2017 International Conference on Computing Intelligence and Information System

Workflow Execution Plan Generation in the Cloud Computing Environment Based on an Improved List Scheduling Algorithm

Xiaoying WANG, Chengshui NIU, Yu-an ZHANG
Department of Computer Science and Technology
Qinghai University
Xining, Qinghai, China, 810016
E-mail: {wxy_cta, csniu, yazhang}@qhu.edu.cn

Lei ZHANG
College of Computer Science
Sichuan University
Chengdu, China, 610064
E-mail: zhanglei@scu.edu.cn

***Abstract*—Focusing on the higher ratio of processor utilization and lower execution cost of a scientific workflow in the cloud environment, an improved list scheduling algorithm was proposed in this paper. This algorithm combines the ideas of both list scheduling and task duplication. According to the priority of the tasks, choosing reasonable parent task to replicate can help reduce the overhead between tasks. To properly insert tasks during processor idling time can help to increase the processor utilization. Based on these, we proposed an improved strategy to generate the workflow execution plan, called *EPGILS*. Experiment results show that the algorithm is feasible and efficient in reducing the task completion time and improving the utilization ratio of the processor.**

Keywords-scientific workflow; cloud computing environment; list scheduling; task scheduling; execution plan

I. INTRODUCTION

Scientific workflow (SWF) is a new type of application developed rapidly in recent years. It can support scientists and researchers to integrate, construct and cooperate various distributed data services and software tools, providing a management platform for complex workflow definition and execution automation of scientific computations [1]. Compared to traditional Business Workflow (BWF), one of the most important features of the scientific workflow is data-oriented. Nowadays, SWF is becoming computation-intensive and data-intensive [2], since the data are generated continuously and fast and the relevant computation becomes complex accordingly. Normal computing environment can hardly meet the requirement of SWF. Hence, cloud computing environment provides a new deployment and execution paradigm for scientific applications, since its infrastructure is usually comprised of high performance computing resources and massive storage resources.

In the cloud computing environment, customers can rent the resources on demand. However, changing the resource allocation scheme will involve the creation of instances and data movements, which incurs costs and might have impact on the workflow execution efficiency and total costs [3]. Thus, it's very important to design reasonable workflow execution plans for both users and service providers [4]. In other words, generating a plan means to map the tasks in the workflow onto the computing resources appropriately.

Hence, in this paper, we propose a task scheduling algorithm, named *EPGILS* (Execution Plan Generation based on Improved List Scheduling), for workflow execution plan generation based on the list scheduling algorithm. The motivation is to utilize the idle time of processors efficiently, thereby reducing the number of necessary processors involved in task execution. By *EPGILS*, the advantages of list scheduling and task duplication are combined to generate the execution plan of scientific workflow tasks upon the homogeneous cloud infrastructure. Experiment results show that this strategy can effectively enhance the parallelism of the workflow execution. As a result, not only the total spent time could be reduced, but also the idle processor time slices could be utilized sufficiently, and thus the total execution cost for running these tasks could be lowered.

II. RELATED WORK

The key issue of workflow execution plan generation is how to schedule a task onto a proper computing resource [5]. Most algorithms proposed in prior work are heuristic, including clustering [6], task duplication [7], list scheduling [8] and GA (Genetic Algorithm)-based algorithms [9]. Clustering-based algorithm tends to map a cluster to a virtual machine (VM), but doesn't consider the task duplication problem among multiple clusters. Task duplication algorithm will replicate tasks from one VM to another in order to reduce the communication overhead of different tasks, but the target server is usually hard to choose. Genetic algorithm can lead to close-to-optimal convergence time by selection, crossover and mutation, but the next task to-be-scheduled can hardly be pre-determined due to the randomness during the selection process. Also, the crossover and mutation process usually consumes much computation time [10]. List scheduling algorithm determines the priority of each task before it's executed so that the entire task queue could be given before execution. Heuristic algorithms were widely-used by many researchers since they are often robust and cost-efficient.

This paper proposes an improved algorithm *EPGILS* based on the list scheduling algorithm and the task duplication based algorithm. In a homogenous cloud environment, this method can generate workflow execution plan in an effective way, which can enhance the parallelism of the workflow execution. As a result, not only the makespan of the workflow could be reduced, but also the processors could be utilized more sufficiently, leading to a lower expense for executing the workflows.

III. Problem Definition

To generate an execution plan of the scientific workflow, it is necessary to map the tasks in the workflow to computing resources properly. Assume that there are n tasks in the workflow in total and m processors in the cloud environment. Denote F as the mapping function, T as the task set and V as the set of processors in the cloud environment, as follows.

$$T=\{t_i \mid t_i \text{ is the } i\text{-th task}, i=1,2,3,\ldots,n\} \tag{1}$$

$$V=\{v_j \mid v_j \text{ is the } j\text{-th processor}, j=1,2,3, \ldots,m\} \tag{2}$$

$$F(T,V)=\{<t_i,v_j> \mid t_i \in T, v_j \in V\} \tag{3}$$

Multiple tasks in a workflow have dependency relationship among each other, which can be described by Directed Acyclic Graphs (DAG), denoted as $G(T,E,M,W)$, where E is the set of edges representing the communications between tasks, M is the set of communication overhead values, W is the set of task computing overhead values. The sets can be expressed as

$$E=\{e_{ij} \mid e_{ij} \text{ is the edge from } t_i \text{ to } t_j, i,j\in[1,n] \} \tag{4}$$

$$\text{M}=\{m_{ij} \mid m_{ij} \text{ is the communication overhead between } t_i \text{ and } t_j, \text{i,j}\square[1,\text{n}]\} \tag{5}$$

$$W=\{c_i \mid c_i \text{ is the computing overhead of } t_i, i\in[1,n] \} \tag{6}$$

Denote $est(t_i)$ and $eft(t_i)$ as the earliest start time and earliest finish time of task t_i, then

$$est(t_{begin})=0 \tag{7}$$

$$eft(t_i)=est(t_i)+c_i \tag{8}$$

$$t_{makespan}=eft(t_{end}) \tag{9}$$

where t_{begin} is the beginning task and t_{end} is the ending task, and $t_{makespan}$ denotes the makespan of entire workflow.

For task t_i, denote $pre(t_i)$ as the previous task of t_i. Then, the free time on processor v_k from the moment when the last task finishes to the moment when the next task starts could be calculated as

$$\text{Free}(t_{pre(t_i)},t_i,v_k) = est(t_i,v_k) - eft(pre(t_i),v_k) \tag{10}$$

Denote P_x as the path in the DAG, and then the finish time of the path could be expressed as

$$\text{Span}(P_x) = \sum_{t_i \in P_x} (c_i) + \sum_{t_i,t_j \in P_x} (m_{ij}) \tag{11}$$

Then, we use $reach(t_{pre(t_i)},t_i)$ to represent the time spent from task $pre(t_i)$ to t_i. There might be two kinds of scenarios, as follows.

1) t_i and *pre(t_i)* locate on the same VM. In this case,

$$reach(t_{pre(t_i)},t_i) = \sum_{t_i \in P_x} (c_i) + m_{pre(t_i),i} \tag{12}$$

2) t_i and *pre(t_i)* locate on different VMs. In this case,

$$reach(t_{pre(t_i)},t_i) = \text{Span}(P_x, t_i) \tag{13}$$

where $Span(P_x,t_i)$ is the total time spent from the beginning task to t_i along the path P_x.

IV. Execution Plan Generation

The purpose of generating the workflow execution plan is to reasonably allocate tasks onto processors, so the key point is to first determine the number of demanded processors. The *EPGILS* strategy we propose here aims at reducing the task makespan and increasing the VM utilization. We combined the list scheduling algorithm and task duplication method and applied the approach upon a homogeneous cloud environment will unlimited number of VMs. There are two stages in the *EPGILS* strategy, as follows:

1) Generate the path priority list of the to-be-schedued tasks;

2) Get the scheduled task according to the above list and allocate tasks onto processors, which is called task pre-scheduling in our algorithm.

A. Determine the priority of the to-be-scheduled tasks

We use an example DAG as shown in Fig.1 to demonstrate our algorithm, where a circle represents a task node t_i, and the notation number beside it means the computing overhead c_i. The number beside an edge represents the communication overhead m_{ij} between t_i and t_j. If the two tasks are allocated to the same processor, we can ignore the communication overhead (i.e. $m_{ij} = 0$).

Then, all possible paths P_x could be obtained by traversing from the beginning task node to the ending task node in the DAG. The makespan time of each path could be calculated by Eq.(11). Then, the values of *Span* for all the paths will be sorted in descending order, and then the path with the largest *Span* value is the critical path in the workflow.

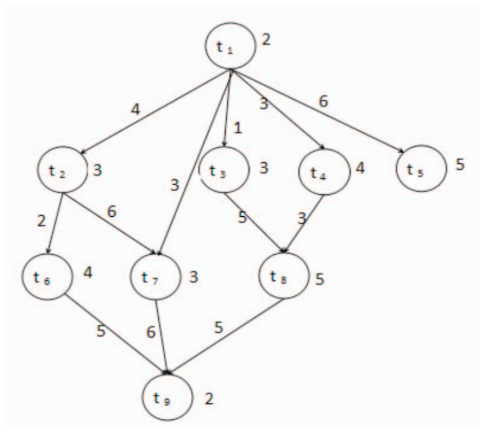

Figure 1. Example of a workflow DAG.

B. Task pre-scheduling

By task pre-scheduling, the mapping scheme from tasks to processors will be determined, and the number of needed processors will be calculated. The paths are examined by their priority in descending order, and the tasks in the current path will then be scheduled one by one. The task to be selected for scheduling should meet the following two requirements:

1) The task hasn't been scheduled yet.

2) The task doesn't have previous tasks or all of its previous ones have already been scheduled.

In the current path, a certain task meeting the above two conditions should be scheduled, and then next. If there are no tasks to be scheduled, the next path should be examined.

When scheduling a task t_i, the number of its previous nodes should be checked. If it doesn't have previous node, it will be mapped onto a new processor; If the task has only one previous node (denoted as t_j) and this node has already been scheduled, then the current task will be mapped onto a new processor, and all of the tasks mapped on the same processor with t_j should be copied to the current processor allocated to t_i. If the task has more than one previous node, these parent nodes should be sorted by the reaching time from the previous node to t_i in descending order, generating the scheduling queue of parent nodes. The head node (denoted as t_h) will be mapped onto the same processor with t_i, and the earliest finishing time of this processor (denoted as v_k) should be updated. Then, we check each parent node (denoted as $pre(t_i)$) to see whether it meets the following two insertion criteria:

$$\mathrm{Free}(t_{pre(t_i)}, t_i, v_k) \geq c_{pre(t_i)} \qquad (14)$$

$$\mathrm{eft}(\mathrm{pre}(t_i), v_k) < \mathrm{est}(t_i, v_k) \qquad (15)$$

If a task meeting the insertion criteria is found, it will be inserted onto processor v_k. All the parent nodes of t_i will be judged in this way until there are no parent tasks meeting the criteria or all of the parent tasks have been inserted.

V. THE EPGILS ALGORITHM

In this section, the core algorithm of EPGILS is described and a case study is conducted accordingly.

A. Algorithm description

The EPGILS algorithm could be performed in following steps:

1) All possible paths in the DAG will be traversed;

2) The Span values of every path are calculated;

3) The scheduling list of paths is generated by sorting the Span values in descending order;

4) Examing and process the next path P_x in the list;

5) Look for an unscheduled task t_i in P_x. If no task is found, go to step 4.

6) If t_i doesn't have previous nodes or all previous node of t_i has been scheduled, go to step 4.

7) Caculate the number of previous nodes of t_i, denoted as n.

8) If n=0, t_i is the beginning node, and will be mapped onto a new processor.

9) If n=1, map t_i onto the same processor with the previous node.

10) If n>1, the previous task queue is generated by sorting the previous nodes by the reaching time to t_i in descending order.

11) Put the latest previous task together with t_i on the same processor, and then update the finishing time of t_i and calculate the free time of the processor.

12) Traverse the previous task queue. If tasks meeting the insertion criteria could be found, it will be inserted into the processor blank slot, and the finishing time of the task will be updated; turn to step 5 otherwise.

13) If all the paths in the list has been examined, end the algorithm; go to step 4 otherwise.

B. Case study

Take the DAG shown in Fig.1 as an example. First, we can obtain the path priority list, as shown in Table 1.

TABLE I. PATH PRIRORITY LIST

Priority	Path p_x	Span	Scheduled tasks
1	t_1, t_2, t_7, t_9	26	t_1 t_2 t_7
2	t_1, t_4, t_8, t_9	24	t_4
3	t_1, t_3, t_8, t_9	23	t_3 t_8
4	t_1, t_2, t_6, t_9	22	t_6 t_9
5	t_1, t_7, t_9	16	——
6	t_1, t_5	13	t_1 t_5

Then, we check the first path and find that t_1 is the beginning node, so we can put it on a new processor, denoted as v_k. Next, examine t_2 and we can see that the only parent node of t_2 has already been scheduled, so we put t_2 onto v_k as well. Next, we check the task t_7 and found that all of its previous nodes have been scheduled. In this case, the reaching time from them to t_7 is calculated as $reach(t_1, t_7)$=2+3=5 and $reach(t_2, t_7)$=2+3+6=11, and then the finishing time of t_7 could be updated. The next task is t_9, but it has three previous nodes (t_6, t_7, t_8) which haven't all been scheduled. In this case, the current path will be skipped and the next path will be examined.

According to the above procedure, when the task t_8 in path 3 is dealt with, we can see that its previous nodes t_3 and t_4 have both been scheduled. Then, the reaching time from them to t_8 can be calculated as $reach(t_3, t_8)$=2+3+5=10 and $reach(t_4, t_8)$=2+4+3=9. Then, t_8 is allocated together with t_3 and the finishing time could be updated. Now the task list on the processor is $\{t_1, t_3, t_8\}$, and $eft(t_3)$=5, $est(t_8)$=9. According to Eq. (10), $Free(t_3, t_8, v_k)=9-5=4 \geq c_4=4$ and $eft(t_4)=6 < est(t_8)=9$. These meet the insertion criteria, so that t_4 can be inserted into the queue $\{t_1, t_3, t_8\}$, and then the list becomes $\{t_1, t_3, t_4, t_8\}$. The next task is t_9, which cannot be scheduled now, and thus the algorithm goes to path 4.

In path 4, there is only one previous node t_2 for task t_6, then all of the tasks before t_2 will be replicated and put to another processor together with t_6. Then, the same process will be performed to t_9 as to t_8.

The final result is that four processors have to be used, and running the tasks as follows.

$v_1=\{t_1,t_3,t_4,t_8,t_9\}$, $v_2=\{t_1,t_2,t_7\}$,

$v_3=\{t_1,t_2,t_6\}$, $v_4=\{t_1,t_5\}$

Finally, the makespan of the entire workflow is $eft(t_9)$=16.

VI. Experimental Results

To evaluate the performance of the proposed *EPGILS* algorithm, we compare it with the other two algorithms – *TDS* (Task Duplication based Scheduling) [11] and *TDCS* (Task Duplication based Clustering Scheduling) [12]. We conducted a series of experiment based on the CloudSim platform.

We choose three kinds of performance metrics for comparison, including the workflow makespan, the number of occupied processors and the utilization of the processors. Ten kinds of different workflow are selected, and the number of tasks in the workflows is set to {10, 20, 30, 40, 50, 60, 70, 80, 90, 100} correspondingly. 20 instances will be generated randomly for each type of workflow, and the average value of every performance metric will be calculated [13].

Figure 2 shows the comparison of the workflow makespan. It can be seen from the figure that when the number of tasks is below 50, the difference of the three algorithms is not very obvious. As the number increases, the *EPGILS* algorithm behaves better than the other two kinds of algorithms, with a lower increasing speed.

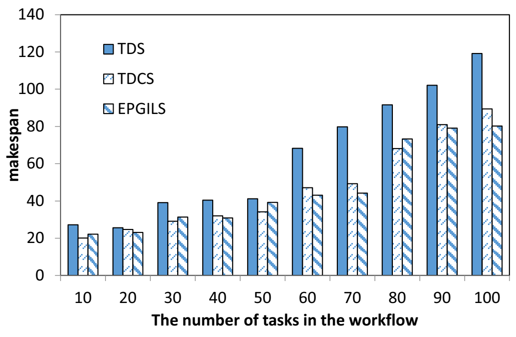

Figure 2. The Workflow Makespan.

Figure 3 shows the number of occupied processors by the three different algorithms. From the figure, it can be observed that *EPGILS* needs much fewer processors compared to *TDS* and *TDCS*. As the number of tasks increases, especially when beyond 80, the advantage of *EPGILS* becomes more remarkable. This means in a large-scale cloud environment, fewer processors are needed and then the energy consumption will be saved consequently.

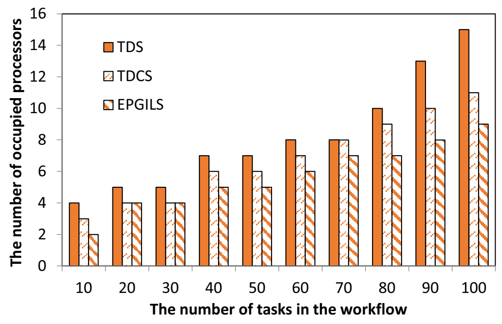

Figure 3. The Number of Occupied Processors.

Figure 4 shows the relative processor utilization rate of these algorithms compared to *EPGILS*. We can see that the *TDCS* leads to 80%-90% utilization than *EPGILS*, while *TDS* can only utilize 70%-80% of the processor time compared to *EPGILS*. Especially as the number of tasks increases, the relative utilization rate goes down, which means *EPGILS* becomes more efficient relatively.

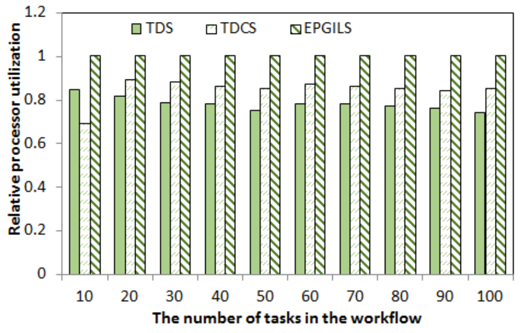

Figure 4. The Processor Utilization.

From the above results, it's obvious that *EPGILS* behaves much better than the other two algorithms, considering the three aspects including makespan, the number of used processors, and the processor utilization.

VII. Conclusion and Future Work

This paper aims at executing scientific workflows in the cloud environment with many homogenous processors, and proposes an improve algorithm to generate the workflow execution plan more effectively. The algorithm called *EPGILS* combines the list scheduling and task duplication scheduling algorithm together and thus reduces the communication overhead between tasks. At the same time, the makespan of the entire workflow will also be decreased. In the algorithm loop, the parent tasks are reasonably inserted into the blank slot of the processor subject to some criteria. Therefore, the utilization of the processors can be increased, fewer processors are occupied and the expense spent on the workflows could also be saved.

Based on the current work, we intend to improve and extend the algorithm in the future to fit more situations and goals, such as load balancing, energy saving and hotspot avoiding.

Acknowledgment

This paper is partially supported by The National Natural Science Foundation of China (No. 61363019, No. 61563044 and No. 61640206); National Natural Science Foundation of Qinghai Province (No. 2014-ZJ-718, No. 2015-ZJ-725).

References

[1] Barker A, Van Hemert J. Scientific workflow: a survey and research directions [C]. Parallel processing and applied mathematics, 2007: 746-753.

[2] ZHAO Y, LI Y, TIAN W, et al. Scientific-workflow-management-as-a-service in the cloud [C]. 2012 Second International Conference on Cloud and Green Computing (CGC). Piscataway: IEEE, 2012: 97-104.

[3] DEJUN J, PIERRE G, CHI C H. EC^2 performance analysis for resource provisioning of service-oriented applications [C]. Proceedings of ICSOC/ServiceWave 2009 Workshops Service-Oriented Computing. Berlin: Springer Heidelberg, 2010: 197-207.

[4] KIM H, El-KHAMRA Y, RODERO I, et al. Autonomic management of application workflows on hybrid computing infrastructure [J]. Scientific Programming, 2011, 19(2): 75-89.

[5] DARBHA S, AGRAWAL D P. Optimal scheduling algorithm for distributed memory machines [J]. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(1): 87-95.

[6] XU Li, ZHOU Jingang, ZHANG Xia, et al. Approach of cloud application resource provision and split clustering schedule [J]. Computer Engineering, 2011, 37(11): 52-55.

[7] Agarwal A, Kumar P.E. Economical duplication based task scheduling for heterogeneous and homogeneous computing systems [C]. Advance Computing Conference, 2009: 87-93.

[8] YU Zhenxia, MENG Fang, CHEN Haining. Anefficient list scheduling algorithm of dependent task in grid [C]. Chengdu,China: 3rd IEEE International Conference on Computer Science and Information Technology, 2010: 102-106.

[9] LI Jianfeng, PENG Jian. Task scheduling algorithm-based on Improved based algorithm in cloud computing environment [J]. Journal of Computer Applications, 2011, 31 (6): 184-186.

[10] Yu J, Buyya R. A budget constrained scheduling of workflow applications on utility Grids using genetic algorithms [C]. Workflows in support of large-scale science, 2006: 1-10.

[11] Park C, Choe T. An optimal scheduling algorithm based on task duplication [J]. IEEE Transactions on Computers, 2002, 51(4): 444-448.

[12] IE Z, HAN Y, QI Y, et al. A scheduling algorithm for multi-core based on critical path and task duplication [J]. Journal of National University of Defense Technology, 2014, 36(1): 172-177. (In Chinese)

[13] Duan Ju Chen Wang-hu, Wang Run-ping, et al. Execution optimization policy of scientific workflow based on cluster aggregation under cloud environment [J]. Journal of Computer Applications, 2015, 35(6): 1580-1584.