Research of Dependent Tasks Scheduling Algorithm in Cloud Computing Environments

Qing-Yi CHEN[1], Zhi-Hong LIANG[2,*], Hong-Wei KANG[3], Yu-Ming MA[4] and Dong WANG

[1,2,3,4]*School of Software, Yunnan University, Kunming, Yunnan, 650091, China*
E-mail: zhliang@ynu.edu.cn

Abstract: With the dependent relationship of tasks submitted by the users in the model of Cloud computing resources scheduling become stronger and stronger, it is worthy of studying how to optimize the scheduling strategy and algorithm to meet the different demands of the users, and it is absolutely importance. In this article, the author analysed the factors that will affect the entire task-sets execution firstly. Then proposed a new tasks scheduling model based on the original priority calculation method and the idea of redundant duplication of tasks. In the phase of tasks scheduling in the model, the execution results of all parent tasks of the subtask that being executing are considered. The costs of communication between task-sets has reduced by the method of redundant duplication of tasks, so that the execution time of some subtasks share be advanced, and the entire execution efficiency of task-sets can be increased. At the end of this article, from the comparative results of the space-time complexity of contrast algorithms and the algorithm proposed by the author during the process of processing dependent tasks, we can find that subtasks execution time can be advanced and the complete time of the whole task-set can be cut down to a certain extent

1 Dependent Tasks and the Resource Node Modelling

The dependent tasks are the tasks that in the task-sets submitted by users exist a certain dependent relationship between each other, and the relationship shows up in that before the subtask be executed, the execution results of its parent tasks must be known. And we can use a directed acyclic graph G to represent the dependent tasks [1]. As is shown in the following figure:

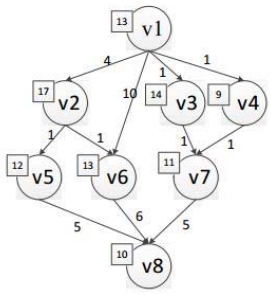

Figure 1. Sketch of Dependent Task

The figure is represented as a four-tuples: $G =< N,\ E,\ Q,\ W >$ in which $N = \{v_1, v_2, \dots,\ v_n\}$, $E = \{e_{ij} | i, j \in n,\ i \neq j\}$, $Q = \{q_1, q_2, \dots,\ q_m\}$ and $W = \{w_{ij} | i, j \in m,\ i \neq j\}$. The N represents the task node collection and the v_i represents a single task node. The E represents the collection of "edge", and the edge determines the dependent relationship between tasks. If tow task nodes are connected by the edge, then there is a dependency relationship between them, and the sub task is the task node that the arrow points to, another side points to the parent task. The Q represents the average calculation cost collection for each task node, and the average calculation cost here refers to the average value of the cost of the task node in all resource nodes. The W represents the collection of communication cost between the tasks. And the special note here is that if tow dependencies are distributed to the same computing node, there is no communication cost between them [2].

We can also make the following definition:

Definition 1: $\mathrm{pre}(\mathrm{v_i})$ represents the predecessor task node set (or parent task node set) of the single task node $\mathrm{v_i}$, and $\mathrm{sub}(\mathrm{v_i})$ represents the subsequent task node set (or sub task set) of the single task node $\mathrm{v_i}$.

Here we consider the computing node modelling next. Before that we need to do some declare first, the scheduling model of dependent tasks is consistent with the scheduling model of independent tasks, which has demonstrated in the

previous chapter and I will not repeat again.

The resource layer is mainly responsible for the calculation of the tasks that based on some prioritization sorting and scheduling, and returns the results [3]. Due to the resource layer consists of large computer clusters and network devices, each computing node is connected by the hubs, gateways and routers. It will inevitably lead to the cost of communication in the course of the process of each computing node communicates with the others. And the cost of communication here just refers to the loss of time. In the heterogeneous computing node sets, the computing ability of each node is different, so we can also consider these nodes as an undirected graph G' as follows:

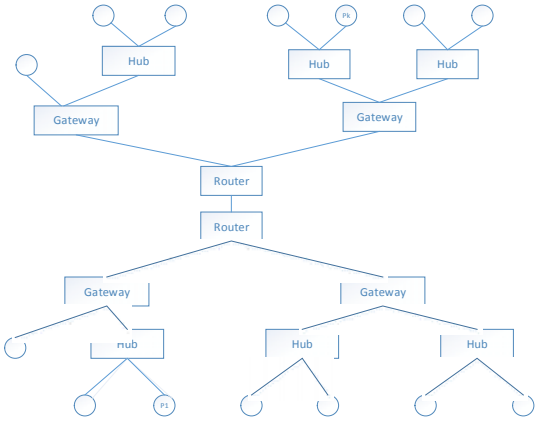

Figure 2. sketch map of computing nodes in resource layer

We use a tow-tuples to represent this figure as $G' =< P, L >$ in which the P represents a collection of computing nodes and it is expressed as $P = \{p_1, p_2, \dots, p_m\}$, m is the total number of the computing nodes. The L represents a collection of edges between each tow computing nodes, it is expressed as $L = \{l_{ij} | i, j \in m, i \neq j\}$. In this article, we assume that each computing node can be interconnected with others. In other words, there is always an edge between each tow computing nodes.

In the process of task modelling, we have defined q_i as the average value of computing cost of each task in all resource nodes. However, in order to further analysis, we will make the cost of task computing in node p_i is $q(v_i, p_j)$. We assume that the total number of computing nodes in the resource layer is m, and the total number of tasks in the task-set is n, then we have the table as following:

Table 1. the cost of task computing in resource node p_i

$\boldsymbol{q(v_i, p_j)}$	$\boldsymbol{p_1}$	$\boldsymbol{p_2}$	$\cdots$	$\boldsymbol{p_m}$
$\boldsymbol{v_1}$	$q(v_1, p_1)$	$q(v_1, p_2)$	$\cdots$	$q(v_1, p_m)$
$\boldsymbol{v_2}$	$q(v_2, p_1)$	$q(v_2, p_2)$	$\cdots$	$q(v_2, p_m)$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\boldsymbol{v_n}$	$q(v_n, p_1)$	$q(v_n, p_2)$	$\cdots$	$q(v_n, p_m)$

From the table above, we can find that the costs of task computing in different nodes are inconsistent. In the process of quantitate the computing nodes, we can't judge the speed of calculation of computing nodes simply by the speed of the task processing (dominant frequency), but quantitate the execution time of the computing nodes by different task types. And thus, it is closer to the actual situation of the real life [4].

2 The Analysis of Tasks Execution

Before analysing the problem of scheduling, we should have a study of the execution conditions and time points of the dependent tasks firstly, and then schedule it according to the correspond scheduling strategy [5].

2.1 Definitions

We should make sure that the resource nodes are in the ready state first, and the execution results of all parent tasks must be known before a task be executed. If not, the task node can't be executed. Based on this feature, we make definitions of the concept of time as follows:

Definition 2: $\text{ready}(p_j)$ is the time of computing node p_j get ready. Otherwise the task node can be calculated on the computing node while the computing node gets in the point of time.

Definition 3: $\text{finit}(v_i, p_j)$ is the completion time of task v_i that be executed on the computing node p_j. In this article, if the start time is 0, then the $\text{finit}(v_i, p_j)$ is given by: $\text{finit}(v_i, p_j) = q(v_i, p_j)$.

Definition 4: $\text{start}(v_i, p_j)$ is the beginning time of task v_i, be executed on computing node p_j.

If the parent task-sets $\text{pre}(v_i)$ of task v_i are all on the same computing node p_j, the cost of communication $w_{ij} = 0$. That means $\text{start}(v_i, p_j) = \text{finit}(\text{pre}(v_i), p_j)$. But in general circumstances, different task nodes will be distributed to different computing nodes, thus we can give more complete definition to definition 3 and definition 4 as follows:

$$start(v_j, p_j) = max\{ready(p_j), max\{finit(v_j, p_k) + w_{kj}\}\}, \ v_j \in pre(v_i) \quad (1)$$

From the equation above we can know that if we want to get the execution time of task node v_i, we shall calculate the ready time $\text{ready}(p_j)$ of the computing node p_j which task v_i will be executed on, and the maximum value of the sum of the complete time of task node v_j, the parent task node of v_i, and the cost of communication of them. Then the execution time is the larger one of the time above. Similarly, the complete time of task node v_i that will be

executed on computing node p_j is the sum of the start execution time of task v_i and the cost of execution.

$$finit(v_j, p_k) = start(v_i, p_j) + q(v_i, p_j) \quad (2)$$

As we know that the start time of a task and the ready time of the computing node which the task will be executed on would not necessarily equal, so we can give a definition as follows:

Definition 5: $wait(v_j, p_k)$ is the time that the task node v_i has arrived the computing node p_j which has ready, but the result of parent task node of task v_i have not been transmitted over. We call this period as the waiting time for the task.

2.2 Correlation analysis

In this paper, the research of the algorithm focuses on that how to complete the task scheduling in the shortest time through a strategy. And to a certain extent, we can suppose that the unit energy consumption of the computing node is known, and cut down the execution time is helpful to reduce the energy consumption, so as to achieve the effect of energy-saving [6]. How to adjust the frequency of computing node in the resource layer to ensure that the tasks have been completed before deadlines will not be considered in this article.

At the same time, we need to make the following assumptions on the algorithm as below:

1. The real-time status between the computing nodes has known. Such as the energy consumption of computing nodes, the CPU performance, memory size, the bandwidth of network and the status of the computing nodes, and so on.
2. It is interconnected between each tow computing nodes, and the cost of communication is known.
3. Since the model map of the task nodes has been identified, so the dependent relationship between tasks has been clear.
4. Each task can be executed on different computing nodes, and there is no case can not be executed.

The algorithm is inspired by the table scheduling algorithm and task duplication algorithm in this paper. In the table scheduling algorithm, we should establish the priority of task-set first. And the priority is established by the sum of the maximum length of task node v_i to the end task node on the critical path and the maximum length of start task node to task node v_i on the critical path. In this way, the tasks node v_i on the critical path can be executed in advance. But on the whole, only the tasks on the critical path can be executed in this way, and the other tasks can not be. Due to there are lots of key factors that will affect the dependency of tasks, so we should consider more factors to adjust the algorithm to ensure the execution time of other tasks can be optimized.

In the table scheduling algorithm, the task queue formed according to the established priority, and then it will be distributed to the computing nodes. In process of choosing computing nodes, the nodes that can ensure the task v_i can be completed in the shortest time are needed. And thus, the cost of communication between the computing nodes or the cost of communication between tasks doesn't take into account. It would lead to the waiting time of task node v_i is too long while the results of the parent task nodes haven't been transmitted over under the circumstance that the task dependence degree is so strong, and it would delay the whole complete time of task-set further. After we were inspired by the thought of task duplication, we know that if the parent tasks are duplicated to the computing node which the task v_i will be executed, the costs of communication will be eliminated, then the task node v_i can be executed in advance [7]. In the HCPFD algorithm has proposed to cut down the cost of communication between tasks by duplicating the parent tasks that on the critical path, but it doesn't take the tasks on non-critical path into account on the whole. Based on the analysis above, we know, when we duplicate the parent tasks we should choose not only the tasks on the critical path but also the tasks on non-critical path to analysis, then pick the tasks by the priority that determined by a certain rule, only in this way can we cut down the execution time of the entire task-set on the whole [8]. Finally, the computing nodes that can guarantee the complete time of task node v_i shortest are the nodes we needed in the process we choose the optimized computing nodes.

3 Algorithm Analysis and design

Based on the analysis of previous chapters and sections, we know that there are two major research directions of the algorithm:

1. Make the tasks in descending order of priority. The first task has the highest priority in the task-set.
2. In the task distributing phase, before to distribute the tasks to the computing nodes, whether the task meets the conditions of task duplication or not should be taken into account first.

3.1 Task priority calculation

In the analysis of previous, we have mentioned that we should not just consider the tasks on the critical path while prioritizing the tasks, but all tasks shall be taken into account. As we know, there are dependences between tasks, a single task node possesses out-degree, in-degree, cost of task execution and cost of communication etc. In this article, our consideration will focus on the out-degree of task and the cost of communication between tasks, and prioritize the priority of task nodes based on this.

3.1.1The out-degree of task node

We use $\mathrm{Out}(\mathrm{v_i})$ to represent the out-degree of tasks, and from the figure 1, we know that $\mathrm{Out}(\mathrm{v_1}) = 4, \mathrm{Out}(\mathrm{v_2}) = 2, \mathrm{Out}(\mathrm{v_8}) = 0$, the out-degrees of other tasks are all 1. If the out-degree value of a task is larger, the impact to the subsequent task is greater. Because if the task is not done, the execution time of a larger number of subsequent tasks will be delayed. And if the task can be completed preferentially, the start time of the subsequent tasks will be brought forward, too.

3.1.2 The cost of communication between task nodes

In general, each tow dependence task nodes exist the cost of communication, unless the tow tasks are both on the same computing node and the cost of communication can be ignored. According to the definition of start time $\mathrm{start}(\mathrm{v_i}, \mathrm{p_j})$ we can know that before task $\mathrm{v_i}$ be executed on computing node $\mathrm{p_j}$, the results of parent tasks must be transmitted over to $\mathrm{p_j}$, or else task $\mathrm{v_i}$ must wait. So the cost of communication on the out-degree of a task node should also be considered. But in order to reflect the integrity and the global view better, the algorithm proposed in this article take the costs of communication of all out-degrees of the task node into account. And we can get the equation as below:

$$SC(v_i) = \sum w_{ij} \quad (3)$$

w_{ij} is the cost of communication between the task node v_i and its sub task node v_j. We call the $SC(v_i)$ as the sum of cost of communication on out-degree of task node v_i. And the larger $SC(v_i)$ is, the greater impact to the start execution time of the subsequent tasks is. Conversely, the earlier the subsequent tasks will be executed.

Above all, this article gives the method of calculating the priority of each task as the equation below:

$$PL = (v_i) = q_i + max\{PL(v_j) + w_{ij}\},\ v_j \in sub(v_i) \quad (4)$$

We can find that from the equation above, the task nodes are traversed from bottom-up in the algorithm. If there is no subsequent task node following task $\mathrm{v_i}$, the priority of $\mathrm{v_i}$ is the average cost of calculation. Or else, select a task node $\mathrm{v_j}$ from the subsequent task-set, which ensures the sum of the priority of $\mathrm{v_j}$ and the cost of communication between $\mathrm{v_j}$ and $\mathrm{v_i}$ is maximum. And due to the equation above just takes only one task node from the task-set into account unilaterally, but not all the subsequent nodes, so we can reform the equation based on the analysis above as below:

$$PL'(v_i) = Out(v_i) \times PL(v_i) + SC(v_i) \quad (5)$$

In this formula, the priority of each task node will be calculated by traversing in the way of bottom-up, and it can reflect the integrity of all tasks very well, what's more, all tasks will be taken into account but not just the tasks on the critical path in the process of priority calculating, and this idea is consistent with the original purpose of this paper [9].

3.2 Task Duplication

As we know, the cost of calculation of each task on the computing nodes is known, and if you want the tasks to be completed in the earliest time, tasks shall be executed as early as possible so that the total completion time of the task-set can be promoted.

And here the analysis of the factors that will affect the begin time of task execution is especially important. The begin time of task execution is determined by the time of the results of parent nodes transmitted over to the computing nodes and the time of computing node ready. So we give the factors that will affect the begin time of tasks as follows:

1. The computing node p_j which the task v_i will be executed on has ready.
2. The execution results of parent nodes of task v_i has transmitted over to the computing node p_j.

Both of factors above are necessary. If the results of parent have arrived but the computing node not yet ready, the task $\mathrm{v_i}$ can't be executed. If the computing node $\mathrm{p_j}$ has ready but the results of parent nodes haven't arrived, the task $\mathrm{v_i}$ can't be executed, and it must be wait. And we can give the equation as follows:

$$wait(v_i, p_j) = start(v_i, p_j) - ready(p_j) \quad (6)$$

So we can know that task waiting time is the difference between the start time of the task node and the ready time of the computing node. And take full advantage of the task waiting time $\mathrm{wait}(\mathrm{v_i}, \mathrm{p_j})$ before the task $\mathrm{v_i}$ be executed will be an effective way to bring forward the complete time. Based on the elicitation of the task duplication, if the parent task nodes of task $\mathrm{v_i}$ are copied redundantly to the computing node $\mathrm{p_j}$ and the execution time of parent nodes are shorter than $\mathrm{wait}(\mathrm{v_i}, \mathrm{p_j})$ can help to promote the overall completion time.

We can give the factors that will affect the parent task nodes duplication from the idea above as follows:

1. The parent task nodes execution time on the computing node p_j shouldn't longer than $wait(v_i, p_j)$.
2. The relationship of priority between task nodes can't be violated.

Furthermore, we can make some rules as below:

Rule1. If the execution time of task node $\mathrm{v_i}$ on the computing node $\mathrm{p_j}$ is $\mathrm{wait}(\mathrm{v_i}, \mathrm{p_j})$, the execution time of parent task node of $\mathrm{v_i}$ on computing node $\mathrm{p_j}$ is $\mathrm{q}(\mathrm{v_j}, \mathrm{p_j})$, while the equation below is true then the parent nodes of task $\mathrm{v_i}$ can be duplicated.

$$wait(v_i, p_j) \geq q(v_j, p_j) \& start(v_i, p_j) > finit(v_j, p_j),$$

$$v_j \in pre(v_i) \quad (7)$$

Maybe there are several parent task nodes that meet the equation above, so we should sort the parent tasks, and then select the parent task with longest of both completion time and transmission time to duplicate every time. And in order to describe the parent task node with longest time, we can make the definition to the optimal parent task node as follows:

Definition 5: the parent task node $\mathrm{mv}\{\mathrm{v_i}, \mathrm{p_j}\}$ of task $\mathrm{v_i}$ should meet the following formula:

$$mv\{v_i, p_j\} = max\{finit(v_j, p_k) + w_{kj}\}, v_j \in pre(v_i) \quad (8)$$

For facilitating the following analysis, we stipulate the parent task that meets the rule of 1 and sorted in the second place in the task queue that based on the definition of 5 is the subordinate parent task, and record as $\mathrm{smv}\{\mathrm{v_i}, \mathrm{p_j}\}$.

After that, we can get the following character and make a simple proof as below:

Character 1: when a task $\mathrm{v_i}$ exists on the computing node $\mathrm{p_j}$, and if $\mathrm{mv}\{\mathrm{v_i}, \mathrm{p_j}\}$ meets the rule of 1, then the $\mathrm{mv}\{\mathrm{v_i}, \mathrm{p_j}\}$ can be copied to the computing node $\mathrm{p_j}$, so as the start time $\mathrm{start}(\mathrm{v_i}, \mathrm{p_j})$ of task $\mathrm{v_i}$ can be brought forward.

And the proof steps are as follows:

1. It's unnecessary to consider duplication while task v_i have no parent task node.
2. The start time $start(v_i, p_j)$ can be brought forward as $finit(mv\{v_i, p_j\}, p_j)$ while there is only one parent task node of task v_i.
3. While task v_i has two or more parent nodes, and we shall discuss on two cases:

Case 1: The result of $\mathrm{finit}(\mathrm{mv}\{\mathrm{v_i}, \mathrm{p_j}\}, \mathrm{p_j}) > \mathrm{smv}\{\mathrm{v_i}, \mathrm{p_k}\}$ has arrived on the computing node $\mathrm{p_j}$. In this case, $\mathrm{mv}\{\mathrm{v_i}, \mathrm{p_j}\}$ has already on the computing node $\mathrm{p_j}$, and the start time $\mathrm{start}(\mathrm{v_i}, \mathrm{p_j})$ of task $\mathrm{v_i}$ will be brought forward to $\mathrm{finit}(\mathrm{mv}\{\mathrm{v_i}, \mathrm{p_j}\}, \mathrm{p_j})$.

Case 2: The result of $\mathrm{finit}(\mathrm{mv}\{\mathrm{v_i}, \mathrm{p_j}\}, \mathrm{p_j}) < \mathrm{smv}\{\mathrm{v_i}, \mathrm{p_k}\}$ has arrived on the computing node $\mathrm{p_j}$. In this case, the start time $\mathrm{start}(\mathrm{v_i}, \mathrm{p_j})$ of task $\mathrm{v_i}$ will be brought forward to $\mathrm{finit}(\mathrm{smv}\{\mathrm{v_i}, \mathrm{p_j}\}, \mathrm{p_j}) + \mathrm{w_{kj}}$.

In summary, the start execution time $\mathrm{start}(\mathrm{v_i}, \mathrm{p_j})$ of task $\mathrm{v_i}$ will always be brought forward in circumstances. And the proof end.

At this point, we can elaborate the strategy of task duplication as following:

1. Traverse through all task nodes and confirm the dependence relationship between task nodes.
2. For each task node v_i, find the parent task-set, and sort the parent task nodes in descending order according to the arrival time of execution results. And the first one is the optimal parent task node $mv\{v_i, p_j\}$.
3. Calculate the waiting time of task v_i. If the optimal parent task node meets the rule of 1, add the parent task into the waiting time of computing node which task v_i will be executed. Then delete this parent task node in the parent task-set. After that, the first one in the rest tasks becomes the optimal target.
4. Update the waiting time of the computing node.
5. Repeat the step 1to step 4.

If the rule of 1can be met, it's obviously that it would not bring forward the start time $\mathrm{start}(\mathrm{v_i}, \mathrm{p_j})$ of task $\mathrm{v_i}$ and even delay the time if parent tasks of $\mathrm{v_i}$ be duplicated redundantly to the computing node.

3.3 Algorithm

Based on the analysis above, the algorithm proposed in this article scheduling the dependent tasks in 3 steps. Firstly, traverse through all tasks and calculate the priority of each task, then sort the tasks in descending order. Secondly, calculate the execution time of task v_i on each computing node and confirm the waiting time of task v_i. Find all parent task nodes, and sort them. Copy the parent tasks that meet the rule of 1 to the waiting time and wait to be executed. Finally, calculate the complete time of the task and update the ready time of computing node [10].

A detailed description is as follows:

1. Calculate the out-degree of each task in the task-set, the cost of communication of out-degree, the average cost of calculation of tasks and the task priority PL.
2. Calculate each task priority $PL'(v_i)$ that in the task-set.
3. Sort the tasks in descending order according to the task priority, and form a task queue.
4. If the task set is not empty, do
5. Pick out the task with highest priority, find all the parent tasks of it.
6. Traverse through all the computing nodes.
7. For every computing node p_j, calculate the arrival time $finit(v_j, p_k) + w_{ki}$ of the results of the parent nodes of task v_j that will be executed on p_j. Sort the tasks according to the results and we can get the optimal parent task $mv\{v_i, p_j\}$.
8. Calculate the waiting time of tasks.
9. Judge whether the optimal parent task meets the rule of 1. If satisfied, copy it to the waiting time of task and update the waiting time, then repeat. Or else, turn to the step 10.
10. Calculate $finit(v_j, p_j)$.
11. End traverse of the computing node p_j.
12. Distribute the task to the corresponding computing node that can complete the task in shortest time.
13. Update the ready time of computing node.
14. Repeat step 5 to step 13 until the task queue is empty.

We assume that there are n tasks will be distributed to m computing nodes, then we can get the time complexity is $O(n)$ in phase of task sorting, and the time complexity is $O(mn^2)$ in phase of task duplicating and distributing. So we can get the time complexity of this algorithm is $O(mn^2)$.

Summary

We have analysed and modelled the dependent task nodes and the computing nodes in this paper, and taken the relationships between task-sets and the heterogeneous characteristics of computing nodes in resource layer in Cloud computing environment into account. Inspired by the table scheduling algorithm and the idea of task duplication, proposed a dependent task scheduling algorithm. And at last, we can find that it is possible to bring forward the execution time of the dependent tasks in the task-sets, so as to shorten the finish time to a certain extent.

Acknowledgement

This work is financially supported by Yunnan Provincial Department of Education Fund for Scientific Research of major special projects under Grant No. ZD2014001, and by the Open Foundation of Key Laboratory of Software Engineering of Yunnan Province under Grant No. 2015SE102, 2015SE204.

References

1. Jayaswal S, Agarwal P. Balancing U-shaped assembly lines with resource dependent task times: A Simulated Annealing approach[J]. Journal of Manufacturing Systems, 2014, 33(4): 522-534.
2. Munir E U, Mohsin S, Hussain A, et al. SDBATS: a novel algorithm for task scheduling in heterogeneous computing systems[C]//Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. IEEE, 2013: 43-53.
3. Ye, Hong. "Research on Emergency Resource Scheduling in Smart City based on HPSO Algorithm." city 5 (2015): 6.
4. Mu R. Node Resource Optimization Algorithm Based on Wireless Network[J]. Journal of Software Engineering, 2015, 9(4): 702-720.
5. Wu Z, Liu X, Ni Z, et al. A market-oriented hierarchical scheduling strategy in cloud workflow systems[J]. The Journal of Supercomputing, 2013, 63(1): 256-293.
6. Xiong W, Wang Y, Mathiesen B V, et al. Heat roadmap China: New heat strategy to reduce energy consumption towards 2030[J]. Energy, 2015, 81: 274-285.
7. Arabnejad H, Barbosa J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table[J]. Parallel and Distributed Systems, IEEE Transactions on, 2014, 25(3): 682-694.
8. Kaur R, Kaur R. Multiprocessor scheduling using task duplication based scheduling algorithms: A review paper[J]. International Journal of Application or Innovation in Engineering and Management, 2013, 2(4): 311-317.
9. Zhang S, Wang Y, Liu W, et al. A model for estimating the out-degree of nodes in associated semantic network from semantic feature view[J]. Concurrency and Computation: Practice and Experience, 2016.
10. Qian Z, Hong L, Jin S. A Scheduling Algorithm of Dependent Tasks on Virtual Computing Environment[C]//Computer Sciences and Applications (CSA), 2013 International Conference on. IEEE, 2013: 266-271.