algorithm selects the processor for the task that will bring the minimum increment of ECR. The frequency $f'$ (line 7) is determined to be the minimum frequency guaranteeing the requirements of all the tasks on processor $c$, including the particular task $\tau_i$. $f'$ can be easily calculated by equation (9), as we record the sum of ECR for each processor. The frequency of tasks on the same processor is same, so are the ECRs of these tasks. The total ECR on the processor is equal to $r(f) \cdot |T_c|$. The algorithm calculates the total ECR for each processor, which can reflect the number of tasks running on it. The processor having more tasks or working under higher frequencies than $f_{opt}$ will obtain the higher total ECR, which means that the processor has higher workloads. A new task may bring more increment on ECR on this processor, which may lead to the processor working under inefficient frequency for a long time. If the two processors have the same ECR under current frequency, the algorithm tends to allocate the processor with less tasks. If the total capacity requirements are stored using suitable data structure, the difference of ECR under two frequencies can be calculated in $O(1)$. Therefore, the total time complexity in this step is $O(|C|)$, where $|C|$ is the number of processors in active servers. If there is no processor can load this coming task, the algorithm will start a new server to load this task. As a result, the total time complexity of Alg. 1 is $O(|C|)$. After selecting the processor for the coming task, we can change the frequency and recalculate the weights of tasks on this processor according to the frequency requirements.

The number of tasks coming in a unit time is usually larger than one, e.g. the MapReduce job usually exposes many subtasks at a time. As we describe above in this section, the problem of task-to-CPU allocation to minimize the total ECR is NP-hard in the strong sense. It is impossible to come up with an efficient online algorithm to find the optimal solution. An alternative is to sort the tasks first and use the Alg. 1 to allocate tasks one by one. This strategy is based the First Fit Decreasing (FFD), which is shown that FFD uses the $\frac{11}{9}OPT+1$ bins (servers) to load the items (OPT is the number of bins used by the optimal solution). The time complexity is still high when there are a larger number of tasks to be allocate in a short time. The datacenter usually has to deal with the scenario that a large number of tasks come in a short time. If we allocate tasks one by one using Alg. 1, then the algorithm would have $O(|N|\log|N| + |N| \cdot |C|)$ time complexity, where $|N|$ is the number of tasks coming in a unit time. This time complexity is intolerable for large size datacenters and may spend much time to assign tasks, which may in turn affect the allocation of in subsequent rounds. For better scalability, we propose a lightweight algorithm to allocate a batch of tasks as shown in Alg. 2. The main idea is to allocate the task with small requirement onto the processors whose optimal frequency can load them by Best Fit strategy, and load the big tasks using the Worst Fit Dereasing (WFD) strategy. The Best Fit strategy could allow more tasks to work under the optimal frequency and reserve large capacities under the optimal frequency for tasks with high requirements. The WFD strategy can balance the workloads, which in turn leads to good performance in energy saving [8].

The Alg. 2 sorts the set of tasks in a non-decreasing order of capacity requirements (line 1). In the first phase (line 2–4) of the algorithm, it uses the optimal frequency to load the new tasks. The algorithm allocates the tasks form the first task and allocate them to the last processor in $C$ whose remaining capacity of optimal frequency is larger than task's requirement (Best Fit strategy). In this phase, because the processor set have been sorted, we can find the best processor for a task in $O(log|C|)$ time complex by binary search. Therefore, the time complexity of first phase is $O(|C|\log|C| + |T|\log|T| + |T|\log|C|)$, where $O(|C|\log|C|)$, $O(|T|\log|T|)$ and $|T|\log|C|$ are the time complexity of task sorting, processor sorting and task allocation.

In the second phase (line 5–15), the algorithm utilizes the Worst Fit Decreasing (WFD) strategy to allocate the remaining tasks in the set, which first allocates the task with highest requirement. As the tasks have been sorted in the first phase, we only need to allocate the remaining tasks from the head of the sorted set. We sort the set of processors

**Algorithm 2** Power-Aware Task Allocation for Batch Tasks, PTAB

```
Input:
    A batch of tasks T
Output:
    Task partition
1: sort T in a non-decreasing order of $F_i$, where $\tau_0$ has minimum requirement
2: $C = \{c|cap(f_{opt}, c) > F_0\}$
3: sort C in a non-decreasing order of $cap(f_{opt}, c)$
4: allocate the tasks using Best-Fit heuristic until the tasks can be allocated under the optimal frequency $f_{opt}$ and remove the allocated tasks
5: Let $T_r$ be the remain tasks that have not allocate in the step 4
6: $C = \{c|cap(f_{opt}, c) > F_i\}$, where $\tau_0$ has minimum requirement in remain tasks
7: sort C in a non-decreasing order of $N_c \cdot r(f)$
8: for k = $|T_r|$ to 1 do
9:     if the first $log|C|$ processors cannot hold $T_r$[k] then
10:        start an idle server
11:        insert all processors into the front of $C$
12:    end if
13:    allocate $T_r$[k] to the first processor $c$ in first $log|C|$ processors could load $\tau_i$ in $C$
14:    insert processor c into correct position using binary search
15: end for
```

in a non-decreasing order of total Energy Consumption Ratio (ECR), which is the product of task number and ECR of current frequency on processor $c$. We sort the processor according to the total ECR instead of the remain capacity of maximum frequency. In the second phase, the algorithm allocates form the end of task set. When allocating a task, the $log|C|$ first processors with least total ECR will be tried to load the task. The task will be allocated the first processor which can load it in the $\log|C|$ processor. If these $\log|C|$ processors cannot load the tasks, a new server will be started and the processor will insert into the front of processor set $C$. After allocating the biggest task, the capacity of the selected processor will reduce and the total ECR will increase. The processor could be inserted into the correct position in the set of processors by binary searching (line 9–14). The cost of maintaining the sorted list of processors is $O(\log|C|)$. And the time complexity of this phase is sum of cost in sorting the processors, allocating each task by trying $log|C|$ first processors and maintaining the order of processor set, which is $O(|C|\log|C| + |T|\log|C| + |T|\log|C|)$. Therefore, the total complexity of Alg. 2 is $O(|T|\log|T| + 2(|T| + |C|)\log|C|)$, which is more lightweight.

### *C. LOCAL TASK MIGRATION*

When a task finishes, the total ECR of current partition of tasks may not be optimal. The processor having tasks leave may have more remain capacity to load the task on processors and make the total ECR of these processors become lower. Although the ECR can describe the relative energy consumption in some degree, it can not represent how much energy a task consume under different frequencies. A task may experience some segments under different frequencies, the real total energy consumption of a task is equal to the sum of energy consumption in all the segments the task experiences. Unfortunately, we do not know the accurate length of each segments beforehand i.e. the execution time. On the other hands, the minimum ECR of the whole cluster does not mean the minimum energy consumption. This problem is caused by the unknown execution time essentially.

As shown in the Fig. 2, there are two tasks $\tau_1$ and $\tau_2$ running on $c_1$ and $c_2$ respectively in the original task partition. Their ECRs under the current frequencies are both equal to 0.7. Then a new task $\tau_3$ comes, $\tau_3$ can be allocated to $c_1$ or $c_2$, which brings $0.7 + 0.9 \times 2$ and $0.7 + 0.8 \times 2$ of ECR respectively. Due to smaller increment of ECR, $\tau_3$ will be allocated to $c_2$. After a short time, $\tau_1$ will finish and the $c_1$ will be idle. However, the processor $c_2$ will still work under a less effective frequency for two tasks. If we allocate $\tau_3$ to $c_2$, $\tau_3$ will be executed under an effective frequency after $\tau_1$ finishes. The second allocation may bring less energy consumption that the first one. Unfortunately, we do not exactly know when the the tasks finish, so we can not calculate exact energy consumption of different allocations. However, if we migrate the task $\tau_3$ to $c_1$ when $\tau_1$ finishes in the first allocation, the total energy can be reduce.

To amend the negative effectiveness of original task allocation, we apply task migration strategy to adjust the task allocations for more energy saving. When a task finishes on a processor, the total capacity requirement of tasks on this processor will reduce, so the frequency of this processor has the chance to be scaled down for reducing total ECR. But the frequency on the other processors could not be scaled down because the capacity requirements have no changes. Some long tasks on the other processors have to be execute under less effective frequencies for a long time. If we migrate some of tasks to the processor that a task completes just now, the total ECR may be decreased. However, the migration between different servers will cost much time and resource wastage(eg. memory copy, network transmission and context recovery). Besides, the optimal migration is also difficult to be applied, it is not a wise choice to migrate the tasks between different servers. Thanks to the multi-processor architecture, the migration between the processors on the same server takes negligible costs. Meanwhile, the number of processors and tasks on a server are usually not large, so the migration on the host is a feasible strategy for task scheduling and we propose our method of task scheduling in the Alg. 3.

To avoid the frequency invokes of migration due to the completion of tasks in a short time, we set a threshold. If the time interval of two migration invokes is less than the threshold, the task scheduler scales the frequency according to equation 8 but not migration. The migration will be conducted

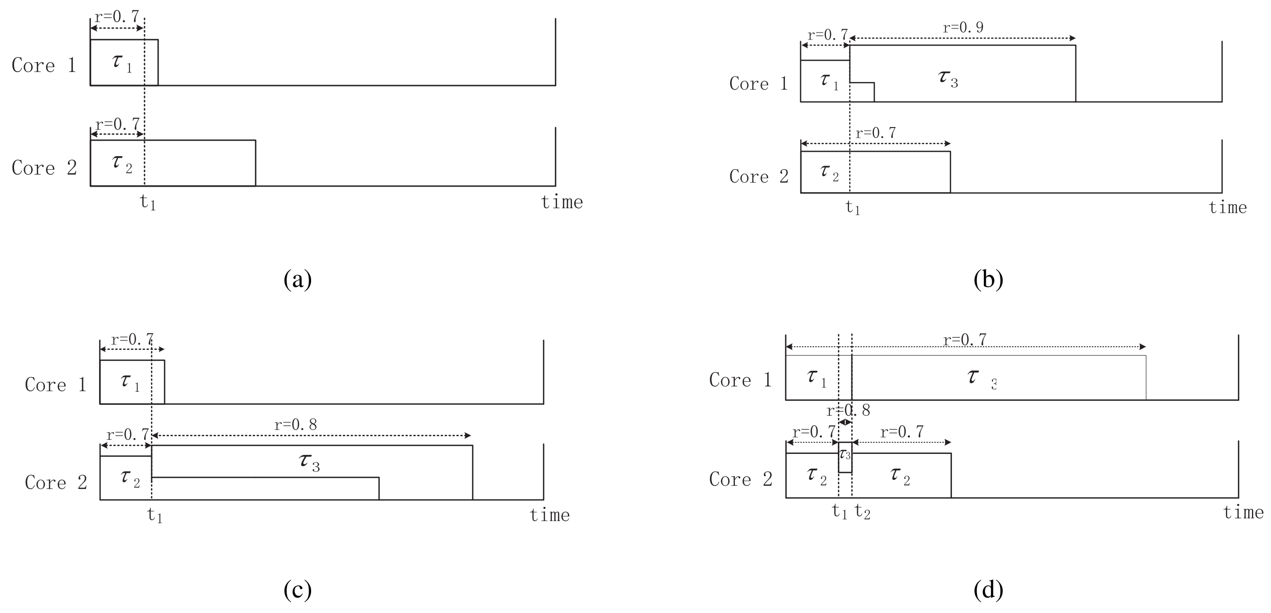

**FIGURE 2.** Allocate a task $\tau_3$ in a two-processor system at time $t_1$. $\tau_3$ is allocated to $c_2$ because of the smaller sum of ECR. However, the real energy consumption in (b) may be less than the real energy consumption (c). If we move $\tau_3$ from $c_2$ to $c_1$ and adjust frequencies at $t_2$ like (d), more energy may be saved.

until the time interval between current time last migration is large than the threshold. Tasks completions may happen in some processors, so the input of Alg. 3 is the set of processors that have task completion. The algorithm divides the processors into two sets i.e. the tried set $C_t$ and untried set $C_u$. The tried set is the processors that have be tried to migrate tasks to other processors or have task leaving, while the untried set stores the untried processors. Our algorithm first sort the set $C_t$ in the non-decreasing order, and tries the migration of tasks according this order of processors. The algorithm selects the processor that have the maximum sprocessor as the source processor and tries to migrate a task to target processor. If the migration will brings the decrease of total ECR of the two processors, the processor will select as a candidate (line 9–13). After comparing all possible migrations to the processors in tried set, the algorithm will select the processor that bring maximum decreasing on ECR to migrate the task and add the processor c to the set $C_t$. This algorithm tries migrate the tasks on the processor with high total ECR to 'tried processors' to lower the total ECR. The processor with high total ECR and the task with high frequency requirement will be tried to migrate with a high priority. By doing so, we want to lower more total ECR.

## V. EVALUATION

In this section, we evaluate the performance of our strategies on energy saving in clusters in real testbed and simulations, respectively. The results show that our strategies can achieve significant energy savings.

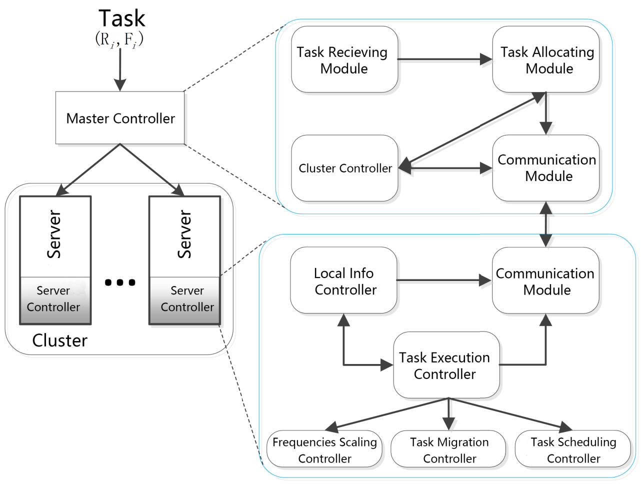

**FIGURE 3.** Prototype system architecture.

### A. PROTOTYPE SYSTEM

The overall structure of the system shown in Fig. 3. This article uses the traditional master-slave frame structure, and every slave server is connected to the master node.

The system is realized with C++ in Linux, consisting of two main components, the master controller and server controllers. The master controller mainly contains the task receiving module, the task information module, the task assignment module and communication module.

According to requests that users submit such as CPU frequency, memory space, network bandwidth, disk size and

**Algorithm 3** Local Task Migration

```
Input:
    The processor $c_f$ where has completed a task just now
Output:
    The migration solution.
1:  $C_u = C - c_f$
2:  put $c_f$ into a tried set $C_t$
3:  for k = 1 to $|C| - 1$ do
4:      select a untried processor $q$ with maximum value of
        $r(c,f) \cdot |T_c|$
5:      sort tasks $T_c$ on $c$ in non increasing order
6:      for each task $\tau_i \in T_c$ do
7:          $max = 0$
8:          for each processor $q \in C_t$ do
9:              $s_a$ indicates the ECR sum of processor $c$ and $q$
10:             if $F_i < cap(c, f_{max})$ then
11:                 $s_b$ indicates the ECR sum of processor $c$ and $q$
                    if task $\tau_i$ migrates from $c$ to $q$
12:             else
13:                 $s_b$ to be the maximum value
14:             end if
15:             if $s_a - s_b > max$ then
16:                 $max = s_a - s_b$, and store $q$
17:             end if
18:         end for
19:         put processor $c$ into $C_t$
20:         if $max > 0$ then
21:             migrate $\tau_i$ from $c$ to $q$
22:         end if
23:     end for
24: end for
```

other information, the master controller deploys tasks to the appropriate server's processor and synchronously updates data in the cluster based on the task allocation algorithm. This paper focuses on the processor resources, in each server, there are a number of modules controlling task execution, frequency adjustment, and task scheduling of processors.

## B. REAL TESTBED RESULT

As shown in Fig. 4, we implemented our algorithms in the real system to evaluate the performance of our strategies on energy conversation. We use three Dell R720 servers to build our experiment system. The R720 server has two Intel Xeon processors and the frequency of each processor can be adjusted independently from 1.2 GHz to 2.1 GHz, plus a turbo mode. The ECRs of each frequency are obtained by offline experiments. The details of server R720 are shown in the Tab. 1. To evaluate the energy consumption precisely, we turn off the hyper-threading on each server. We use the Aitek Power Analyzer AWE2101[1] to record the power consumption, according to which the energy consumption can be calculated.

[1]http://www.aitek.tw/BIG5/awe2101.asp

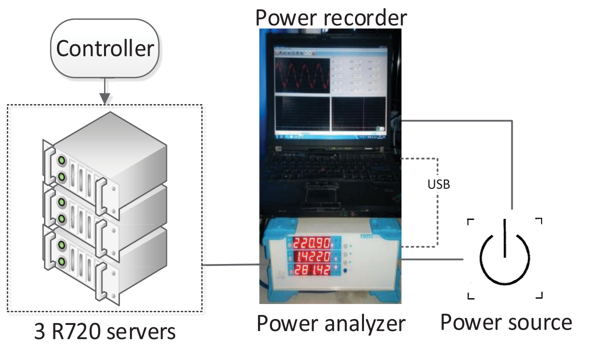

**FIGURE 4.** Real system architecture.

**TABLE 1.** Details of Dell R720 server.

| | |
|---|---|
| Name | Dell PowerEdge R720 |
| CPUs | Two Intel Xeon E5-2620 @2.1GHz |
| Frequency Steps (GHz) | 1.20, 1.30, 1.40, 1.50, 1.60, 1.70, 1.80, 1.90, 2.00, 2.10 |
| Memory | 64G ECC DDR3 |
| Disk | One 10k SAS, 300GB |
| Operation System | CentOS 6.5 |

We use NAS Parallel Benchmarks 3.3 (NPB) to generate the testing tasks, which is widely used to simulate the real cluster tasks. The tasks are generated with different classes of NPB 3.3 for outputting with different execution time. Meanwhile, the arrival time of tasks are generated following a Poisson Distribution with average arriving interval in one minute.

The mechanism mentioned in this paper is called OECR (Energy Consumption Ratio), that is, when there is only one task in a unit time, the single task allocation algorithm is called, meanwhile batch task allocation algorithm is called when there are multiple tasks between time units. When a task is over, system uses the task migration algorithm to adjust the CPU frequency and tasks., we also realize a variety of traditional algorithms for comparing with the strategy proposed in this paper, such as First Fit Decreasing (FFD), Worst Fit Decreasing (WFD), RESERVATION (RES) [3], Modified Best Fit Decreasing (MBFD) [5].

Firstly, we compared the efficiency of the task migration algorithm on one single server. In the benchmark set, a set of task sequences are randomly generated, and the arrival time of tasks obeys the Poisson Distribution with an average arrival rate of 5/min.

Here the *OECR* and *OECRv2* represent the strategies using and without using the local migration algorithm, and compared with FFD algorithm as shown in Fig. 5 (a) below. It shows that the migration of tasks plays a certain effect for energy-saving. With increasing of the task number, the effect of migration on energy savings is more obvious.

At the same time we also compare the energy saving effect of the single task allocation algorithm (MECRI) and the batch task allocation algorithm (PTAB) with the energy saving effect of FFD in the case of a batch of tasks arriving at the same time, As shown in Fig. 5 (b), we compare the results in cases of different task numbers. It shows that when the task

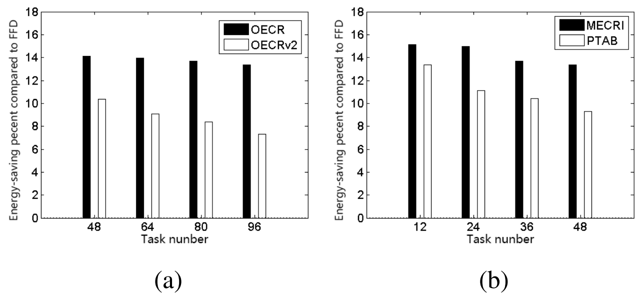

**FIGURE 5. Energy saving effect in a single server. (a) Task migration. (b) Single task vs. Batch tasks.**

number is relatively small, the two algorithms lead to good results, that's because they can have tasks working in more energy-efficient frequencies, while the FFD algorithm may focuses tasks on some processors, increasing the total time the server completes this tasks. In addition, tasks is not working at the best energy-saving frequency. When the number of tasks are large, single task allocation algorithm performs better than the batch algorithm.

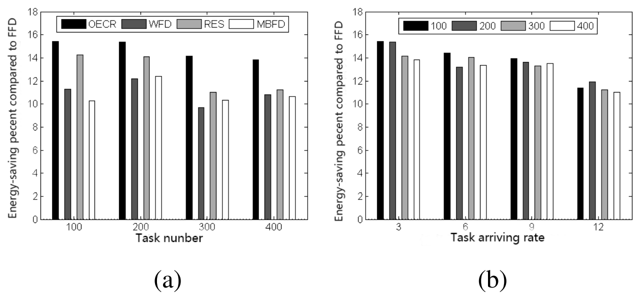

**FIGURE 6. Energy saving effect in three servers. (a) Energy saving effect. (b) Different arriving ratio.**

In the real environment of three servers, we compare the effect for energy saving at different arrival rates. The task arrival time and frequency requirements are still the same. The energy-saving effect of each algorithm in the actual environment is shown in Fig. 6. Fig. 6 (a) shows the difference of the energy consumption of above algorithms with the FFD algorithm. It can be seen that the algorithm proposed in this paper achieves the best results in cases of different task numbers. In Fig. 6 (b), we compare the energy efficiency of the OECR algorithm with respect to the FFD algorithm at different task arrival rates and task numbers, in the best case, our algorithm can save more than 15% energy. With increasing of the number of tasks and the arrival rate, however, energy-saving effect has declined compared with FFD algorithm. It may due to the large number of arrival tasks, the server almost keeps in a full load state, resulting in reduction of the static power energy consumption of the server.

In addition, we also compare loaded task numbers using different algorithms. In the real environment, the number of servers is limited, when the number of tasks, the arrival rate and the frequency requirements vary, the task number server can host is different using different algorithms, because the minimum frequency requirements of tasks needs to be guaranteed. It's shown in Fig. 7. We can see that FFD algorithm can host the most tasks, meanwhile our proposed strategy performs better than WFD and MBFD in terms of the number of tasks accepted.

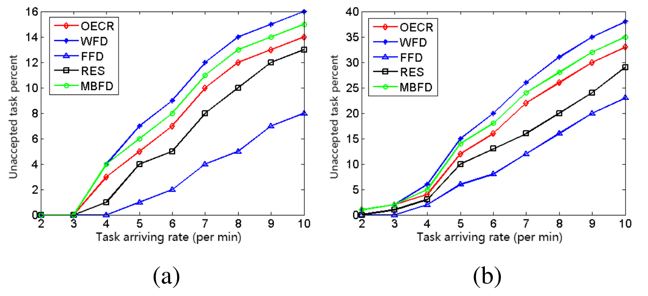

**FIGURE 7. Proportion of unaccepted task number in different arriving ratio. (a) 100 Tasks. (b) 200 Tasks.**

## C. SIMULATION RESULT

We perform some simulations to evaluate our algorithm in the large-scale DC because the complexity of a actual large cluster. The simulation involves different scales of clusters, different task numbers and different arriving rates. We model the R720 server whose details are shown in Tab. 1 to build the simulated clusters. What's more, we also compared the execution time of our method with other algorithms, where we generate the arriving time and the minimum frequency requirement as the pattern in the real environment.

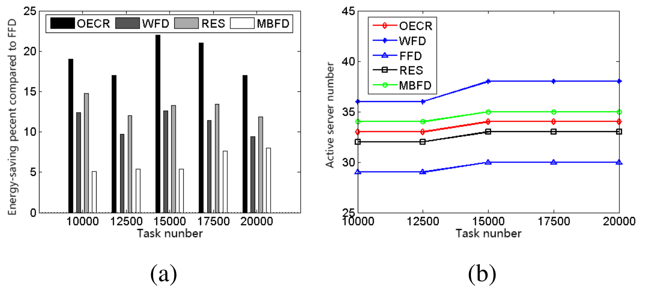

**FIGURE 8. Energy saving effect and active server number when erver number is unlimited. (a) Energy saving effect. (b) Active server number.**

In Fig. 8, this paper compares results of different algorithms in terms of energy saving and the servers usage in the case of low arrival rate. The time distribution of task arriving in Fig. 8 satisfies the Poisson Distribution with an average arrival rate of 10/min, and the maximum frequency of the task execution time is a random value between 60s to 3600s, where the task executing time is mainly used to simulate the execution of tasks.

In Fig. 8 (a), we can see that the task allocation strategy and migration strategy proposed in this paper can save about 20% energy, which is better than other allocation strategies.

As the number of tasks increases, energy efficiency of our strategy is relatively stable. In Fig. 8 (b), we compare the number of servers used of different algorithms in cases of different number of tasks, the FFD algorithm uses the least number of servers, but the performance of FFD is poor in terms of energy savings. The FFD algorithm focuses the task on a small number of servers resulting in high load on these servers, which leading to a certain amount of energy waste. With the increase in the number of tasks, the number of servers used has little change, mainly because the task arriving rate is low in our experimental settings. When some tasks arrived, previous tasks has been completed, so that these tasks can be assigned to active servers.

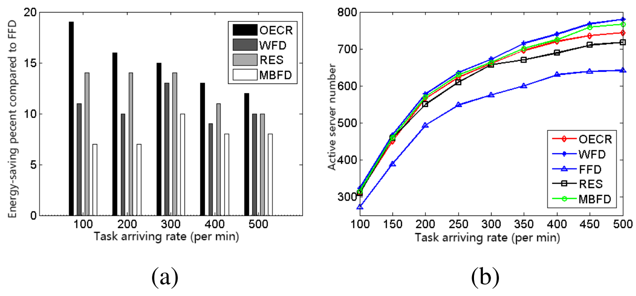

**FIGURE 9.** **Different task arriving ratio when erver number is unlimited. (Total task number is 20000). (a) Energy saving effect. (b) Active server number.**

We also experimented with a high task arriving rate and results are shown in Fig. 9. It can be seen that our proposed strategy achieves the best energy saving effect at different task arriving rates. However the energy saving effect has been reduced the task arriving rate increases compared with the FFD algorithm. The main reason is the difference of numbers of servers used between different algorithms and FFD results become obvious, as shown in Fig. 9 (b), the proportion of static power consumption of the server itself increases. On the other hand, the number of servers used becomes flattened with increasing the task arriving rate. As can be seen from the comparison in Fig. 8 (b) and Fig. 9 (b), the task arriving rate has a great influence on number of servers.

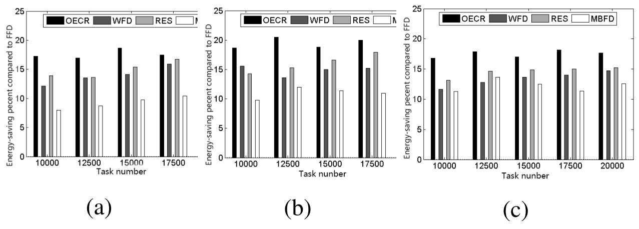

**FIGURE 10.** **Energy saving effect when erver number is limited. (a) 30 Servers. (b) 65 Servers. (c) 100 Servers.**

At the same time, we compared energy saving effect of different algorithms when the number of servers is fixed, as shown in Fig. 10. The number of tasks performed among clusters of different sizes is same, but different the task arriving rate, 10/min in (a), 20/min in (b) and 40/min in (c). We can see that the algorithm proposed in this paper has achieved better energy saving effect in clusters of various sizes.

## VI. CONCLUSION

In this paper, we study the energy-efficient task scheduling, where the task execution time is unknown. We define a novel task model to describe the tasks and the energy consumption ratio to describe the effectiveness of different frequencies. We prove that the task allocation is similar to the variable size bin packing problem and is more complex than it. Then, we present two effective heuristics to allocate tasks. We also design a local task migration algorithm to improve the performance when a task finishes. Finally, this paper introduce a prototype system to evaluate our strategies, which obtains a good performance in terms of energy saving.

## REFERENCES

[1] W. Forrest, "How to cut data centre carbon emissions?" in *Proc. Website*, Dec. 2008. [Online]. Available: http://www.computerweekly.com/feature/How-to-cut-data-centre-carbon-emissions

[2] R. Buyya, A. Beloglazov, and J. Abawajy. (2010). "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges." [Online]. Available: https://arxiv.org/abs/1006.0308

[3] A. Beloglazov *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Adv. Comput.*, vol. 82, no. 2, pp. 47–111, 2011.

[4] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo, "Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems," in *Proc. Real-Time Embedded Technol. Appl. Symp.*, 2006, pp. 408–417.

[5] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *Proc. IEEE RTCSA*, Aug. 2007, pp. 28–38.

[6] V. Devadas and H. Aydin, "Coordinated power management of periodic real-time tasks on chip multiprocessors," in *Proc. Green Comput. Conf.*, 2010, pp. 61–72.

[7] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems," in *Proc. DATE*, 2009, pp. 694–699.

[8] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proc. IEEE IPDPS*, Mar. 2003, p. 9.

[9] T. AlEnawy *et al.*, "Energy-aware task allocation for rate monotonic scheduling," in *Proc. IEEE RTAS*, Apr. 2005, pp. 213–223.

[10] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1540–1552, Nov. 2008.

[11] C. Xian and Y.-H. Lu, "Dynamic voltage scaling for multitasking real-time systems with uncertain execution time," in *Proc. ACM GLSVLSI*, 2006, pp. 392–397.

[12] C. Xian, Y.-H. Lu, and Z. Li, "Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time," in *Proc. ACM DAC*, 2007, pp. 664–669.

[13] M. Zhang, S. Wang, G. Yuan, Y. Li, and Z. Qian, "Energy-efficient real-time task allocation in a data center," in *Proc. IEEE Smart Data (SmartData)*, Dec. 2016, pp. 680–687.

[14] R. Wilhelm *et al.*, "The worst-case execution-time problemłoverview of methods and survey of tools," *ACM TECS*, vol. 7, no. 3, p. 36, 2008.

[15] D. Li and J. Wu, "Energy-aware scheduling for aperiodic tasks on multi-core processors," in *Proc. IEEE ICPP*, Sep. 2014, pp. 361–370.

[16] J. Niu, C. Liu, Y. Gao, and M. Qiu, "Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems," *IEEE IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 8, pp. 2043–2052, Aug. 2014.

[17] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Proc. IEEE IPDPS*, May 2009, pp. 1–12.

[18] E. M. Elnozahy, M. Kistler, and R. Rajamony, *Energy-Efficient Server Clusters*. Berlin, Germany: Springer, 2003, pp. 179–197.
[19] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," *ACM SIGMETRICS*, vol. 37, no. 1, pp. 157–168, 2009.
[20] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores," in *Proc. DATE*, 2011, pp. 1–6.
[21] L. Wang, G. Von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," in *Proc. IEEE CCGrid*, May 2010, pp. 368–377.
[22] J. Goossens, S. Baruah, and S. Funk, "Real-time scheduling on multiprocessor," in *Proc. 10th Int. Conf. Real-Time Syst.*, May 2002, pp. 368–377.
[23] J. M. López, M. García, J. L. Diaz, and D. F. Garcia, "Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems," in *Proc. 12th Euromicro Conf. Real-Time Syst. (Euromicro RTS)*, 2000, pp. 25–33.
[24] D. K. Friesen and M. A. Langston, "Variable sized bin packing," *SIAM J. Comput.*, vol. 15, no. 1, pp. 222–230, 1986.
[25] R. G. Michael and S. J. David, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA, USA: W. H. Freeman Company, 1979, pp. 90–91.

**SONGYUN WANG** is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include distributed system and cloud computing.

**ZHUZHONG QIAN** (M'09) received the Ph.D. degree in computer science in 2007. He is currently an Associate Professor with the Department of Computer Science and Technology, Nanjing University, China. He is a Research Fellow with the State Key Laboratory for Novel Software Technology. He is also the PI and a Chief Member of several national research projects on distributed computing and networking. He has authored over 60 research papers in related fields. His current research interests include cloud computing, distributed systems, and datacenter networking. He is a member of the ACM.

**JIABIN YUAN** is currently a Professor with the Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He  is also a Research Fellow with the Laboratory for Grid and Cloud Computing. His research interests include cloud computing, cryptography, and distributed system.

**ILSUN YOU** (SM'13) received the M.S. and Ph.D. degrees in computer science from Dankook University, Seoul, South Korea, in 1997 and 2002, respectively, and the second Ph.D. degree from Kyushu University, Japan, in 2012. From 1997 to 2004, he was with THINmultimedia Inc., Internet Security Co., Ltd., and Hanjo Engineering Co., Ltd., as a Research Engineer. He is currently an Associate Professor with the Department of Information Security Engineering, Soonchunhyang University. His main research interests include internet security, authentication, access control, and formal security analysis. He is a fellow of the IET. He has served or is currently serving as a Main Organizer of international conferences and workshops, such as MobiWorld, MIST, SeCIHD, AsiaARES, IMIS, and so forth. He is the EiC of *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*. He is in the Editorial Board for *Information Sciences*, the *Journal of Network and Computer Applications*, the IEEE ACCESS, *Intelligent Automation & Soft Computing*, the *International Journal of Ad Hoc and Ubiquitous Computing*, *Computing and Informatics*, and the *Journal of High Speed Networks*.

• • •