Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 589243, 16 pages
doi:10.1155/2012/589243

Research Article

Energy-Efficient Multi-Job Scheduling Model for Cloud Computing and Its Genetic Algorithm

Xiaoli Wang,[1] Yuping Wang,[1] and Hai Zhu[2]

[1] *School of Computer Science and Technology, Xidian University, Xi'an, Shaanxi 710071, China*
[2] *School of Computer Science and Technology, Zhoukou Normal University, Zhoukou 466001, China*

Correspondence should be addressed to Xiaoli Wang, wangxiaolibox@gmail.com

Received 15 September 2011; Revised 19 October 2011; Accepted 19 October 2011

Academic Editor: Anders Eriksson

For the problem that the energy efficiency of the cloud computing data center is low, from the point of view of the energy efficiency of the servers, we propose a new energy-efficient multi-job scheduling model based on Google's massive data processing framework. To solve this model, we design a practical encoding and decoding method for the individuals and construct an overall energy efficiency function of the servers as the fitness value of each individual. Meanwhile, in order to accelerate the convergent speed of our algorithm and enhance its searching ability, a local search operator is introduced. Finally, the experiments show that the proposed algorithm is effective and efficient.

1. Introduction

Cloud computing [1] is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. As a new business model, while being favored by providing services such as on-demand self-service, broad network access, and rapid elasticity, cloud computing faces some new challenges. One of the prominent issues is the energy efficiency of data centers.

According to Amazon's CEMS project [2], based on a 3-year amortization schedule for servers and 15-year amortization schedule for other infrastructure, the monthly capital investment of the data center is illustrated in Figure 1. As can be seen from this figure, energy-related costs (including three parts: direct power consumption, power infrastructure, and cooling infrastructure) amount to 41.62% of the total. In other words, the largest investment to build data centers for cloud computing is not only to purchase thousands of server equipment, but also to buy the distribution and cooling infrastructure and to pay the bill

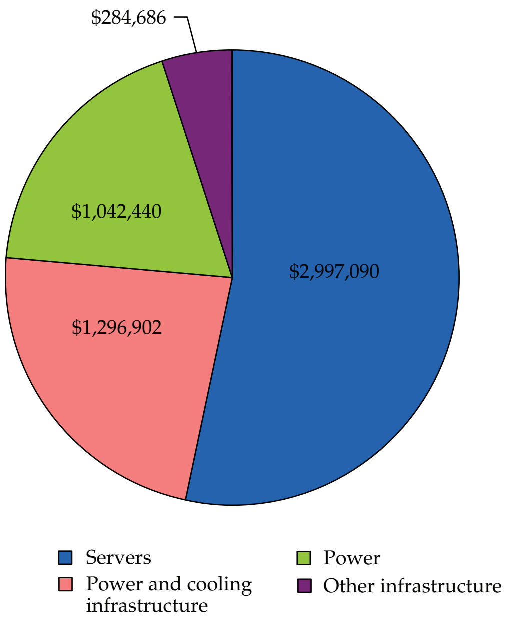


Figure 1: Monthly costs of the data center.

for energy consumption of all these facilities. In order to illustrate the importance of energy consumption for data centers, we introduce the concept, power usage effectiveness (PUE), which was developed by a consortium called The Green Grid.

Definition 1.1. Power usage effectiveness [3] is the ratio of total amount of power used by a data center facility to the power delivered to computing equipment. It is a measure of how efficiently a computer data center uses its power:

$$\text{PUE} = \frac{\text{Total facility power}}{\text{IT equipment power}}. \tag{1.1}$$

The IT equipment power is the power delivered to the critical load, the servers in the data center, while the total facility power in addition to the servers also includes other energy facilities, specifically, the energy consumed by distribution and cooling infrastructure which accounts for the main part. A PUE of 2.0 states that, for every watt delivered to the servers, we dissipate 1 watt in cooling system and power distribution. That is, the data center has to pay 2 watts of electricity, but only one single watt for cloud computing, because only servers can provide cloud computing services. The energy utilization is only 50%. In the Environmental Protection Agency (EPA)'s report [4] to the US Congress, they estimate that, in 2006, the typical enterprise data center had a PUE of 2.0 or higher. It is expected that equipment efficiency improvements alone, with current practices, could result in a 2011 PUE of 1.9. Data centers combining these efficiency gains with better operational practices are expected to reach a PUE of 1.3~1.7. Beyond that, the EPA predicted that "state-of-the-art" data centers could reach a PUE of 1.2. By now, Google has claimed that their data centers, on average for all, have exceeded the EPA's most optimistic scenario [5], which is of course accompanied by doubt voices from other cloud computing providers [2].

To reduce the energy consumption of data centers and improve energy efficiency, many scholars have done some related research, such as literatures [6–10]. Overall, we can make efforts in three aspects.

(1) Reduce power loss during distribution. However, the statistics from Amazon's CEMS project show that, for a data center with a PUE of 1.7, an overall power distribution loss only accounts for 8% of total energy consumption. Even with better technology, the reduction will not exceed 8% [2].

(2) Reduce energy consumed by cooling system. For example, you can use Google's "free cooling" mode, removing heat from servers by using evaporating water or low temperature ambient air. Google claims that there is no cooling equipment in its data centers in Belgium [11]. The climate in Belgium will support free cooling almost year-round, according to Google engineers, with temperatures rising above the acceptable range for free cooling about seven days per year on average. The average temperature in Brussels during summer reaches 66 to 71 degrees, while Google maintains its data centers at temperatures above 80 degrees. If the weather gets hot, Google says it will turn off equipment as needed in Belgium and shift computing load to other data centers. This approach is made possible by the scope of the company's global network of data centers, which provide the ability to shift an entire data center's workload to other facilities.

Although the "free cooling" mode can reduce the energy consumed by cooling system, it has a key prerequisite that the providers have sufficient enough financial and technical strength to run several data centers around the world and the data can backup across those data centers with seamless migration of computing load. This is hardly possible for majority of cloud computing providers.

(3) Improve energy efficiency of servers. Say a data center with a PUE of 2.0, only 50% of the power can be used on severs. Therefore, it becomes critical whether servers have used all the energy to complete the workload. We are aware that low energy utilization of a server is mainly due to its idle state caused by low CPU utilization. Even at a very low load, such as 10% CPU utilization, the power consumed is over 50% of the peak power [12]. Thus, the energy efficiency of servers plays an important role for the entire energy efficiency of the data center.

This paper mainly focuses on how to improve the energy efficiency of servers through appropriate scheduling strategies. Taking full consideration of the relationship between the performance and energy consumption of servers, we propose a new energy-efficient multi-job scheduling model based on the Google's massive data processing framework, MapReduce, and give its corresponding algorithm. As the basics of our model, Section 2 highlights Google's MapReduce framework; Section 3 gives the mathematical description of the energy-efficient multi-job scheduling problem and its corresponding model. In order to solve this model, a genetic algorithm and its genetic operators are designed in Section 4. Finally, simulation experiments show the proposed algorithm is effective and efficient in Section 5.

2. MapReduce Framework

MapReduce [13] is Google's massive data processing framework. It finishes the computation by mapping and reducing data under cluster environment. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world jobs are expressible in this model, for example, counting the frequency of all words

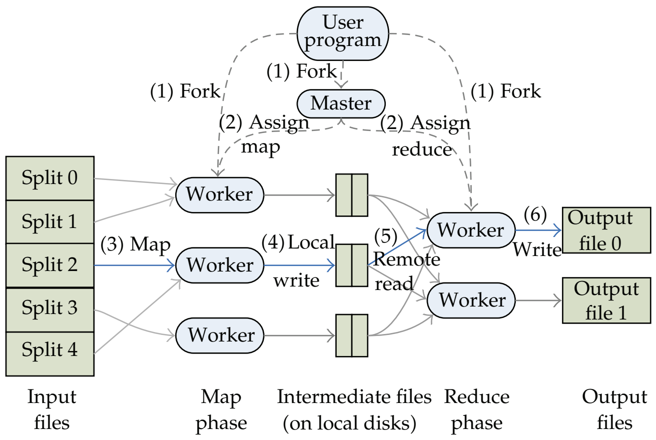


Figure 2: Overall flow of a MapReduce operation.

appeared in a paper. Figure 2 shows the overall flow of a MapReduce operation. When the user program calls the MapReduce function, the following sequence of actions occurs (the numbered labels in Figure 2 correspond to the numbers in the list below).

Step 1. The MapReduce library in the user program first splits the input files into M pieces of typically 64 megabytes (MBs) per piece. It then starts up many copies of the program on a cluster of machines.

Step 2. One of the copies of the program is special—the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.

Step 3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined *map* function. The intermediate key/value pairs produced by the *map* function are buffered in memory.

Step 4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

Step 5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers.

Step 6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered; it passes the key and the corresponding set of intermediate values to the user's reduce function. The output of the reduce function is appended to a final output file for this reduce partition.

Step 7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

3. Energy-Efficient Multi-Job Scheduling Model Based on MapReduce

From the above background knowledge, we know that by improving the energy efficiency of servers, the PUE of data centers can be enhanced. However, this problem cannot be solved as easy as balancing loads among servers so as to make all the servers' CPU utilization reach 100%. Instead, there exists an optimal performance and energy point for each server [12]. Energy consumption per task is influenced by the CPU utilization of servers (certainly, it may also be affected by other resource utilization such as memory, bandwidth, etc., but, in order to simplify the model, we only consider the impact of CPU utilization). When the CPU utilization is low, idle power is not amortized effectively and hence the energy per task is high. At high CPU utilization, on the other hand, energy consumption is high due to the competition for resources among tasks, which leads to performance degradation and longer execution time. Typical variation of energy per task with CPU utilization can be expected to result in a "U-shaped" curve. Therefore, it can be assumed that the servers achieve the maximum energy efficiency when all servers running at its optimal performance and power point. We can get this optimal point of each server by experiments. In literature [12], for example, data shows that the server meets the highest when its CPU utilization reaches 70%.

We first give a general mathematical description of the energy-efficient multi-job scheduling problem and then build its corresponding single-objective optimization model.

Assuming that there are N servers in a data center. The current CPU utilization of sever k is CS_k and its optimal point is CO_k, where $k = 1, 2, \ldots N$. There are F jobs $A = \{A_1, A_2, \ldots, A_F\}$ need to be processed, and the input data of job A_q is D_q, where $q = 1, 2, \ldots, F$. The input data D_q will be divided into m_q splits with each size of 64 M, so there are $\overline{m} = \sum_{q=1}^{F} m_q$ splits, which are denoted as $D = \{D_1, D_2, \ldots, D_F\} = \{d_1, d_2, \ldots, d_{\overline{m}}\}$. First, we need to randomly store these $\overline{m}$ splits on N servers. To ensure the reliability of data, each split will choose three different servers for storage. We use a $\overline{m} \times 3$ matrix P to represent the storage location of every split, and the element p_{ij} indicates the storage location of split d_i, where integer $p_{ij} \in [1, N]$, $i = 1, 2, \ldots, \overline{m}$ and $j = 1, 2, 3$. From the MapReduce framework, we know that each input data D_q will be processed by m_q map tasks and r_q reduce tasks, provided that the CPU required for every map task of job A_q is CM_q and, for every reduce task, is CR_q. The problem is how to assign these $v = \sum_{q=1}^{F} m_q + \sum_{q=1}^{F} r_q$ tasks on N servers, so that the energy efficiency of all servers reaches the highest.

We use vector $S = (s_1, s_2, \ldots s_i, \ldots, s_v)$ to represent the final task scheduling scheme, and the ith element of vector S indicates that task i is assigned on server s_i, where $1 \le s_i \le N$ and $i = 1, 2, \ldots, v$. Traversing through the scheduling scheme S, we can get the set of all map tasks and reduce tasks of job A_q which are assigned on server k, denoted as M_k^q and R_k^q, respectively, where $k = 1, 2, \ldots N$ and $q = 1, 2, \ldots, F$. Let $NM_k^q = |M_k^q|$ and $NR_k^q = |R_k^q|$. Here we give the single-objective optimization model for the energy-efficient multi-job scheduling problem based on MapReduce for cloud computing:

$$\min \quad \sum_{k=1}^{N} \left(CO_k - \left(CS_k + \sum_{q=1}^{F} \left(NM_k^q \times CM_q \right) + \sum_{q=1}^{F} \left(NR_k^q \times CR_q \right) \right) \right)^2 \tag{3.1}$$

$$\text{s.t.} \quad \text{for scheduling scheme } S, \begin{cases} s_i \in \{p_{i1}, p_{i2}, p_{i3}\}, & \text{for } i = 1, 2, \ldots, \overline{m}, \\ s_i \in [1, N], & \text{for } i = \overline{m} + 1, \overline{m} + 2, \ldots, v, \end{cases} \tag{3.2}$$

$$NM_k^q = \left| \left\{ s_i \mid s_i = k,\ i = \sum_{j=0}^{q-1} m_j + 1,\ \sum_{j=0}^{q-1} m_j + 2, \ldots,\ \sum_{j=0}^{q-1} m_j + m_q \right\} \right|, \quad \text{where } m_0 = 0, \quad k = 1, 2, \ldots N, \quad q = 1, 2, \ldots, F, \tag{3.3}$$

$$NR_k^q = \left| \left\{ s_i \mid s_i = k,\ i = \overline{m} + \sum_{j=0}^{q-1} r_j + 1,\ \overline{m} + \sum_{j=0}^{q-1} r_j + 2, \ldots,\ \overline{m} + \sum_{j=0}^{q-1} r_j + r_q \right\} \right|, \quad \text{where } r_0 = 0, \quad k = 1, 2, \ldots N, \quad q = 1, 2, \ldots, F, \tag{3.4}$$

$$CS_k + \sum_{q=1}^{F} \left(NM_k^q \times CM_q \right) + \sum_{q=1}^{F} \left(NR_k^q \times CR_q \right) \le 1, \quad \text{where } k = 1, 2, \ldots N, \quad CM_q \in [0, 1], \quad CR_q \in [0, 1], \quad q = 1, 2, \ldots, F. \tag{3.5}$$

The objective function indicates the minimum sum of the difference between all servers' CPU utilizations after scheduling and their optimal points. Constraint (3.2) expresses that if a map task i is assigned to server s_i, then server s_i must have stored the corresponding input data of this map task. This is because the MapReduce framework is mainly used in massive data processing and the network bandwidth is a relatively scarce resource in cloud computing environment. MapReduce prefers moving the executive program to the node which stores the data, rather than moving the data as in traditional distributed computing. This scheduling scheme based on data location can avoid a large-scale data movement, which not only reduces the network overhead, but also makes the map tasks locally read and process the data. Constraints (3.3) and (3.4) compute the number of map tasks NM_k^q and reduce tasks NR_k^q of job A_q which are assigned to server k. Constraints (3.5) indicates that the CPU utilization of any server should not exceed 100% before and after the task scheduling.

4. An Energy-Efficient Multi-Job Scheduling Algorithm Based on MapReduce

Task scheduling is an NP problem, and the genetic algorithm based on evolutionary theory is very suitable for complex optimization problems. Here, we give the energy-efficient multi-job scheduling algorithm in detail, including the encoding, decoding methods for individuals, and other genetic operators. In particular, to make the new generated individuals meet the constraints, a modified operator is designed in the proposed algorithm. Meanwhile, in order to accelerate the convergence of the proposed algorithm, a local search operator is introduced. Finally, the overall genetic algorithm flow will be given in this section.

4.1. Encoding, Decoding, and Initializing Population

In genetic algorithm, the encoding method is of great significance. Based on the characteristics of this energy-efficient multi-job scheduling problem, we adopt the integer coding, provided that there are $v = \sum_{q=1}^{F} m_q + \sum_{q=1}^{F} r_q$ tasks need to be processed, including $\overline{m} = \sum_{q=1}^{F} m_q$ map tasks and $\overline{r} = \sum_{q=1}^{F} r_q$ reduce tasks. We use vector $S = (s_1, s_2, \ldots, s_v)$ as an individual to represent a scheduling scheme, where the ith element indicates task i is assigned on server s_i. This encoding method has the advantage that we can use relatively simple multipoint crossover operator for the evolution of individuals.

When initializing individual $S = (s_1, s_2, \ldots, s_v)$, for map tasks $i = 1, 2, \ldots \overline{m}$, it is necessary that they have to be assigned on servers which store their corresponding input data. Take a random integer $j \in [1, 3]$, let $s_i = p_{ij}$; for reduce tasks $i = \overline{m}+1, \overline{m}+2, \ldots, v$, there is no such requirement, so just take a random integer $k \in [1, N]$, let $s_i = k$.

To compute individuals' fitness value, we need to decode it first. The individual decoding method is as follows.

Algorithm 4.1.

Step 1. Initializing NM_k^q and NR_k^q, let $NM_k^q = 0$ and $NR_k^q = 0$, where $k = 1, 2, \ldots N$ and $q = 1, 2, \ldots q$. Empty set M_k and R_k.

Step 2. For each element s_i of individual S, set $k = s_i$ and $m_0 = 0$. For job A_q, where $q = 1, 2, \ldots, F$, if $i = \sum_{j=0}^{q-1} m_j + 1,\ \sum_{j=0}^{q-1} m_j + 2, \ldots, \sum_{j=0}^{q-1} m_j + m_q$, NM_k^q plus 1 and put i into set M_k; else if $i = \overline{m} + \sum_{j=0}^{q-1} r_j + 1,\ \overline{m} + \sum_{j=0}^{q-1} r_j + 2, \ldots, \overline{m} + \sum_{j=0}^{q-1} r_j + r_q$, NR_k^q plus 1 and put i into set R_k.

4.2. Modified Operator

As the CPU utilization of each server cannot exceed 100% after task scheduling and whether an individual is generated by population initialization or by genetic operators like crossover and mutation, constraints cannot be guaranteed. So the new generated individuals may need to be modified. Based on different status for each server, we will remove its corresponding excess map tasks and reduce tasks allocated on it. The following shows the specific steps for the modified operator.

Algorithm 4.2.

Step 1. Say individual S need to be modified. Decode individual S according to Algorithm 4.1.

Step 2. Let $k = 1$.

Step 3. If $k > N$, stop; else if $CS_k + \sum_{q=1}^{F}(NM_k^q \times CM_q) + \sum_{q=1}^{F}(NR_k^q \times CR_q) > 1$, go to Step 4; otherwise, let $k = k + 1$ go to Step 3.

Step 4. Let $d = CO_k - CS_k$. If $d \leq 0$, go to Step 5; otherwise, go to Step 6.

Step 5. Equation $d = 0$ indicates that server k has been the best state before task scheduling, so there is no need to assign more tasks on it, while $d < 0$ means that the CPU utilization of server k has already been higher than its optimal point before task scheduling, so the best choice is not to assign more tasks on it. Therefore, in both cases, all tasks allocated on server k should be deleted. Let cut $= \sum_{q=1}^{F}(NM_k^q \times CM_q) + \sum_{q=1}^{F}(NR_k^q \times CR_q)$, go to Step 7.

Step 6. If $d > 0$, then server k has not been the best state before scheduling, but its CPU utilization exceeds 100% after the scheduling, so we need to remove the excess part of its assignment. Let cut $= CS_k + \sum_{q=1}^{F}(NM_k^q \times CM_q) + \sum_{q=1}^{F}(NR_k^q \times CR_q) - CO_k$. In order to avoid the situation that is always deleting tasks with smaller numbers, which is caused by the fixed sequence of sets M_k and N_k, we randomly disrupt task orders in them.

Step 7. Remove excess map tasks. For $x = 1, 2, \ldots, NM_k$, take the xth map task i from set M_k. There exists an integer $s \in [1, F]$ which satisfies $\sum_{q=1}^{s} m_q \leq i \leq \sum_{q=1}^{s+1} m_q$. The value of s indicates that this map task belongs to job A_s thus, its CPU requirement is CM_s. If $\text{cut} - CM_s < 0$, then go to Step 8; otherwise, reassign task i to a new server w which satisfies $w \neq k$ and $w \in \{p_{i1}, p_{i2}, p_{i3}\}$. Set $s_i = w$. Continue the next iteration, let $x = x + 1$ and $\text{cut} = \text{cut} - CM_s$.

Step 8. Remove excess reduce tasks. For $x = 1, 2, \ldots NR_k$, take the xth reduce task i from set R_k. There exists an integer $s \in [1, F]$ which satisfies $\overline{m} + \sum_{q=1}^{s} r_q \leq i \leq \overline{m} + \sum_{q=1}^{s+1} r_q$. The value of s indicates that this reduce task belongs to job A_s, and its CPU requirement is CR_s. If $\text{cut} - CR_s < 0$, then go to Step 1; otherwise, reassign this task to a new server w which satisfies $w \in [1, N]$ and $w \neq k$. Set $s_i = w$. Start the next iteration, let $x = x + 1$ and $\text{cut} = \text{cut} - CR_s$.

4.3. Crossover Operator

We adopt the multipoint crossover operator for the evolution of individuals. To make individuals meet the constraints of our model, the new generated individuals may need to be modified. Meanwhile, in order to speed up the convergence of the proposed algorithm, we conduct a local search for the new generated individuals. Take two jobs $F = 2$ as an example, and the crossover process is as follows.

Algorithm 4.3.

Step 1. Say the crossover probability is pc. Empty the crossover pool set pl. For each individual in the population, generate a real number $q \in [0, 1]$. If $q \leq pc$, then put this individual into pl.

Step 2. If pl is empty or there is only one individual in it, stop; otherwise, select two individuals S^1 and S^2 from pl without replacement. Generate four random integers $c1 \in [1, m_1]$, $c2 \in [m_1 + 1, \overline{m}]$, $c3 \in [\overline{m} + 1, \overline{m} + r_1]$, and $c4 \in [\overline{m} + r_1 + 1, v]$ as the crossover points.

Step 3. For individuals S^1 and S^2, generate new individuals S^3 and S^4 as follows:

$$\begin{aligned}
S^1 &= \left(s_1^1, \ldots, s_{c1}^1, \ \vdots\ s_{c1+1}^1, \ldots, s_{c2}^1, \ \vdots\ s_{c2+1}^1, \ldots, s_{c3}^1, \ \vdots\ s_{c3+1}^1, \ldots, s_{c4}^1, \ \vdots\ s_{c4+1}^1, \ldots, s_v^1\right) \\
S^2 &= \left(s_1^2, \ldots, s_{c1}^2, \ \vdots\ s_{c1+1}^2, \ldots, s_{c2}^2, \ \vdots\ s_{c2+1}^2, \ldots, s_{c3}^2, \ \vdots\ s_{c3+1}^2, \ldots, s_{c4}^2, \ \vdots\ s_{c4+1}^2, \ldots, s_v^2\right) \\
& \qquad \Downarrow \qquad\qquad \Downarrow \qquad\qquad \Downarrow \qquad\qquad \Downarrow \qquad\qquad \Downarrow \\
S^3 &= \left(s_1^1, \ldots, s_{c1}^1, \ \vdots\ s_{c1+1}^2, \ldots, s_{c2}^2, \ \vdots\ s_{c2+1}^1, \ldots, s_{c3}^1, \ \vdots\ s_{c3+1}^2, \cdots, s_{c4}^2, \ \vdots\ s_{c4+1}^1, \ldots, s_v^1\right) \\
S^4 &= \left(s_1^2, \ldots, s_{c1}^2, \ \vdots\ s_{c1+1}^1, \ldots, s_{c2}^1, \ \vdots\ s_{c2+1}^2, \ldots, s_{c3}^2, \ \vdots\ s_{c3+1}^1, \ldots, s_{c4}^1, \ \vdots\ s_{c4+1}^2, \ldots, s_v^2\right).
\end{aligned} \tag{4.1}$$

Step 4. Modify individuals S^3 and S^4 according to Algorithm 4.2.

Step 5. Locally search individuals S^3 and S^4 according to Algorithm 4.5. Go to Step 2.

4.4. Mutation Operator

We use single-point mutation operator for the evolution of individuals. To make individuals meet the constraints of our model, the new generated individuals may need to be modified.

Meanwhile, in order to speed up the convergence of our algorithm, we conduct a local search for the new generated individuals. The mutation process is as follows.

Algorithm 4.4.

Step 1. Select individuals from the offspring of crossover according to the mutation probability pm.

Step 2. For each selected offspring, say $S = (s_1, s_2, \ldots s_i, \ldots, s_v)$, randomly generate an integer $i \in [1, v]$. If $i \leq \overline{m}$, then task i is a map task, reassign this task to a new server w which satisfies $w \neq s_i$ and $w \in \{p_{i1}, p_{i2}, p_{i3}\}$. Set $s_i = w$; otherwise, task i is a reduce task. Randomly generate an integer $k \in [1, N]$ that satisfies $k \neq s_i$. Set $s_i = k$. The new generated individual is denoted as S'.

Step 3. Modify individual S' according to Algorithm 4.2.

Step 4. Locally search individual S' according to Algorithm 4.5.

4.5. Local Search Operator

In order to accelerate the convergent speed and enhance the searching ability of the proposed algorithm, a local search operator is designed in this paper. We know that after scheduling the smaller the difference between the CPU utilization and the optimal point, the better the individual. Thus, in each iteration, we select the server with the maximum difference between the CPU utilization after scheduling and its optimal point, then reassign tasks on it. If the new generated individual is better than the current one, update the current individual and continue to the next iteration; otherwise, stop the local search.

Algorithm 4.5.

Step 1. Say the fitness value of individual S is f. Copy this individual. Let $S' = S$.

Step 2. Decode individual S' according to Algorithm 4.1.

Step 3. Among all the servers, there exists a server k with the highest CPU utilization, and at least one task is assigned on it. Let $d = CO_k - CS_k$. If $d < 0$, then the initial CPU utilization of server k has been higher than its optimal point before scheduling, so all tasks allocated on server k should be deleted. Let cut $= \sum_{q=1}^{F}(NM_k^q \times CM_q) + \sum_{q=1}^{F}(NR_k^q \times CR_q)$, go to Step 4; otherwise, let cut $= CS_k + \sum_{q=1}^{F}(NM_k^q \times CM_q) + \sum_{q=1}^{F}(NR_k^q \times CR_q) - CO_k$. Randomly disrupt task orders in sets M_k and N_k.

Step 4. Remove excess map tasks. For $x = 1, 2, \ldots, NM_k$, take the xth map task i from set M_k. There exists an integer $s \in [1, F]$ which satisfies $\sum_{q=1}^{s} m_q \leq i \leq \sum_{q=1}^{s+1} m_q$. If cut $- CM_s < 0$, then go to Step 5; otherwise, reassign task i on a new server w which satisfies $w \neq k$ and $w \in \{p_{i1}, p_{i2}, p_{i3}\}$. Let $s_i = w$ and cut $=$ cut $- CM_s$.

Step 5. Remove excess reduce tasks. For $x = 1, 2, \ldots, NR_k$, take the xth reduce task i from set R_k. There exists an integer $s \in [1, F]$ which satisfies $\overline{m} + \sum_{q=1}^{s} r_q \leq i \leq \overline{m} + \sum_{q=1}^{s+1} r_q$. If cut $- CR_s < 0$, compute the fitness value of individual S' denoted as f'; otherwise, reassign a server w which satisfies $w \in [1, N]$ and $w \neq k$ for task i. Let $s_i = w$ and cut $=$ cut $- CR_s$.

Step 6. If $f' < f$, then S' is better than S, let $S = S'$, and go to Step 2; otherwise, recopy individual S, let $S' = S$. Decode individual S' according to Algorithm 4.1.

Step 7. Among all the servers, there exists a server k with the lowest CPU utilization.
Let add $= CO_k - CS_k + \sum_{q=1}^{F}(NM_k^q \times CM_q) + \sum_{q=1}^{F}(NR_k^q \times CR_q)$.

Step 8. Add map tasks. According to the storage location of each data split, we can get all tasks which can be assigned on server k, denoted as set MM_k. Randomly disrupt task orders in set MM_k.

Step 9. There exists a map task $p \in MM_k$ and $s'_p \neq k$. For this task, there exists an integer $s \in [1, F]$ which satisfies $\sum_{q=1}^{s} m_q \leq p \leq \sum_{q=1}^{s+1} m_q$. If add $- CM_p < 0$, go to Step 10; otherwise, let $s'_p = k$ and add = add $- CM_p$, go to Step 9.

Step 10. Add reduce tasks: There exists task $p \in [\overline{m}+1, v]$ and $s'_p \neq k$. For this task, there exists an integer $s \in [1, F]$ which satisfies $\overline{m} + \sum_{q=1}^{s} r_q \leq p \leq \overline{m} + \sum_{q=1}^{s+1} r_q$. If add $- CR_p < 0$, compute the fitness value of individual S' denoted as f'; otherwise, let $s'_p = k$ and add = add $- CR_p$, go to Step 10.

Step 11. If $f' < f$, then S' is better than S. Let $S = S'$, go to Step 7; otherwise, stop.

4.6. An Energy-Efficient Multi-Job Scheduling Algorithm Based on MapReduce

Algorithm 4.6.

Step 1. Initializing. Choose proper genetic parameters: population size X, crossover probability pc, mutation probability pm, and elitist number k. Generate an initial population P. Modify each individual in population P according to Algorithm 4.2, and compute its fitness values. Set generation number $t = 0$.

Step 2. Crossover. Execute crossover by Algorithm 4.3. The offspring set is denoted as $P1$ and compute each individual's fitness value.

Step 3. Mutation. Execute mutation on $P1$ by Algorithm 4.4. The offspring set is denoted as $P2$, and compute each individual's fitness value.

Step 4. Elitist strategy. Sort the individuals in set $P \cup P1 \cup P2$ according to its fitness value, and select the best k individuals directly to form the next generation population, while the others are selected by using roulette wheel method on the set $P \cup P1 \cup P2$.

Step 5. If stopping criterion is not met, let $t = t + 1$, go to Step 2; otherwise, stop.

5. Experiments and Analysis

5.1. Parameter Values

Given that there are 200 servers in a data center and 2 jobs need to be processed, as $N = 200$ and $F = 2$, the data sizes of the jobs are 500 G and 750 G, respectively, and they can be divided into 8000 splits and 12000 splits, which means $m_1 = 8000$ and $m_2 = 12000$. Each split randomly selects three servers to back up its data. Suppose that the number of reduce tasks required for completing the two jobs are 180 and 270, respectively, which means $r_1 = 180$ and $r_2 = 270$.

Based on a 3-year amortization schedule for servers in a data center, different server may have different optimal performance-energy point for how long it has been used. Here we assume that 1/3 servers have been used for one year with their optimal point of 0.9. Other 1/3 servers have been used for two years with their optimal point of 0.7, while the others have been used for three years with their optimal point of 0.5. Take random real numbers over $[0, 0.35]$ as servers' initial CPU utilization value. To reflect the effectiveness of the proposed algorithm well, we set some special initial states of servers as follows:

$$\begin{aligned} CS_5 &= 0.5, & CS_{25} &= 0.7, & CS_{45} &= 0.9, \\ CS_{75} &= 0.5, & CS_{95} &= 0.7, & CS_{115} &= 0.9, \\ CS_{145} &= 0.5, & CS_{165} &= 0.7, & CS_{195} &= 0.9. \end{aligned} \tag{5.1}$$

In the proposed energy-efficient multi-job scheduling algorithm, we adopt the following genetic parameters: population size $X = 100$; crossover probability $pc = 0.6$; mutation probability $pm = 0.02$; elitist number $k = 5$; stop criterion $t = 2000$.

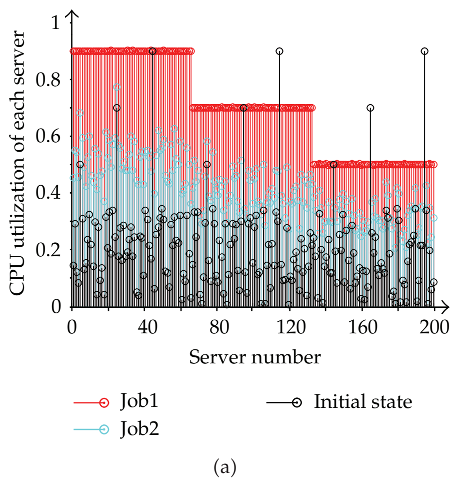
5.2. Simulation Results and Comparions

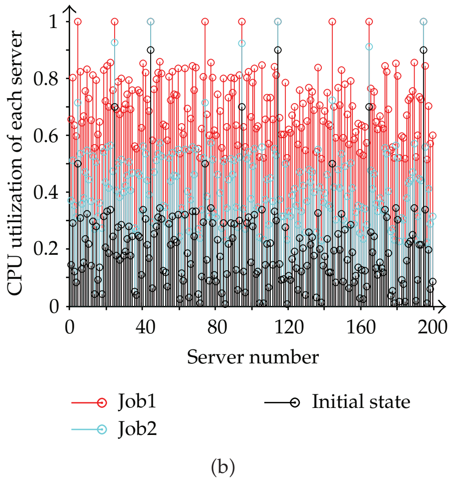
We conduct three sets of comparative experiments between the proposed algorithm and Hadoop MapReduce which is an open-source MapReduce framework implementation [14].

Comparison 1

Set the CPU requirements for each map task of the two jobs as $CM_1 = 0.0055$ and $CM_2 = 0.0046$, respectively, and the CPU requirements for each reduce tasks as $CR_1 = 0.0017$ and $CR = 0.0022$, respectively. The experimental results of the energy-efficient multi-job scheduling algorithm proposed in this paper are shown in Figure 3(a), while the results of the Hadoop MapReduce scheduling are shown in Figure 3(b).

It can be clearly seen by comparing Figure 3(a) with Figure 3(b) that the proposed algorithm in this paper can effectively schedule multi-job on servers according to each server's optimal performance-energy point. For the 5th, 75th, and 145th servers with the same initial CPU utilization of 0.5, since the optimal points of these three servers are 0.9, 0.7, and 0.5, the proposed algorithm only assigns tasks on the 5th and 75th servers, while the 145th server stays at its original state. Similarly, for the 25th, 95th, and 165th servers with the same initial CPU utilization of 0.7, the proposed algorithm only assigns tasks on the 25th server. Also, for the 45th, 115th, and 195th servers with the same initial CPU utilization of 0.9, the proposed algorithm does not assign any tasks on them. From another perspective, we check the total amount of tasks assigned on each server. For the proposed algorithm, it can be seen from Figure 3(c) that the higher the server's optimal performance-energy point, the more tasks it needs to deal with, expect for those servers with high initial CUP utilization, while, for the Hadoop MapReduce scheduling, if not taken server's initial state into consideration, tasks are assigned nearly equally. Furthermore, computing the energy efficiency of all servers by the proposed algorithm and Hadoop MapReduce scheduling according to the objective function in our model, we will get the results of 0.240227 and 6.79271, respectively, which means that the proposed algorithm can greatly improve the energy efficiency of servers, so as to enhance the PUE of data centers.





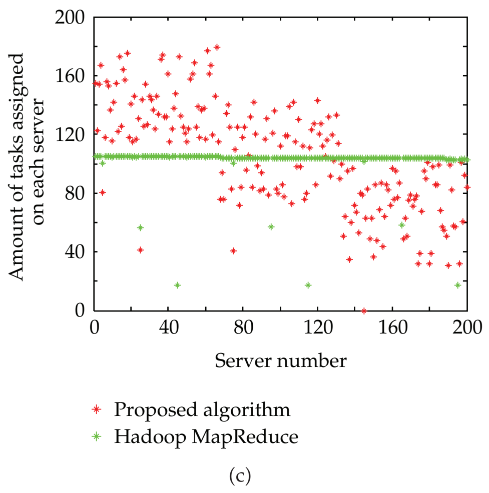
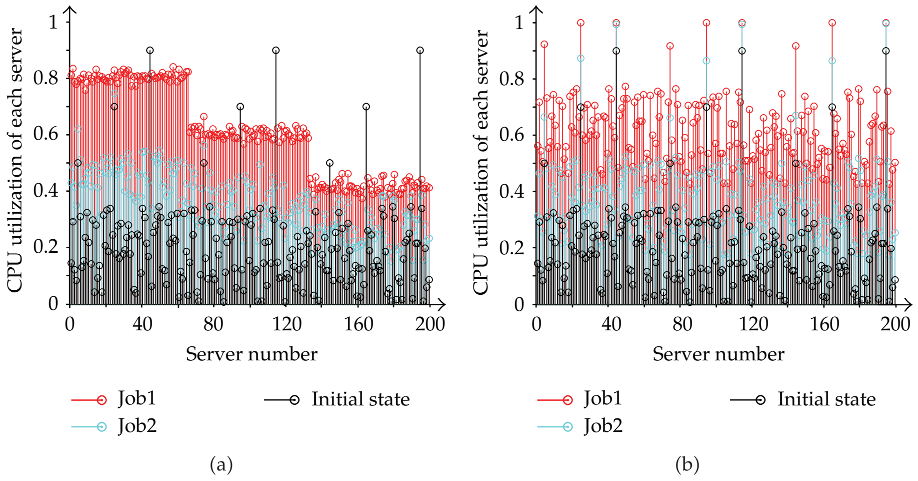


Figure 3: (a) The results of the energy-efficient multi-job scheduling algorithm in experiment 1. (b) The results of the Hadoop MapReduce scheduling algorithm in experiment 1. (c) The total amount of tasks assigned on each server in experiment 1.

Comparison 2

Suppose that the input data to be processed is relatively small. Set the CPU requirements for each map task of the two jobs as $CM_1 = 0.0042$ and $CM_2 = 0.004$, respectively, and the CPU requirements for each reduce tasks as $CR_1 = 0.0015$ and $CR = 0.002$, respectively. The experimental results of the energy-efficient multi-job scheduling algorithm proposed in this paper are shown in Figure 4(a), while the results of the Hadoop MapReduce scheduling are shown in Figure 4(b).

From Figure 4(a), it can be seen that even when the input data to be processed is relatively small, the proposed algorithm can effectively schedule multi-job on servers according to each server’s optimal performance-energy point. Although the CPU utilizations of all servers are not able to reach their optimal points after the scheduling, each server’s CPU utilization is near as much as possible to its optimal point. Similar to comparison 1, we check the total amount of tasks assigned on each server. For the proposed algorithm, it can be seen



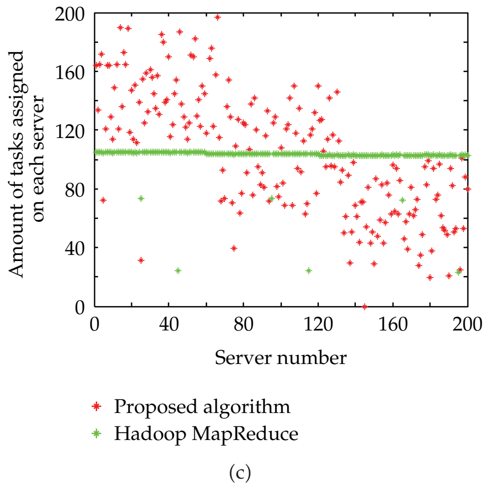
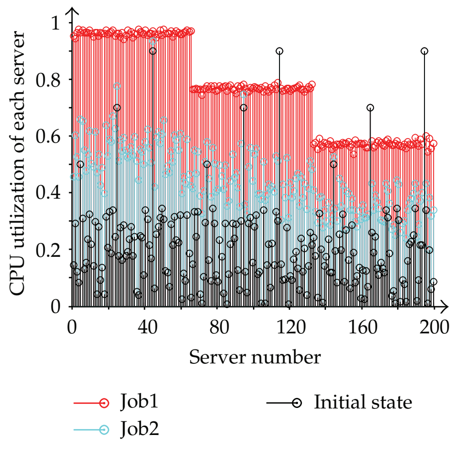


Figure 4: (a) The results of the energy-efficient multi-job scheduling algorithm in experiment 2. (b) The results of the Hadoop MapReduce scheduling algorithm in experiment 2. (c) The total amount of tasks assigned on each server in experiment 2.

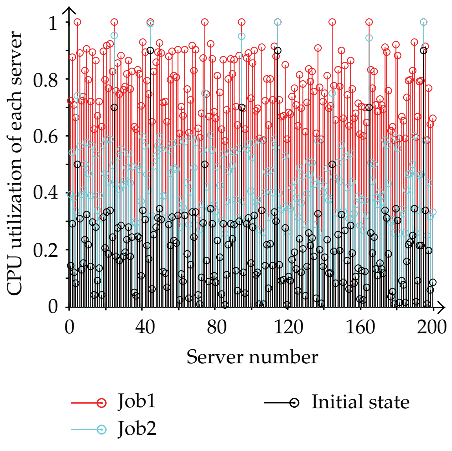
from Figure 4(c) that the higher the server's optimal performance-energy point, the more tasks it needs to deal with, expect for those servers with high initial CUP utilization, while, for the Hadoop MapReduce scheduling, if not taken server's initial state into consideration, tasks are assigned nearly balancing. Furthermore, computing the energy efficiency of all servers by the proposed algorithm and Hadoop MapReduce scheduling according to the objective function in our model, we will get the results of 1.12455 and 8.61073, respectively, which proves that the proposed algorithm can greatly improve the energy efficiency of servers, so as to enhance the PUE of data centers.

Comparison 3

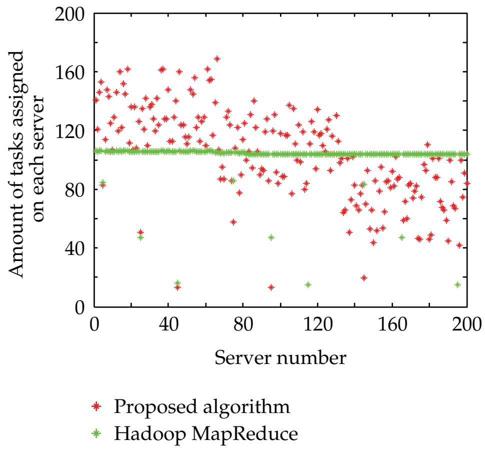
Suppose that the input data to be processed is relatively large. Set the CPU requirements for each map task of the two jobs as $CM_1 = 0.0065$ and $CM_2 = 0.0054$, respectively, and the CPU requirements for each reduce tasks as $CR_1 = 0.002$ and $CR = 0.003$, respectively. The experimental results of the energy-efficient multi-job scheduling algorithm proposed in this



(a)



(b)



(c)

Figure 5: (a) The results of the energy-efficient multi-job scheduling algorithm in experiment 3. (b) The results of the Hadoop MapReduce scheduling algorithm in experiment 3. (c) The total amount of tasks assigned on each server in experiment 3.

paper are shown in Figure 5(a), while the results of the Hadoop MapReduce scheduling are shown in Figure 5(b).

From Figure 5(a), it can be seen that even when the input data to be processed is relatively large, the proposed algorithm in this paper can effectively schedule multi-job on servers according to each server's optimal performance-energy point. Although the CPU utilizations of all servers are beyond their optimal points after the scheduling, each server's CUP utilization is near as much as possible to its optimal point. From another perspective, we check the total amount of tasks assigned on each server. For the proposed algorithm, it can be seen from Figure 5(c) that the higher the server's optimal performance-energy point, the more tasks it needs to deal with, expect for those servers with high initial CUP utilization, while, for the Hadoop MapReduce scheduling, if not taken server's initial state into consideration, tasks are assigned nearly balancing. Furthermore, computing the energy efficiency of all servers by the proposed algorithm and Hadoop MapReduce scheduling according to the objective function in our model, we will get the results of 1.93834 and

7.37484, respectively, which proves that the proposed algorithm can greatly improve the energy efficiency of servers, so as to enhance the PUE of data centers.

6. Conclusion

The energy efficiency of servers plays a significant role in the overall energy consumption of the data center. This paper mainly focuses on how to improve the energy efficiency of servers through appropriate scheduling strategies. Taking full consideration of the relationship between the performance and energy consumption of servers, we propose a new energy-efficient multi-job scheduling model based on the Google's massive data processing framework, MapReduce, and give the corresponding algorithm. Meanwhile, we design a practical encoding and decoding method for the individuals and construct an overall energy efficiency function of the servers as the fitness value of the individual. Also, in order to accelerate the convergent speed and enhance the searching ability of our algorithm, a local search operator is introduced. Finally, the experiments show that the proposed algorithm is effective and efficient.

Acknowledgments

This work was supported by National Natural Science Foundation of China (no. 60873099, no. 61103143) and the PhD Programs Foundation of Education Ministry of China (no. 20090203110005).

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, 2009.
[2] J. Hamilton, "Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services," Citeseer.
[3] C. Belady, "The green grid data center power efficiency metrics: PUE and DCiE," White paper: Metrics & Measurements, 2007.
[4] ENERGY STAR, "Report to congress on server and data center energy efficiency public law 109–431," Public law,109:431, 2007.
[5] "Efficiency measurements [EB/OL]," http://www.google.com/corporate/datacenter/efficiency-measurements.html.
[6] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '10)*, pp. 577–578, Melbourne, Australia, May 2010.
[7] A. Berl, E. Gelenbe, M. Di Girolamo et al., "Energy-efficient cloud computing," *Computer Journal*, vol. 53, no. 7, pp. 1045–1051, 2010.
[8] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-Efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," *Distributed, Parallel, and Cluster Computing, http://arxiv.org/abs/1006.0308/*, pp. 6–17, 2010.
[9] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: balancing energy in processing, storage and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.
[10] L. A. Barroso and U. Hölzle, "The datacenter as a computer: an introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.
[11] "Google's chiller-less data center," 2009, http://www.datacenterknowledge.com/.
[12] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the Conference on Power aware computing and systems*, p. 10, USENIX Association, San Diego, Calif, USA, 2008.

[13] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[14] W. T. Hadoop, *The Definitive Guide*, O’Reilly Media, Sebastopol, Calif, USA, 2009.