Vol.2010-AL-128 No.8
2010/1/26

A Flexible User-centric Resource Scheduling Algorithm

Laiping Zhao[†] and Kouichi Sakurai[††]

In service-oriented Grid environment, user QoS requirements are important that it is directly related to the popularization utilization of Grid. This paper proposes a innovative resource searching and scheduling mechanism, called SPSE. Our idea, inspired from web search engine, is to sort all the service providers from internet, and return a service providers list back to end-user. Compared with existing resource scheduling algorithms, our method is much more flexible in satisfying different kinds of user requirements, which also performs well in scalability, and supports multi-objective and user personalization. We also design a simulation prototype for our method, proving that we can capture user's preferences value precisely and automatically

1. Introduction

1.1 Background

New technologies, such as Grid computing, Cloud computing, Ubiquitous computing are developing quickly currently, generating a variety of web services in internet environment. These geographically distributed heterogeneous services perform differently in service quality. How to select the most appropriate service for an end-user is quite a challenge for scheduling algorithm. In other words, existing resource scheduling algorithms need improving to suit modern distributed environments.

1.2 Motivation

We focus on the user centric scheduling algorithms in this paper. In our real life, different people always have different QoS interests, it is impossible to take into account all kinds of users' requirements by one scheduling method in a large-scale network environment. And no generic scheduling algorithms have ever been presented yet. To address this problem, present research mostly focus on some certain sub-topics, for example, most of the existing scheduling approaches only optimize execution time. Along with the economic model [2] introduced into Grid, economic cost that an application needs to pay for resources utilization becomes a concern of some users. Trust degree and reliability of the service provider are introduced into Grid in paper [3,4,5]. Besides these, many research works have been done on multi-object scheduling problem. In this paper, We try to give a new scheduling method from a different view, and satisfy different kinds of requirements as much as possible.

1.3 Previous work

Resource scheduling is always an important and hot topic in Grid research. Many research work have been done on it[1][6], such as: Min-min, Max-min, suffrage, Xsuffrage, FirstPrice and FirstReward, which are all simple heuristics that usually applied to make just-in-time scheduling. These heuristics provide a basic scheduling method for resource scheduling, but obviously are not enough for current fast developing heterogeneous computing environment. Along with the requirements of user QoS satisfaction or system performance rising, more advanced scheduling algorithms are needed.

Considering the time and cost factors, DBC scheduling algorithm [7], HRED heuristic [8] and Yu et al. [9] [10] are give a scheduling solution on in Grid. obviously, these algorithms are far from enough for a user-centric scheduling problem, which may concern many different kinds of scheduling criteria. Jack J. Dongarra[3]et al. take running time and reliability as scheduling criteria, propose a bi-objective scheduling algorithm. Paper [4][5] also take reliability as the scheduling criteria. Wieczorek et al. [11] propose a general bi-criteria scheduling heuristic called Dynamic Constraint Algorithm (DCA). They propose a requirement specification method based on the sliding constraint, which expects the user define which criterion should be the primary criterion, and which one should be the secondary criterion.

Paper [10][12][13] make use of genetic algorithm to assign jobs to resources under QoS constraints: time and cost. Ruay S. C.et al [14] use ant colony algorithm to balance the entire system load while minimizing the makespan of a given set of jobs. The complexity of GA and Ant colony based scheduling algorithms makes themselves unscalability for Grid environment.

1.4 Challenging issues

Generally, a user may have multi-QoS requirements, including: execution time, economic cost, trust degree, reliability, and any other related criteria. However, those existing scheduling algorithms have a lot of limitations in satisfying all these requirements, because they only consider one or two criteria as the scheduling objective. In addition, even on the same criteria, two persons hardly have the complete same preference degree. User preferences on different criteria should be calculated precisely too.

[†] Department of Informatics, Kyushu University
PH.D. candidate, Email: zlp@itslab.csce.kyushu-u.ac.jp
[††] Department of Informatics, Kyushu University
Professor, Email: sakurai@csce.kyushu-u.ac.jp

1.5 Our contribution with comparing to related works

In this paper, we will address the resource scheduling problem based on above four crucial requirements (Scalability, Flexibility, Multi-objective supported, User personalization supported) from a different perspective and give a novel resource scheduling method in the service oriented Grid environment. Inspired by the web search engine in internet, like Yahoo[16],Google[17], we design our own "Service Providers Search Engine" (SPSE): people submit their job description to our SPSE, and SPSE return a list of service provider candidates. Then user select one solution from the list, his job will be executed on the selected resource. Our SPSE has four advantages: Firstly, SPSE can be scalable deployed in large scale Grid environment, supporting large number of users and services. Secondly, SPSE is flexible and generic because it can be employed by different kinds of people, who have different interests on the services. Thirdly, SPSE supports multi-objectives scheduling, including time, economic cost, trust degree etc. and can be extended to support more criteria easily. Finally, SPSE can calculate every user's preferences precisely and automatically, which means our algorithm is a user personalization supported mechanism.

2. System Model and Problem Statement

2.1 Job model

Job is represented like this:

$$Job = \{User\ id, Job\ id, Instructions, service\ type\} \quad [1]$$

Where $User\ id$ is from the job's owner; $Job\ id$ uniquely identifies the job; $Instructions$ means the amount of computation; and $service\ type$ indicates which kind of service this job needs.

Jobs are real-time, and non-preemptive. Non-preemptive means jobs cannot be interrupted during execution and must finish to completion. At one time, only one task can execute on a resource, and each task cannot be divided further for parallel processing, and thus must be scheduled in its entirety on a processor.

2.2 Resource Model

Resource is represented like this:

$$Resource = \{Resource\ id,\ \ service\ type, cpu, price, trust\ degree, \ldots (other\ criteria)\} \quad [2]$$

Where $Resource\ id$ uniquely identifies the job; $service\ type$ indicates which kind of service this resource will provide; cpu represents its cpu capability; $price$ is how much user have to pay for resource utilization per second; $trust\ degree$ represents resource's reliability; we user the $other\ criteria$ field to extend SPSE to support more criteria.

2.3 SPSE Operations

SPSE mechanism consists of several sub-algorithms. We define some operations and conceptions before giving the mechanism details:

Definition 1: Service providers list (SPL)

SPL_i lists all the service providers that provide the same service for job_i. SPL_i has 2 attributes:

(1) All the service providers in SPL_i provide the same service type.
(2) Different service providers may show different performance.

Definition 2: Solutions list (SoL)

SoL_i is also a list of service providers that provider the same service for job_i. It has relation and difference with SPL_i:

(1) SoL_i is the service providers list for job_i that will be shown to end user, while SPL_i is just all the service providers for job_i.
(2) SoL_i is generated by filtering SPL_i, which means its size is not bigger than size of SPL_i.
(3) Both SoL_i and SPL_i contain the service provider SP_i who will execute job_i finally.

Definition 3: Ranked solutions list ($RSoL$)

$RSoL_i$ is gotten by ranking all service providers in SoL_i.

(1) $RSoL_i$ contains exactly the same service providers with SoL_i.
(2) Service providers in $RSoL_i$ is ranking in order by certain sorting algorithm.
(3) $RSoL_i$ is presented to end user in sequence.

Operation 1: $SPL_i \leftarrow Search(t_i)$

$Search(t_i)$ operation searches service providers from Grid information services (GIS) for a job_i, and return a SPL_i.

Operation 2: $SoL_i \leftarrow Filter(SPL_i)$

$Filter(SPL_i)$ operation sifts out the poor service providers from SPL_i, and leave behind good service providers SoL_i.

Operation 3: $RSoL_i \leftarrow Rank(SoL_i)$

After receiving SoL_i from $Filter(SPL_i)$ operation, SoL_i should be re-ranked again into $RSoL_i$ using $Rank(SoL_i)$ operation: let good solutions rank higher than poor solutions.

Operation 4: $UP \leftarrow Update(UP)$

User will select one solution to execute job. Based on the selected option and $RSoL_i$, $Update(UP)$ operation is in charge of calculating user's preferences.

3. SPSE Scheduling Algorithm Design

3.1 SPSE Overview

The SPSE mechanism is illustrated in Figure 1. In Grid environment, all service providers

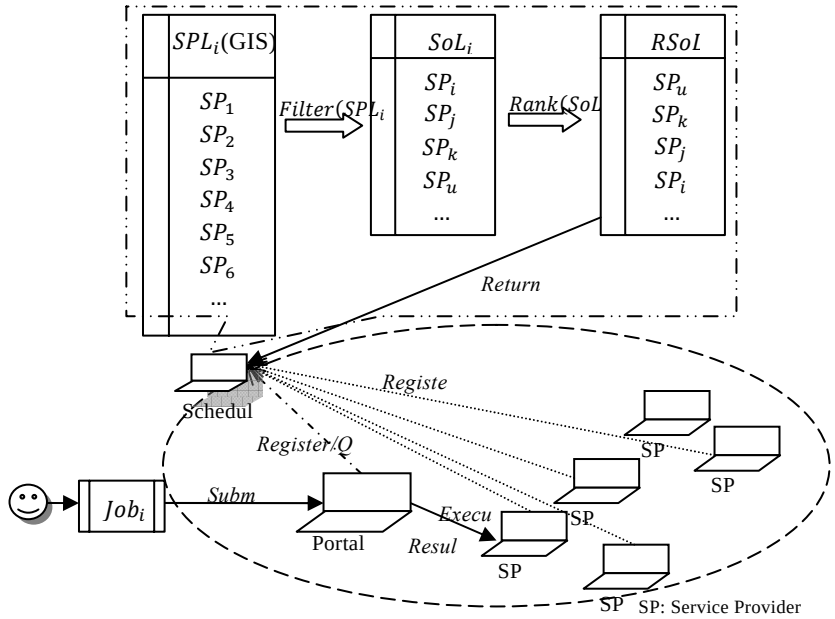


Figure 1: Resource scheduling model

are registered into GIS. When a new service provider joins Grid, its service type, resource performance (e.g. cpu, memory) and some other related attributes are registered into GIS using resource specification language (e.g. RSL). Table 1 shows the details of SPSE: end-user submits his job description to Grid portal, then the Grid portal communicate with scheduler to search service providers. After screening out poorer solutions and ranking the better solutions, a solutions list is returned back to end-user. End-user chooses one solution to execute his job. Then the job is submitted to remote service provider, and update user preferences.

The next, we give the detailed discussion on operations: $Search(t_i)$, $Filter(SPL_i)$, $Rank(SoL_i)$ and $Update(UP)$.

3.2 $Search(t_i)$

Operation $Search(t_i)$ travel through all the registered service providers (e.g. MDS), search for service providers that must satisfy two conditions:

(1) Its service type is right the same with job requirement's service type.

(2) The service provider is available right now.

Table 1 The overview of SPSE

SPSE mechanism
Step 1: End-user submits his job description Job_i to a Grid portal (e.g. Web portal).
Step 2: Web portal receives Job_i, and queries GIS according to the service type. $Search(t_i)$ operation returns back SPL_i, which lists all available service providers for Job_i.
Step 3: $Filter(SPL_i)$ operation sifts the poor service providers from SPL_i, returns SoL_i back.
Step 4: $Rank(SoL_i)$ rank all solutions in SoL_i, and get $RSoL_i$.
Step 5: $RSoL_i$ is present to user. User selects one solution as his choice.
Step 6: Job execution: Job is transferred to selected resource, which will take in charge of the job execution.
Step 7: According to user's choice, calculate and update user's preference on different criteria. $Update(UP)$ will take care of this.
Step 8: After task execution completes, return results back to portal.

SPL_i is generated after this operation. The time complexity of $Search(t_i)$ is $O(N)$, where N is the number of all kinds of resource providers .

3.3 $Filter(SPL_i)$

Assuming different service providers always show different performances, and users are rational, compared with slow-running, expensive, unreliable providers, users prefer fast-running, cheap, reliable service providers. Above all, bad service providers whose performance is weak will not be involved in the scheduling process. To improve the scheduling efficiency, we employ a Pareto optimal-based service providers selection method, leaving behind good service providers, while screening out bad service providers.

An example 1: As shown in figure 2, considering time and economic cost as scheduling objectives, there are 10 service providers for one job: A,B,C,D,E,F,G,H,I,J. Their performances on time and economic cost are shown in the coordinates. For example, point $A(1,10)$ means that, service provider A could complete this task in 1 second, charge 10 dollars. If compare service provider D with F, D spends less money than F for $2 < 4$, and spends less time than F for $6.5 < 7$. Meanwhile, there are no other service providers that spend less time and less money than D. Therefore, D is one non-dominated solution [19]. Similarly, service provider: A,B,C,G are all non-dominated solutions. As a result, the final Pareto optimal solution set is $\{A,B,C,D,G\}$. These 5 service providers will take part in the following scheduling steps. The filter algorithm [20] is shown in Tab. 3.2, and its maximum time complexity is $O(MN^2)$, where M is the number of criteria, and N is the number of service providers in SPL_i.

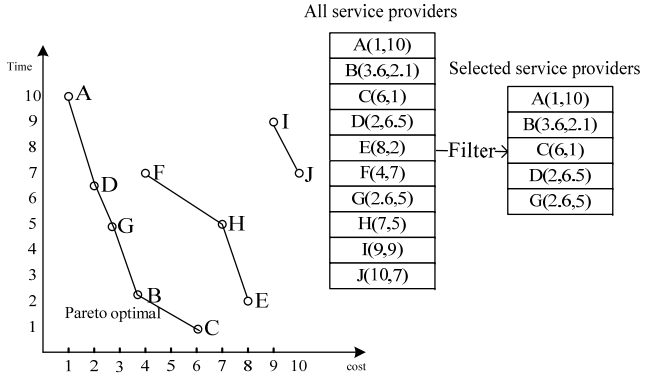


Figure 2: Filter service providers

Theorem 1: *After* $Filter(SPL_i)$*, the time minimization solution (cost minimization solution, trust maximization solution) is still kept.*

Proof: Suppose the time minimization solution sol_j is sifted out by $Filter(SPL_i)$ operation. Then we know that $sol_j$is a dominated solution. According to the definition of "dominated"[19], we know that there exist a solution, suppose it is sol_k, whose execution time is shorter, economic cost is lower, and trust degree is higher. Because sol_j is the time minimization solution, then contradict sol_k. Therefore, the assumption is incorrect, and so the theorem.

Through theorem 1, we know that SPSE can be used for single objective scheduling too.

3.4 $Rank(SoL_i)$

SPSE support multi-objective scheduling: execution time, economic cost, and trust degree are the most popular scheduling criteria in Grid environment. However, SPSE not only support these criteria but also can be extended to support more criteria easily.

$Rank(SoL_i)$ operation takes in charge of ranking all these solutions into certain order, where solutions with shorter time, less economic cost, higher trust degree rank ahead of solutions with longer time, more economic cost, and lower trust degree. Moreover, the most attractive solution for an end-user should be ranked at the top of the list. Before design the ranking algorithm, firstly we introduce the user preferences definition.

Definition 4:

User preferences (UP) are a set of parameters-$\{p_1, p_2, p_3, \dots, p_m\}$, where m is the number of criteria. Each parameter reflects how highly user values the corresponding criteria. UP has five attributes:

(1) Every user x has his own set of preferences: $UP_x = \{p_1^x, p_2^x, p_3^x, \dots, p_m^x\}$.
(2) Initialization: $p_1^x = p_2^x = p_3^x = \cdots = p_m^x = 1$.
(3) Suppose two users: x and y, then there are no relationship between $\{p_1^x, p_2^x, p_3^x, \dots, p_m^x\}$ and $\{p_1^y, p_2^y, p_3^y, \dots, p_m^y\}$.
(4) There are no relationship between two parameters: p_i and p_j.
(5) UP_x will be updated every time after a scheduling process.

Based on criteria (time, economic cost, and trust degree) and user preferences, we give a fuzzy ranking method to sort all the solutions. First we give a solution struct design that will be used both in analysis and implementation, and then we will introduce our algorithm with example 2.

Solution ID	Time rank	Cost rank	Trust rank	(Reserved)	Final rank

Figure 3: Ranking Struct

The solution struct consists of five parts: solutions ID, time rank, cost rank, trust rank, research space, and final rank. Solutions ID uniquely identifies a solution; Time rank space keep the solution's sorting number after sorting all the solutions according to the execution time cost; Cost ranking space and trust rank space keep the cost sorting number and trust degree sorting number respectively; Reserved space is used to support extending more criteria; Final rank keeps the computing final rank value, which is computed by formula:

$final\ rank = time\ rank \times p_1^x + cost\ rank \times p_2^x + trust\ rank \times p_3^x + (reserved \times p_i^x)$ [3]

where p_1^x , p_2^x, p_3^x, p_i^x are user preference value on time, economic cost, trust degree, and the future extended criteria respectively.

An example 2: As shown in Figure 4, after $Filter(SPL_i)$ operation, there are 5 solutions left in SoL_i: S_1, S_2, S_3, S_4, S_5. Sorting all these 5 solutions in time ascending order, we get: $S_2 < S_5 < S_4 < S_3 < S_1$; in cost ascending order, we get: $S_1 < S_3 < S_5 < S_2 < S_4$; and in trust degree descending order, we get: $S_1 > S_4 > S_2 > S_5 > S_3$. Based on the sorting number and user preference, we get the final rank value using formula 3: $final\ rank_{s_1} = 5 \times 1 + 1 \times 1 + 1 \times 1 = 7$, $final\ rank_{s_2} = 1 \times 1 + 4 \times 1 + 3 \times 1 = 8$, $final\ rank_{s_3} = 4 \times 1 + 2 \times 1 + 5 \times 1 = 11$, $final\ rank_{s_4} = 3 \times 1 + 5 \times 1 + 2 \times 1 = 10$, $final\ rank_{s_5} = 2 \times 1 + 3 \times 1 + 4 \times 1 = 9$.

Having gotten the all final rank values, sorting all solutions again according to final rank value, we get the final sequence list: $S_1 > S_2 > S_5 > S_4 > S_3$, which will be shown to the end user. Table 2 gives the solutions ranking algorithm. First, sort all the solutions depend on

different scheduling criteria, and save the ranking number into struct (Lines 1-3). Then

Tab. 2 $Rank(SoL_i)$ algorithm

Input：SoL_i
1: for each criteria
2: sort(SoL_i);
3: save_ranking_number();
4: for(each solution in Solutions[])
5: calculate_final_rank (solution);
6: sort_final_rank (solutions);
Output: $RSoL_i$

calculate the final ranking number using formula 3(Lines 4-5). At last, ranking all solutions based on the final ranking number. If we user quick sort, the time complexity of lines 1-3 is $O(MN(log\,N))$ (where M is the number of criteria, and N is the number of solutions), and the time complexity of lines 4-5 is $O(N)$, and the last sort_final_rankings() function's time complexity is $O(N\,log\,N)$. Therefore, the overall time complexity is: $O(MN(log\,N))$.

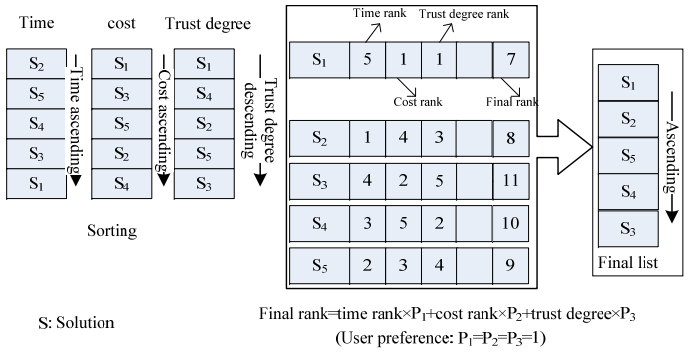


Figure 4: Ranking solutions

3.5 $Update(UP)$

As defined in last section, every user x has his own set of preferences: $UP_x = \{p_1^x, p_2^x, p_3^x, \dots, p_m^x\}$. How to calculate the preferences value is quite a challenge. Existing work have ever used UP to describe user preference to different criteria, but not calculate it automatically, they let the end user set the value artificially [11]. Obviously, it is difficult for a user to set an accurate value describing his preferences. SPSE returns to user a list of solutions candidates. What user needs to do is just selecting one solution to execute his job. User's choice is very good material for capturing the exact user's preference value. Therefore, we update UP based on this user's choice.

Formula 4,5,6,7 are given to update the UP:

$$p_1' = p_1 \times \frac{\overline{t_{top}} - t_{user}}{\overline{t_{top}}} \qquad [4]$$

From the formula 4, we get that:

(1) If $\overline{t_{top}} = t_{user}$, then $p_1' = p_1$;
(2) If $\overline{t_{top}} > t_{user}$, then $p_1' > p_1$;
(3) If $\overline{t_{top}} < t_{user}$, then $p_1' < p_1$;

where t_{user} is the execution time of user selected solution, p_1 is the user preference value on execution time, p_1' is the updated value of p_1, and $\overline{t_{top}} = \frac{\sum_{i=1}^{n_{top}} t_i}{n_{top}}$ (where n_{top} is the number of solutions above the user selected solution).

$$p_2' = p_2 \times \left(1 + \frac{\overline{c_{top}} - c_{user}}{\overline{c_{top}}}\right) \qquad [5]$$

$$p_3' = p_3 \times \left(1 + \frac{r_{user} - \overline{r_{top}}}{\overline{r_{top}}}\right) \qquad [6]$$

$$p_m' = p_m \times \left(1 + \frac{\mp e_{user} \pm \overline{e_{top}}}{\overline{e_{top}}}\right) \qquad [7]$$

Because for trust degree, bigger is better, which is different from time and economic cost. So here is a little different with the formula 3-3.

After having gotten p_i', some rules should be followed:

(1) If $p_i' < 0$, then set $p_i' = 0$;
(2) Reward factor:

If($p_1 < threshold$ && $\frac{\overline{t_{top}} - t_{user}}{\overline{t_{top}}} > threshold$) then set $p_1' = 1$.

If($p_2 < threshold$ && $\frac{\overline{c_{top}} - c_{user}}{\overline{c_{top}}} > threshold$) then set $p_2' = 1$.

If($p_3 < threshold$ && $\frac{r_{user} - \overline{r_{top}}}{\overline{r_{top}}} > threshold$) then set $p_3' = 1$.

If($p_m < threshold$ && $\frac{\mp e_{user} \pm \overline{e_{top}}}{\overline{e_{top}}} > threshold$) then set $p_m' = 1$.

Where $threshold$ indicates the degree of changes to user's preferences. If preference on one criteria is improved obviously, then reward principle will be employed and this preference value will be set to 1 directly. We choose $threshold = 0.2$ in our experiments.

4. Analysis and Experiments

4.1 The SPSE prototype

We design and implement a simulation prototype for SPSE. In the prototype, jobs and resources are all initialized by random numbers. For a $job_i = \{User\ id_i, job\ id_i, Instructions_i, service\ type_i\}$, the instructions is randomized in $Instructions_i \in [0, 10000]$, $service\ type_i \in [1,9]$. For a $Resource_j = \{Resource\ id_j,$, $service\ type_j, cpu_j, price_j, trust\ degree_j\}, service\ type_i \in [1,9]$, $cpu_j \in [20,100]$,

```
############ SPSE ############
------------------------------
Waiting for job...
User 1 is ready, creating job...
Job ID: 6, Requested service type: 7, Calculation amount: 3763
------------------------------------------------------------
Submitting the job 6 to the scheduler...
------------------------------------------------------------
The job can be completed by these solutions:
------------------------------------------------------------
Solution: 228:
Time: 38, Cost: 1950, Trust: 0.77
Time rank: 10, Cost rank: 7, Trust rank:8
Overall rank: 27.9999
------------------------------------------------------------
------------------------------------------------------------
Solution: 831:
Time: 150, Cost: 1500, Trust: 0.82
Time rank: 21, Cost rank: 4, Trust rank:6
Overall rank: 28.6315
------------------------------------------------------------
------------------------------------------------------------
Solution: 583:
Time: 129, Cost: 1500, Trust: 0.68
Time rank: 18, Cost rank: 1, Trust rank:11
Overall rank: 28.7719
```

Figure 5 SPSE Prototype

$price_j \in [10,20]$, $trust\ degree_j$ is a decimal in [0,1]. SPSE prototype is illustrated in Figure 5. After user submits his job description, a list of solutions is returned. What user should do is select one from the list, then his job will be executed on selected resource.

4.1 Experiments

Three performance metrics are introduced to evaluate the SPSE prototype:

(1) *Scalability:*

From above descriptions, we know that the time complexity of each sub-algorithm in SPSE method is: $O(N)$, $O(MN^2)$, $O(MN(log\ N))$, so the overall time complexity is $O(MN^2)$. We evaluate the scheduling efficiency of SPSE by our prototype; the execution time of

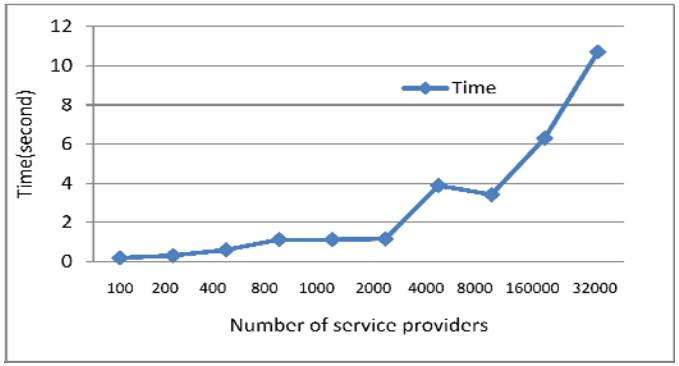
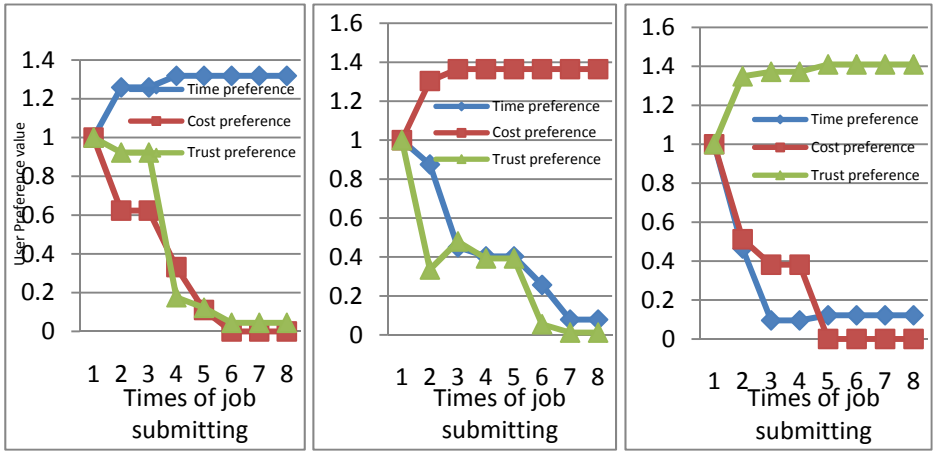


Figure 6 Scheduling efficiency

scheduling process is illustrated in Figure 6. We can see that, along with the number of service providers increasing from 100 to 2000, the execution time is much shorter, even there are 8000 service providers, the time is no longer that 4 seconds. If SPSE is implemented by parallel algorithm, like Google search engine, we believe the scheduling time will be much shorter.

(2) *Precision of preference values:*

The second experiment concentrates on updating user preferences value. First, let three end-user participant in the SPSE scheduling process, where the first user only care about job execution time, the second user only care about the economic cost, and the third one only care about service provider's trust degree. After 5 times of job submitting, we see that (Fig. 7) user preferences of the first user have become stable, where preference value on execution time is around 1.4, and the preference value on economic cost and trust degree is almost 0. The same situation happens to the second and the third user, where with the preference values on



a b c

Figure 7. The Precision of preference values. a: *UP* values of a user who prefers minimum execution time; b: *UP* values of a user who prefers minimum economic cost; c: *UP* values of a user who prefers maximum trust degree.

economic cost and trust degree up to 1.4 respectively, the preference values on the other two criteria are much lower, which are around 0.

If the first user changes his interest from time minimization to cost minimization, and then changes to trust maximization again. From Figure 8, we can see that, after the sixth job submitting, he change his preference from time to cost, his preference value on time and trust degree continue to decline after the 6th job submitting, while the preference value on cost rises up suddenly. Then after 18th job submitting, user changes his interests again, his preference value on trust degree rises up, and preference on time and cost decline rapidly. This experiment proves that SPSE can capture user's preferences value quickly and precisely, even user changes his preferences.

(3) *Precision of solutions:*

If submit the same job again and again, from Figure 9, we know that user preferences values of the first user is no longer changed after the second job submitting, which means if the job and Grid environment is the same, then user preferred solutions will rank at the topmost from the second job submitting. User preferences values of the second user and the third user show the same situation. This experiment proves that SPSE can rank the user preferred solution at the topmost quickly, and capture the user preference precisely.

Fig. 8 The user preference value when user changes his interests

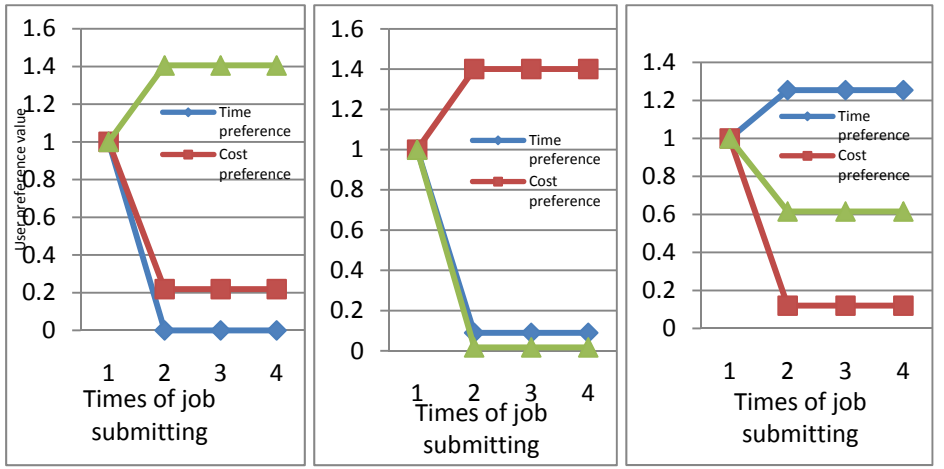


Fig. 9 User preferences of the same job submitting

5. Conclusion and Future Work

User QoS satisfaction is vital for the popularization utilization of Grid. In this paper, we research the resource scheduling algorithm in Grid environment, and find that there are still many limitations in current scheduling methods to satisfy different user QoS requirements. Based on this, we propose the SPSE scheduling mechanism, which is flexible enough for supporting user requirements, and can calculate user preferences precisely and automatically. The SPSE scheduling mechanism is scalability to be deployed into large-scale Grid environment, and it also support multiobjective scheduling.

We implement a simulation prototype for SPSE, evaluating its performance from three aspects: scalability, precision of preference values and precision of solutions. The experiments show that SPSE works well even thousands of service providers existed in the Grid environment. If designed into distributed algorithm, we assume that the scheduling efficiency would be much higher. This is one part of our next work. we also prove that SPSE can sort all the solutions in order accurately according to *UP*. *UP* value will become stable after several jobs submitting, and the most preferred solution will rank at the topmost in the list, even user changes his preferences, the preference value can be recalculated to support the new situation.

As the dynamics and complexity of the Grid environment, the next, we will continue to improve the SPSE scheduling method by taking into account two aspects: the fault tolerance and preemptible in job scheduling.

Acknowledgment The first author of this research is supported by the governmental scholarship from China Scholarship Council. The authors would like to thank Dr. Weifeng Sun, who gives many constructive comments and suggestions to this paper.

Reference

[1] F. Dong, S.G. Akl, Scheduling Algorithms for Grid Computing: state of the Art and Open Problems, Technical report No. 2006-504, (2006)

[2] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic Models for Resource Management and Scheduling in Grid Computing. J. of Concurrency and Computation: Practice and Experience. vol.14, pp. 1507-1542, (2002)

[3] J.J. Dongarra, E. Jeannot, E. Saule, Z. Shi, Bi-objective Scheduling Algorithms for Optimizing makespan and Reliability on Heterogeneous Systems. In: Proceedings of the 19th annual ACM symposium on Parallel algorithms and architectures, ACM Press, San Diego, pp. 280-288, (2007)

[4] X. Qin, H. Jiang, A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. Parallel Computing. vol. 32, pp.331-356, (2006)

[5] A. Dogan, F. Ozguner, Biobjective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems. The Computer Journal. vol.48, pp.300-314, (2005)

[6] M. Wieczorek, A. Hoheisel, R. Prodan, Towards a general model of the multi-criteria workflow scheduling on the grid. Future Generation Computer Systems. vol.25, pp.237-256, (2009)

[7] R. Buyya, M.Murshed, D. Abramson, S. Venugopal: IScheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm. International Journal of Software: Practice and Experience. vol.35, pp.491 - 512, (2005)

[8] S. Kumar, K. Dutta, V. Mookerjee, Maximizing business value by optimal assignment of jobs to resources in Grid computing. European Journal of Operational Research. vol.194, pp.856-872, (2009)

[9] J. Yu, R. Buyya, C.K. Tham: Cost-based Scheduling of Scientific Workflow Applications on Utility Grids. In: Proceedings of the 1st International Conference on e-Science and Grid Computing, IEEE Computer Society, Melbourne, pp.140 - 147, (2005)

[10] J. Yu, R. Buyya: A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms. In: Proceedings of Workshop on Workflows in Support of Large-Scale Science, Paris, (2006)

[11] M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of Scientific Workflows in the ASKALON Grid Environment. ACM SIGMOD Record, vol.34, pp. 56-62, (2005)

[12] S. Garg, P. Konugurthi, R. Buyya, A Linear Programming Driven Genetic Algorithm for Meta-Scheduling on Utility Grids. In: 16th International Conference onAdvanced Computing and Communications, IEEE Press, Chennai, pp. 19-26, (2008)

[13] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming, vol. 14, pp.217-230, (2006)

[14] R.S. Chang, J.S. Chang, P.S. Lin, An ant algorithm for balanced job scheduling in grids. Future Generation Computer Systems, vol. 25, pp.20-27, (2009)

[15] E. Zitzler, M. Laumanns, S. Bleuler, A Tutorial on Evolutionary Multiobjective optimization. Springer-Verlag, Berlin, (2003)

[16] Yahoo website, http://www.yahoo.com/

[17] Google website, http://www.google.com/

[18] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, vol. 6, pp.182-197, (2000)