A Priority Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing

Hu Wu*, Zhuo Tang*, Renfa Li*

*School of Computer and Communication of Hunan University, Embedded Systems & Networking Laboratory, China

276821738@qq.com, hust_tz@126.com, lirenfa@vip.sina.com

Abstract—**Cloud computing has emerged as a new paradigm for solving large-scale problems. To provide good performance in cloud computing, many algorithms have been investigated. However, there is a conflict between fairness in scheduling and user's priority. This paper proposed a novel algorithm PISA that both considered fairness and consumers' priority. The algorithm can schedule multiple workflows which are specified priority and the task weights are taken into account. Experimentation shows that our algorithm is able to increase the scheduling success rate significantly.**

***Keywords*— Cloud Computing, Scheduling, Priority, Workflow, Algorithm**

I. INTRODUCTION

As the rapid development and widely use of Internet, large-scale computing clusters are increasingly likely to be used. Many internet giant and big enterprise have investigated heavily in Cloud computing. In June, Apple introduced its latest product and service iCloud. Cloud computing is becoming the most important distributed computing paradigm, more and more researchers and developers are interested in it. Since cloud computing can use the computing resource on the internet, so it can provide a lot of resources for the needs of consumers. Many simplified cloud computing programming models have been proposed such as MapReduce. However, MapReduce was designed for small-scale problems where users' contention for computing resource can be resolved by using FIFO scheduling, which doesn't consider users' priority.

In this paper, we proposed a simply algorithm that take into account the user priority, and extends the existing FIFO algorithm. The simplest scheduling strategy is to increasing spending on workflows that are more important and drop spending on less important workflows. Importance may be implied by proximity to deadline, current demand of anticipated output or whether the application is in a test or production phase. We define the importance in this paper as workflow priority and task weight. The most important point is that the systems itself does not know which workflow or which task is the most important,, so the users should input the value that signal the workflow 's importance and task weights.

The remainder of the paper is organized as follows. Section II presents the background of MapReduce and its scheduling algorithm. Section III presents our improved scheduling algorithm. Section IV presents the experiment. Finally, Section V addresses the conclusion and the future work.

II. BACKGROUND

Scheduling in cloud environment system is NP-complete in general. Some typical workflow scheduling algorithms have been introduced. Li Wenhao in [1] introduced a community cloud oriented workflow system framework which meets the need of the process-based fast collaboration well, and a community cloud oriented task scheduling strategy. Meng Xu et al. proposed a multiple QoS constrained scheduling strategy of multiple workflows that can schedule multiple workflows which are started at any time and the QoS requirements are taken into account in [2]. Matei Zaharia presents a delay scheduling strategy that can achieve nearly optimal data locality in a variety of workloads and can increase through-put [3]. Thomas Sandholm and Kevin Lai introduced the Dynamic Priority parallel task scheduler for hadoop [6], it allows users to control their allocated capacity by adjusting their spending over time. Xiao Liu et al. proposed a probability based temporal consistency model to define the temporal violations which are statistically recoverable by light-weight exception handling strategies [7].

MapReduce is introduced by google, which is aim to resolve large-scale data-processing problems. The architecture of MapReduce includes one master node and many slave work nodes. Before user input the data, the data should be splitted and replicated in 64MB blocks. MapReduce includes two process steps, the first step is Map. When a job executes, it is divided into blocks and assign to idle nodes. The master node decides which slave idle nodes can be assigned. The second step is Reduce, it is similar to the Map step. The intermediate data, which are produced by Map step, are shuffled and transported across the reduce nodes. In this way, the data with a given key can be redirected into the same nodes. If some

tasks failed, the master node can detects the fails and reschedules the task.

Hadoop is developed from MapReduce, which is an open-source implementation of MapReduce [8]. It is composed of two parts: MapReduce and HDFS.HDFS stands for Hadoop Distributed File System. HSFS is a highly fault-tolerance system, which is can be deployed on cheap machines. The basic architecture of HDFS is master/slave, includes one Name node and some Data nodes. Since Hadoop's default scheduler's theorem like the task schedule in Operating System, once the task is scheduled, it can't be interrupted. Some scheduling algorithm have been proposed and investigated, Such as FIFO(First In First Out), RR(Round-Robin), HPF(Highest Priority First) and WRR(Weighted Round Robin). In the next section, we will introduce our scheduling algorithm PISA.

In conclusion, although the algorithms mentioned above have their respective benefits in their particular environment, none of them is designed for multiple workflows with workflow priority and task weights constrained. Thus, the motivation of our work is to design an algorithm that addresses these problems.

III. PRIORITY IMPACT SCHEDULING ALGORITHM (PISA)

In our model, we take into account the users' priority. The differences between users' priority may base on the fee they paid or something else. Next, we will first introduce the prototype algorithm, then we introduce PISA. Figure 1 shows the overview of our scheduling system.

Algorithm 1: prototype algorithm based on workflow's priority

Classify the workflow based on priority, so far, yet there is one solution that can clearly signal the importance of workflow, so the users themself should set the priority. the acquirement of priority may be based on the fee that user could pay for their workflow, for example ,a cloud computing service provider may classify the consumers as free users, VIP, et al，so the value of priority could be 0,1,2,...。If there are many workflows are competing for the cloud resources, the higher the priority of the workflow, the faster it can access to the resource. If a workflow that can't access to the resource because of its low priority level for a long time, we could set a value for the countdown. For example, the initial value can be set 0, after each round of scheduling, it plus 1, if after many rounds, if the value exceeds a fixed value, the workflow can access to the resource next round regardless of its priority. The fixed value can be calculated as follows: Assume that there are m workflows competing for the resource, and the cloud service provider define the resource as n level, so the value can be set m/n. For the workflow being executed on the node ,if its priority lower than the workflows that competing for the resource, we should wait for the completion of workflow rather than interrupt its execution, because if we interrupt the execution, it will waste the resources previously used.

Algorithm 2: PISA (Priority Impact Scheduling Algorithm)

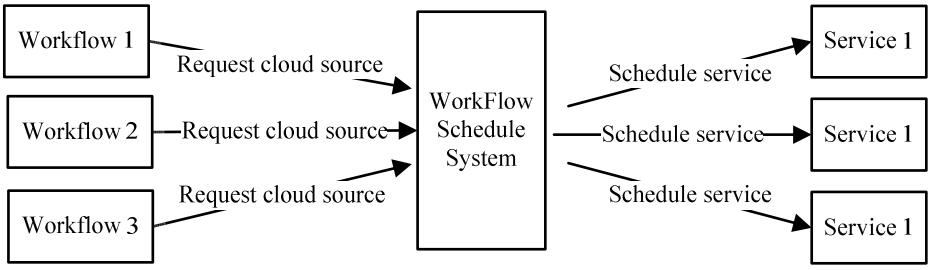

Figure 1. Overview of Scheduling Workflow.

In the second model, we introduce Access Policy Check procedure. When a workflow requests the cloud computing resource provider for services, the provider first queries the Access Strategy library. We define one policy name as WAPC, the definition of WAPC are as follow:

WAPC [(workflow_level, int priority),Task,(T_s, T_e, P)].

T_s is the start time of Task T_i, T_e is the end time of Task T_i, P is a periodic expression. The value of priority determine the highest level of cloud resource it can get .If the request level consistent with the priority value and its request meet the time quota, the application is accepted. If the value is not big enough or the time quota expires, the application is denied. We define Time quota variance T_q, It means the execution time limit of task.

$$T_q = Min(T_i) + AT(T_i)(T_e - T_s) / \sum_{i=1}^{n} T_i \qquad (1)$$

$Min(T_i)$ is the minimum time of the task T_i executed. $\sum_{i=1}^{n} T_i$ is the time variance sum of all task in the workflow. $AT(T_i)$ is time variance which the task executed on all the services. T_q is must not greater than the value T_e-T_s, otherwise it can't be accepted.

Definition 1 T_q: time quota

If the workflow is accepted by the system, it then enters the schedule sequence. The workflow composed of many tasks, for a workflow W, it has the priority value W_{pi} (the int priority in the previous).Each task in the workflow may has different weight T_{pi}, the task weight is similar to the workflow priority in the system. Then split the workflow's priority value according to user-defined weights. Because the total value is limited, if one task receives an increased share of the value, then the others get a decrease in share .The reason for this is

that different task may need different resources, some task may time sensitive or request more CPU resources, the others may not. By spend more resource on critical tasks. We can increase the overall utility of the workflow. So the task T_i's position in the scheduling sequence can calculated as follows:$W_{pi}*a+T_{pi}*b$, where *a* is called first class impact factor, *b* is called second class impact factor. Factor *a* signifies the workflow 's priority ,*b* signifies task weight.

Definition 2 ***a*: first class impact factor**

Definition 3 ***b*: second class impact factor**

The value of *a* and *b* are calculated as follows:

For a series of N workflows $W_1,W_2,...,W_n$, a user can specify their relative importance, so the weights of workflows can be $Wp_1,Wp_2,Wp_3,...,Wp_n$, the average weight *awp* is

$$awp = (\sum_{i=1}^{n} W_{pi}) / N \qquad (2)$$

For workflow W_i, difference degree is W_{pi}-*awp*, so $a=(W_{pi}-awp)/awp$.

For a workflow W_i, it includes n tasks, a user may specify the relative importance of n tasks as $T_{p1},T_{p2},T_{p3},...,T_{pn}$, W_{pi} is the sum of n tasks ,so the average weight *atp* is

$$atp = (\sum_{i=1}^{n} T_{pi}) / N \qquad (3)$$

Every task 's difference degree is T_{pi}-*atp*， so $b=(T_{pi}-atp)/atp$.

We also should take into account the time control, in every round of scheduling, if the task being executed is interrupted by some unpredictable factors, such as the shortage of electricity or some nature disaster, the task can't continue to access to the resources, then it return into the scheduling sequence, it should be given the highest priority in the next round of scheduling.

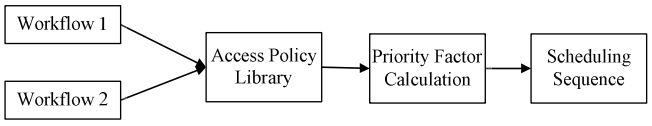

Figure 2. Overview of Scheduling Algorithm

For the task's scheduling, we also have some considerations:

1) The number of service a cloud resource provider can provide is finite
2) The task which belong to higher priority workflow should be scheduled first, that means factor a is more important than factor b, because in some case some tasks their calculated weights may big even though they may belong to lower priority workflow.

We get the algorithm shown in figure 3:

Input: (w: a workflow)

1. T： =the set of all tasks in the workflow w;
2. T_q: =time quote
3. T_s： =the execute start time of task T 's resource.
4. T_e: =the end access of resource time point.
5. W_{pi}: =the priority of Workflow W_i
6. T_{pi}: =the priority of Task T_i in W_i
7. $\sum_{i=1}^{n} T_{pi} = W_{pi}$
8. **AccessPolicyCheck**() {
9. **For** each request task T_i **do**
10. **If** T_i's request qualify for the priority and Its T_q is less than T_e-T_s
11. **Then** place the Ti in the sequence
12. }
13. **SelectTaskToRun**() {
14. PriorityImpactFactor PIF: = a*W_{pi} +b*T_{pi}
15. TaskToRun: =null;
16. **While** T ⊄ F **do**
17. **For** each T_i ∈ T-F **do**
18. Calculate the PIF of T_i
19. **If** T_i 's PIF is the biggest in the queue or TimecountValue >= MaxWaitValue
20. **Then** select t as the taskToRun
21. **End**
22. **Else**
23. T_i remain in the queue to wait until the next round
24. TimecountValue+1
25. **Return** wait;
26. **End**
27. **Return** done;
28. }
29. **Exec()**
30. {
31. **While** there is an idle Hadoop node **do**
32. **Switch SelectTaskToRun**() **do**
33. **Case** done: **return**;
34. **Case** wait: sleep(some time);
35. **Case** task T_i to run:
36. Submit T_i to Hadoop node;
37. **End**
38. **End**
39. }

Figure 3. PISA

IV. EXPERIMENTS AND RESULTS

To evaluate the performance of the priority impact scheduling algorithm (PISA), we have conducted an experiment. Our simulate cloud environment has 30 services. Every service can execute one task at the same time. There are 10 users to request the use of the cloud service to execute their workflows. The 10 users have different priority level, so the workflows also have different priority value. The values of all the workflows are shown in Table 1.Then the users specify the

weights of different tasks of their own workflow as shown in Table 2. Each workflow has 3 tasks. In FIFO scheduling algorithm, users don't specify the task's weight. From the Table 3 we can find that PISA can meet the priority requirement of different users, the most important workflows and tasks can get the resource firstly, so they can be executed successfully and more quickly.

TABLE I. WORKFLOW PRIORITY

workflow	Priority Level	FIFO sequence(random)
Workflow1	10	2
Workflow2	9.5	10
Workflow3	9	3
Workflow4	8.5	5
Workflow5	8	8
Workflow6	7.5	1
Workflow7	7	4
Workflow8	6.5	7
Workflow9	6	6
Workflow10	5.5	9

TABLE II. TASK WEIGHT

Workflow	Task 1 weight	Task2 weight	Task 3 weight
Workflow1	3	5	2
Workflow2	2.5	6	0.5
Workflow3	1	6	2
Workflow4	1.5	4	3
Workflow5	2	4	2
Workflow6	2.5	1	4
Workflow7	1	5	1
Workflow8	3.5	2	1
Workflow9	3	1	2
Workflow10	0.5	4	1

TABLE III. SCHEDULING SUCCESS RATE

	PISA	FIFO
Success Rate	70.0%	43.3%

As a result, PISA has a higher success rate than FIFO .The reason is that PISA takes into account workflow priority and task weight while FIFO only considers the arrive sequence of workflows. In this experiment PISA has the average success improvement rate over FIFO by 26.7% when we input 30 workflows. We define the average success rate as the successful execution ratio of high priority workflows. When the total number of concurrent workflows increases from 10 to 30, we find that the success improvement rate are also climbing, we think the reason is that the more workflows executed the more obvious of PISA's efficiency.

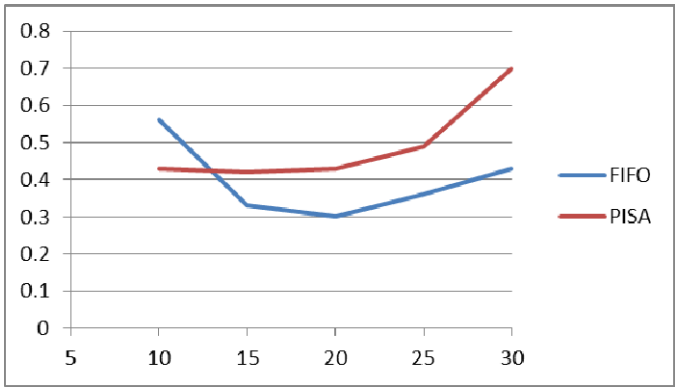

Figure 4. Success Rate of Scheduling

V. CONCLUSION AND FUTURE WORK

The workflow of different users may have different priority constraint. It is a main challenge of cloud workflow system to scheduling workflow that both considered workflow priorities and task weights. However, most existing scheduling algorithms are not designed to address these problems.

To address this problem, we proposed PISA scheduling strategy of multi workflows for cloud computing. The experiment has demonstrated that it can produce better performance than FIFO.

In the future work, we plan to import RBAC model to control the user access policies [4], [5]. And we will study how to improve the algorithm by adding more constraints.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China(61103047), National Postdoctor Science Foundation of China(20100480936), Hunan Natural Science Foundation(11JJ4052).

REFERENCES

[1] Li Wenhao,"A Community Cloud Oriented Workflow System Framework and its Scheduling Strategy",978-1-4244-6359-6/10 2010 IEEE

[2] Meng Xu,Lizhen Cui,Haiyang wang,Yanbing Bi,"A multiple QoS constrained Scheduling Strategy of Mutiple Workflows for Cloud Computing",2009 IEEE International Sympsium on Parallel and Distributed Processing with Applications

[3] Matei Zaharia et al."Delay Scheduling:A Simple Technique for Achieving Locality and Fainess in Cluster Scheduling",EuroSys'10

[4] Ravi S.Sandhu,Edward J.Coyne,Hal L.Feinstein and Charles E.Youman,"Role-Based Access Control Models",IEEE computer,Volume 29,Number 2,February 1996,pages 38-47.

[5] James B.D. Joshi,Elisa Bertino,Usman Latif,Arif Ghafoor,"A Generalised Temporal Role-Based Access Control Model",IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING.

[6] Thomas Sandholm,Kevin Lai,"Dynamic Proportional Share Scheduling in Hadoop",JSSPP 2010,LNCS 6253,pp.110-131 2010

[7] Xiao Liu,JinJun Chen,Zhangjun Wu,Zhiwei Ni,Dong Yuan,Yun Yang,"Handling Recoverable Temporal Violations in Scientific Workflow System:A Workflow Rescheduling Based Strategy",2010 10th IEEE/ACM International Conference on Cluster,Cloud and Grid Computing.

[8] Hadoop.http://hadoop.apache.org/