SPSE: A Flexible QoS-based Service Scheduling Algorithm for Service-Oriented Grid

Laiping Zhao*, Yizhi Ren*†, Mingchu Li†, and Kouichi Sakurai‡
* *Department of Informatics*
Kyushu University, Fukuoka, Japan
Email: {zlp,ren}@itslab.csce.kyushu-u.ac.jp
† *School of Software*
Dalian University of Technology, Dalian, China
Email: li_mingchu@yahoo.com
‡*Department of Informatics*
Kyushu University, Fukuoka, Japan
Email: sakurai@inf.kyushu-u.ac.jp

***Abstract*—With the development of the Grid computing, increased attention is paid to services and user personalization. How to search and schedule the most suitable service for an end user direct affects the popularization use of service oriented Grid. Inspired from the mode of web search engine, such as Yahoo, Google, this paper proposes an innovative service searching and scheduling algorithm (SPSE: Service Providers Search Engine) for the Grid. The SPSE sorts all services from Internet and returns the most appropriate ones to the end user. Compared with the existing scheduling algorithms, our method is much more flexible in meeting user's QoS requirements, especially supporting the multiobjective and user personalization. The related simulation experiments show that our method performs well in scalability, and can capture user's preferences value precisely and automatically.**

***Keywords*-QoS; Service-oriented Grid; Scheduling; Fuzzy ranking; Pareto optimal**

I. Introduction

A. Background

The emerging new technologies, such as Grid computing, Cloud computing, Ubiquitous computing, are developing quickly, producing various web services in the Internet environment. These geographically distributed heterogeneous services perform differently on service quality. How to select the most suitable service for an end user is a challenge.

The scheduling algorithm can be categorized into the user oriented algorithms and the system oriented algorithms[1]. User oriented algorithms aim to optimize each independent application's performance, such as execution time, economic cost. System oriented algorithms aim to optimize the system's performance, such as capacity, availability. Because the QoS satisfaction direct affects the popularization use of service oriented Grid, this paper concentrates on the user oriented scheduling algorithm.

B. Motivation

Many existing algorithms consider one scheduling objective, for example: execution time, economic cost[2], and trust degree [3][4][5]. Besides these, some research works base on multiobjective scheduling problem. However, people always have different QoS requirements, and not many research have been conducted on satisfying the different requirements. Moreover, no user personalization-supported scheduling algorithms have ever been proposed in service-oriented Grid.

C. Previous work

To resolve the QoS-based scheduling problem in service oriented Grid, many scheduling algorithms have been proposed [1][6], such as: Min-min, Max-min, suffrage, Xsuffrage, FirstPrice and FirstReward. These algorithms provide a basic method for resource scheduling problem. However, more advanced scheduling algorithms are necessary in response to new needs of QoS satisfaction or system performance improvement.

To maximize the resource consumers' business objective under the deadline constraint, the HRED heuristic [8] sorts all combination rank values between resource providers and users, and provides a near-optimal solution for the user's utility maximization problem. J. Yu et al. [9] propose a cost-based workflow scheduling algorithm that minimizes execution cost while meeting the deadline for delivering results. In the software service provision, B.Y. Wu et al. [10] propose to improve resource allocation through tracing and prediction of workload dynamics of component services as requests traverse and pipeline through the workflow. S. Garg et al. [12] propose a Linear Programming/Integer Programming model for scheduling jobs to resources, and design the LPGA algorithm to decrease the combined user spending and resource utilization.

R.S. Chang et al. [14] use ant colony algorithm to balance the entire system load while minimizing the makespan of a given set of jobs. C. Hu et al. [7] recognize the problem of dynamic Web Service selection with QoS global optimal in grid workflow, and present the HPSOA algorithm, which mixes the PSO and genetic algorithm together to find the optimal route. However, this work does not support user personalization. In [13], considering the execution time and execution cost, A.K.M. Talukder et al. apply the Genetic algorithm into scheduling to generate a set of trade-off schedules. This work is an improvement in QoS based scheduling research. However, it does not support other objectives rather than time and cost, and it does not give any concrete solution for supporting user personalization. Besides, the complexity of GA and Ant colony based scheduling algorithms makes themselves not scalable for Grid environment.

Trust degree and reliability are another two popular QoS requirements in scheduling. Dongarra et al. [3] take running time and reliability as scheduling objectives, propose a bi-objective scheduling algorithm. The reliability is also taken as one scheduling objective in [4] [5]. 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, and, which one should be the secondary.

Above all, these works only consider limited scheduling objectives, more research should be done to give a flexible QoS-based scheduling algorithm, which supports arbitrary objectives and user personalization.

D. Challenging issues

Generally, QoS requirements include execution time, economic cost, trust degree, reliability, and some other criteria [6]. The new scheduling algorithm should support multi-objective scheduling to meet these QoS requirements. In addition, even on the same criteria, people have the different preference degree, so user personalization should also be supported.

E. Our contribution with comparing to related works

To enhance and improve the user QoS-based scheduling, we have identified four crucial requirements:
Scalability: New scheduling mechanism should be scalable to support large infrastructures with thousands of service providers.
Flexibility: Not only to satisfy the specific end users, it is a bigger challenge to meet all different kinds of users' requirements.
Multi-objective supported: A rich collection of the scheduling objectives has been collected in [6]. An ideal user oriented scheduling algorithm must support any QoS requirements, and should be easily extended to more scheduling objectives.
User personalization supported: Ideally, the users' preferences value on different objectives should be computed automatically and precisely.

In this paper, we address the resource scheduling problem based on the four requirements and propose a novel resource scheduling method in the service oriented Grid environment. Inspired by the mode of the Internet web search engine, like Yahoo, Google, we design our own "Service Providers Search Engine" (SPSE). People submit their job description to SPSE, and SPSE searches and returns a list of good service provider candidates. Then the end user selects one solution from the list, his job will be executed on the selected resource. SPSE has four advantages: First, it can be scalable deployed in large-scale Grid environment, supporting a large number of users and services. Second, it is flexible and generic because it can be employed by arbitrary different people, who have different preferences on the services. Third, it supports multiple objectives, including time, economic cost, trust degree and so on, and can be easily extended to support more criteria. Finally, it can capture every user's preference values precisely and automatically, support user personalization.

This article is organized like this: we discuss the system model and state the problem in section 2. The detailed design of the SPSE scheduling mechanism is given in section 3. In section 4, we evaluate SPSE mechanism from three aspects: scalability, precision of preference values and precision of solutions. In section 5, we give the conclusion and discuss the future works.

II. System model and problem statement

A. Job model

A job is represented as in Formula 1:

$$job = \{user\_id, job\_id, instructions, service\_type\} \quad (1)$$

Where *user_id* represents the job's owner; *job_id* uniquely identifies the job; *instructions* represents 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 the 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.

B. Resource model

A service provider is represented as in Formula 2:

$$resource = \{resource\_id, service\_type, cpu, price, trust, \ldots, (other\ criteria)\} \quad (2)$$

Where *resource_id* uniquely identifies the job; *service_type* indicates which kind of service this resource provides; *cpu* represents its processor's capability; *price* indicates the money that users have to pay for resource utilization per second; *trust* represents the resource's trust degree; we reserve the *other criteria* field to support more objectives.

C. SPSE operations

SPSE mechanism comprises several sub-algorithms. We define the related operations and conceptions before giving the details:

Definition 1 (Service providers list (*SPL*)) *SPL_i lists all the service providers that can serve for job_i. SPL_i has two attributes:*

(1) All the service providers in SPL_i provide the same type of service.

(2) Different service providers may show different performances.

Definition 2 (Solutions list (*SoL*)) *SoL_i is also a list of service providers that provider service for the job_i. It is closely connected, but different 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 no bigger than the size of SPL_i.

(3) Both SoL_i and SPL_i contain the service provider SP_i who will execute job_i at last.

Definition 3 (Ranked solutions list (*RSoL*)) *$RSoL_i$ is generated by sorting 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 sorted in order by certain algorithm.

(3) $RSoL_i$ is presented to the end user in order.

Operation 1 ($\boldsymbol{SPL_i \leftarrow Search(job_i)}$) *$Search(job_i)$ operation searches service providers from Grid information services (GIS) for job_i, and returns SPL_i.*

Operation 2 ($\boldsymbol{SoL_i \leftarrow Filter(SPL_i)}$) *$Filter(SPL_i)$ operation sifts out the poor service providers from SPL_i, and leaves behind good service providers into SoL_i.*

Operation 3 ($\boldsymbol{RSoL_i \leftarrow Rank(SoL_i)}$) *After receiving SoL_i from $Filter(SPL_i)$ operation, SoL_i should be sorted into $RSoL_i$ using $Rank(SoL_i)$ operation: let better solutions rank higher than poorer solutions.*

Operation 4 ($\boldsymbol{UP \leftarrow Update(UP)}$) *User will select one solution from $RSoL_i$, and his job_i will execute according to his selection. Based on the selection and $RSoL_i$, $Update(UP)$ operation is in charge of calculating user's preferences(UP).*

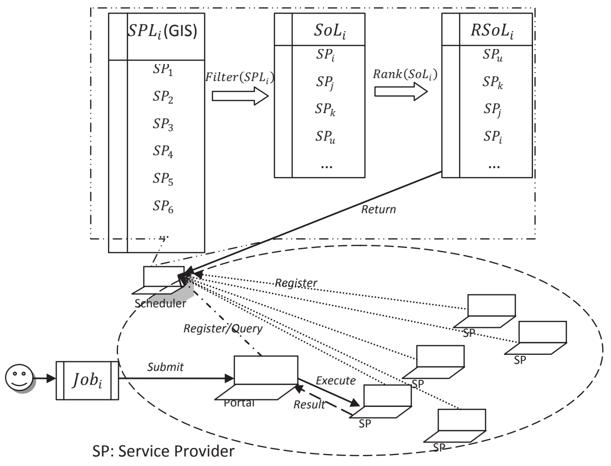


Figure 1. SPSE: resource scheduling mechanism.

D. Problem statement

In the Grid environment, given job model, resource model and SPSE operations, we seek a QoS-based scheduling algorithm, which supports multi-objective and user personalization.

III. SPSE SCHEDULING MECHANISM DESIGN

A. SPSE overview

The SPSE algorithm is illustrated in Figure 1. In Grid environment, all service providers are registered into GIS. When a new service provider joins Grid, its ID, service type, resource performance (e.g. CPU, memory) and other related features are registered into GIS using resource specification language (e.g. RSL). Table 1 shows the details of SPSE: the end-user submits his job description to the Grid portal, then the Grid portal contacts with scheduler to search service providers. After sifting out poor solutions and ranking the remained solutions, a solution list is returned back to the end-user. The end-user chooses one service provider to execute his job. Then the job is submitted to this remote service provider, and the user preferences value is updated.

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

B. $Search(job_i)$

Operation $Search(job_i)$ searches from the registered service providers, and finds the service providers based on two conditions:

(1) Its service type is the same with job's required service type;

Table I
DESCRIPTION OF SPSE

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

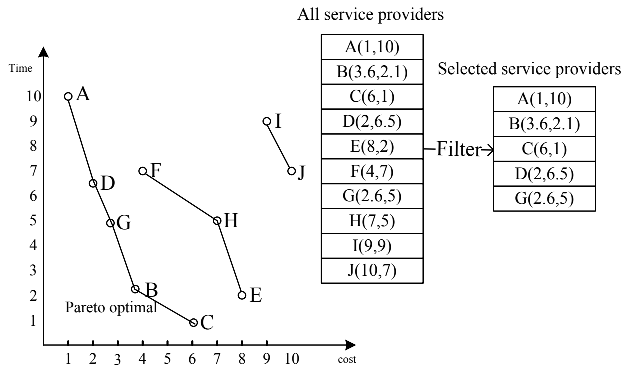


Figure 2. Filter service providers

(2) The service provider is available.
SPL_i is generated after this operation. The time complexity of operation $Search(job_i)$ is $O(N)$, where N is the number of all service providers .

C. $Filter(SPL_i)$

We assume the providers show different performances on service quality, and users are rational, compared with slow-running, expensive, unreliable providers, users prefer fast-running, cheap, reliable service providers. Then, the bad service providers whose performance is weak need not be involved in the scheduling. To improve the scheduling efficiency, we employ a Pareto optimal-based selection method to filter the service providers.

Example 1: As shown in figure 2, we have 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, the service provider A could finish this task in 1-second with 10 dollars. If comparing service provider D with F, we know D spends less money, and less time than F (2 < 4, 6.5 < 7). Meanwhile, no other service providers spend less time and less money than D. Therefore, D is one non-dominated solution [15]. Similarly, the service providers: $\{A, B, C, G\}$ are all non-dominated solutions. Therefore, the final pareto optimal solution set is $\{A, B, C, D, G\}$. These five service providers will take part in the following scheduling steps, and the other five will be sifted out. The time complexity of operation $Filter(SPL_i)$ is $O(MN^2)$ [16], where M is the number of criteria, and N is the number of service providers in SPL_i.

Theorem 1 *The time minimization solution (cost minimization solution, trust maximization solution) is not sifted out by the operation $Filter(SPL_i)$.*

Proof: Suppose the time minimization solution SoL_i^j is sifted out by the $Filter(SPL_i)$ operation. Then we know that SoL_i^j is a dominated solution. According to the definition of "dominated" [15], we get there exist a solution, suppose it is SoL_i^k, whose execution time $t_i^k < t_i^j$, cost $c_i^k < c_i^j$, and trust degree $r_i^k > r_i^j$. However, SoL_i^j is the time minimization solution, which contradict SoL_i^k. Therefore, the assumption is false, and so the theorem. ■

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

D. $Rank(SoL_i)$

SPSE supports multiple objectives. Execution time, economic cost, and trust degree are the most popular scheduling criteria in Grid environment. SPSE supports these criteria, and can be easily extended to support more criteria.

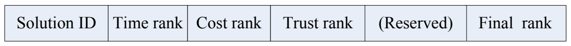
The $Rank(SoL_i)$ operation sorts all solutions into a certain order, where solutions with shorter time, less economic cost, higher trust degree are ranked ahead of others. Moreover, the most favorite solution by the end user should be listed at the top. Before designing the ranking algorithm, first we introduce the concept of user preferences.

Definition 4 (User preferences (*UP*)) *are a set of parameters: $\{p_1, p_2, p_3, \ldots, p_m\}$, 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: $UP_x = p_1^x, p_2^x, p_3^x, , p_m^x$.*

(2) Initialization: $p_1^x = p_2^x = p_3^x = \ldots = p_m^x = 1$.

(3) Suppose there are two users: x *and* y*, then no relationship between $\{p_1^x, p_2^x, p_3^x, \ldots, p_m^x\}$ and $\{p_1^y, p_2^y, p_3^y, \ldots, p_m^y\}$.*



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

Figure 3. Ranking struct

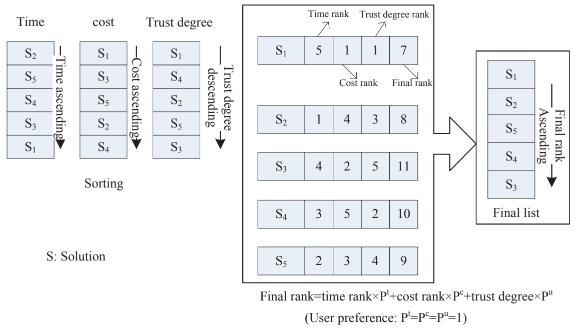


Figure 4. Ranking solutions

(4) There are no relationship between two parameters: p_i and p_j, where $i \neq j$ and $1 \leq i, j \leq m$.

(5) UP_x will be updated every time after a scheduling process.

Based on the 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 (as in Figure 3) that will be used in both analysis and implementation, and then we will introduce the $Rank(SoL_i)$ operation with example 2.

The solution struct comprises six parts: *solution ID*, *time rank*, *cost rank*, *trust rank*, *reserved field*, and *final rank*. *Solution ID* uniquely identifies a solution; *Time rank* keeps the solution's rank number after sorting all the solutions into the execution time ascending order; *Cost rank* and *trust rank* store the cost rank number and trust degree rank number respectively, according to the cost ascending order and the trust degree descending order; *Reserved* field is reserved for more objectives; *Final rank* stores the computed final rank number, which is gotten by using Formula 3:

$$final\_rank = time\_rank \times p_1^x + cost\_rank \times p_2^x + trust\_rank \times p_3^x + (reserved \times p_i^x) \tag{3}$$

where p_1^x, p_2^x, p_3^x, p_i^x are UP values on time, economic cost, trust degree, and the extended criteria, respectively.

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 rank number and user preferences, we get the final rank values using formula 3: $final\ rank_{s_1} = 5 \times 1 + 1 \times 1 + 1 \times 1 = 7$, $final\ rank_{s_2} = 1 \times 1 + 4 \times 1 + 3 \times 1 = 8$, $final\ rank_{s_3} = 4 \times 1 + 2 \times 1 + 5 \times 1 = 11$, $final\ rank_{s_4} = 3 \times 1 + 5 \times 1 + 2 \times 1 = 10$, $final\ rank_{s_5} = 2 \times 1 + 3 \times 1 + 4 \times 1 = 9$.

Table II
$Rank(SoL_i)$ ALGORITHM

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

Sorting all solutions again based on the 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 details of $Rank(SoL_i)$ operation. First, sort all the solutions according to different scheduling criteria, and save the rank number into struct (Lines 1-3). Second, calculate the final rank number using formula 3 (Lines 4-5). At last, sort all solutions according to the final rank number. If we use 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 $O(N)$. The last $sort\_final\_rankings()$ function's time complexity is $O(N \log N)$. Therefore, the overall time complexity is: $O(MN(\log N))$.

E. $Update(UP)$

How to calculate the preferences value is a challenge. Existing work have ever used UP to describe user preference to different criteria, but not compute it automatically, they let the end user set the value artificially [11]. Obviously, it is difficult for a user to set an accurate value to describe his preferences. As defined in last section, we assume that every user x has his own set of preferences: $UP_x = \{p_1^x, p_2^x, p_3^x, , p_m^x\}$. After the $Rank(SoL_i)$ operation, SPSE returns to the user a list of solution candidates. The solution user selected can be used to update *UP*.

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

$$p_1^{'} = p_1 \times (1 + \frac{\overline{t_{top}} - t_{user}}{\overline{t_{top}}}) \tag{4}$$

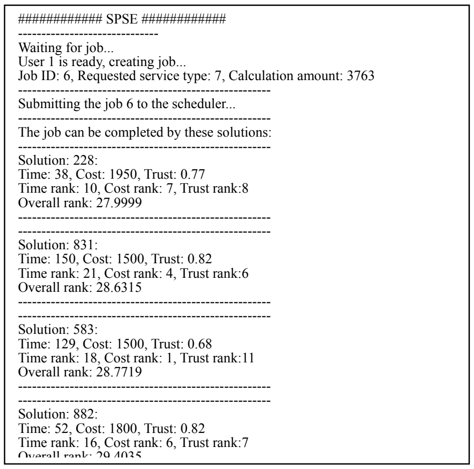
From this formula, we get that:

(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, t_{top} is the execution time of number one solution, p_1 is the *UP* value on execution time, $p_1^{'}$ is the updated new 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



```
############ 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
------------------------------------------------------
------------------------------------------------------
Solution: 882:
Time: 52, Cost: 1800, Trust: 0.82
Time rank: 16, Cost rank: 6, Trust rank:7
```

Figure 5. The prototype of SPSE

solutions above the user selected solution). Similarly, p_2, p_3 and p_m, which stand for the UP value on economic cost, trust degree, and extended criteria respectively, are calculated below:

$$p_2^{'} = p_2 \times (1 + \frac{\overline{c_{top}} - c_{user}}{\overline{c_{top}}}) \tag{5}$$

$$p_3^{'} = p_3 \times (1 + \frac{r_{user} - \overline{r_{top}}}{\overline{r_{top}}}) \tag{6}$$

$$p_m^{'} = p_m \times (1 + \frac{\mp e_{user} \pm \overline{e_{top}}}{\overline{e_{top}}}) \tag{7}$$

To constrain the preferences value, some rules are followed after having gotten $p_i^{'}$ $(i \in 1, 2, 3, \ldots, m)$:

(1) If $p_i^{'} < 0$, then set $p_i^{'} = 0$;

(2) Reward principle: reward factors are used to award the user preferences, whose values are improved sharply once user changes his interests:

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

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

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

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

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

IV. Analysis and Experiments

A. The SPSE prototype

We design and implement a simulation prototype for SPSE. In the prototype, the parameters of 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 $Instructions_i \in [0, 10000]$, $service\_type_i \in [1, 2, \ldots, 9]$. For a $resource_j = \{resource\_id_j, service\_type_j, cpu_j, price_j, trust_j\}$, $service\_type_i \in [1, 2, \ldots, 9]$, $cpu_j \in [20, 100]$, $price_j \in [10, 20]$, $trust\ degree_j$ is a decimal in [0,1]. The SPSE prototype is illustrated in Figure 5. After the user submits his job description, a list of solutions is returned to him. What user should do is selecting one from the list, then his job will be executed on the selected resource.

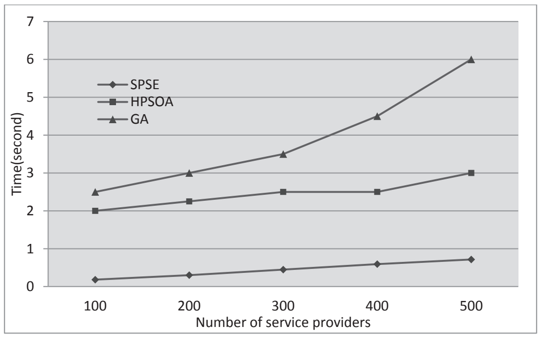


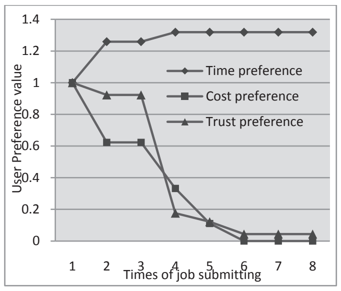
Figure 6. Scheduling efficiency

B. Experiments

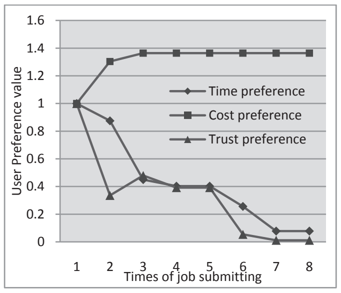
Three performance metrics are introduced to evaluate the SPSE prototype:

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 using our prototype: the execution time of scheduling algorithm is illustrated in Figure 6. We can see that, along with the number of service providers increasing from 100 to 500, the execution time is always less than 1 second. Compared with the Genetic algorithm and HPSOA algorithm [7], the SPSE has a much higher scheduling efficiency. Actually, in our experiments, even there exists 2000 service providers, the scheduling time is only around 1 second. If SPSE is re-implemented by the parallel algorithm,we assume the scheduling time will become much shorter.

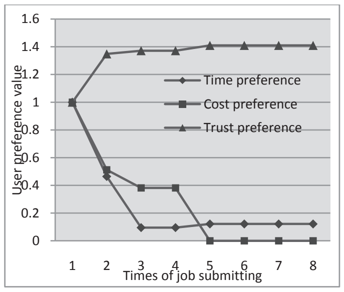
2) Precision of preference values: The second experiment concentrates on how precise the SPSE could capture the UP value. Let three end users participant in the SPSE scheduling, where the first user only care about job execution time, the second user only care about the economic cost, and the third one only care about service provider's trust degree. After 5 times of job submitting, for the first user, we see that UP values have become stable, where preference value on execution time is around 1.4, and the preference value on the economic cost and trust degree is almost 0 (Figure 7a). The same situation happens to the second and the third user, where with the preference values on the economic cost and trust degree up to 1.4 respectively, 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;

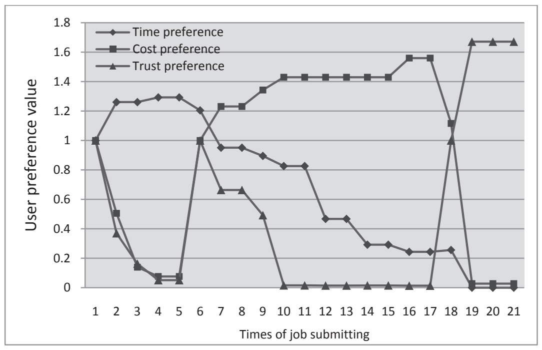


Figure 8. UP values of a user who prefers minimum execution time first, then changes to prefer minimum economic cost, and at last, changes to prefer maximum trust degree.

the other two criteria are much smaller, which are around 0.

If the first user changes his preference, for example: initially, he prefers time minimization. From Figure 8, we can see that, after the 6th job submitting, he does not care the execution time any more, so his preference value on the time and trust degree continue to decline after the 6th job submitting, while the preference value on the cost rises up suddenly because of the reward principle. Then after 18th job submitting, the user changes his interests again, the preference value on the trust degree rises up, and the preference value on the time and cost decline rapidly. This experiment proves that SPSE can capture user's preferences value quickly and precisely, even when the user changes his preferences.

3) Precision of solutions: If submitting the same job for several repeated times, after how many times the user favorite solutions will be the No. 1? From Figure 9, we know that UP values of the first user are no longer changed only after the second job submitting, which means if the job and Grid environment is the same, then the user's favorite service provider will be at the top from the second job submitting. The UP values of the second user and the third user show the same situation. This experiment proves that after capturing the user's preference value, the most preferred solution by the user will be at the topmost quickly and precisely.

V. Conclusion and Future work

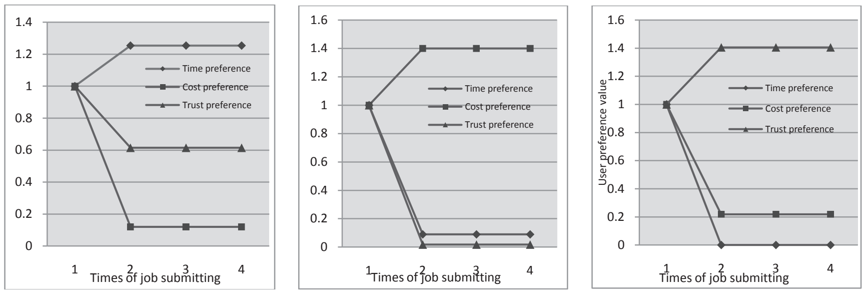
The user QoS requirements are important for the popularization use of Grid. In this paper, we identify four requirements for the QoS-based service scheduling problem. Based on this, we propose the SPSE scheduling algorithm, which is flexible in satisfying different QoS requirements, supporting multi-objective scheduling and user personalization. Also, the SPSE scheduling mechanism is scalable to be deployed into large-scale Grid. To the best of our knowledge, this is the first user personalization supported algorithm on service searching and scheduling.

We implement a simulation prototype for SPSE, evaluating its performance from three aspects: scalability, precision of preference values and precision of solutions. Compared with the GA and HPSOA algorithm, SPSE shows a much higher scheduling efficiency. And SPSE works well even thousands of service providers existed in the Grid environment. If designed into the parallel algorithm, 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 based on UP. And UP value will become stable after several jobs submitting. Using SPSE, the most preferred solution by an end user will be listed at the top. Even the user changes his preferences, the preference value would be recaptured automatically and quickly.

However, the proposed SPSE is still simple, we ignore some issues like: fault tolerance, preemptible. These will be considered in our future work.

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.



(a) (b) (c)

Figure 9. UP values of a user who submits a same job to a same Grid environment: (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;

References

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

[2] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic Models for Resource Management and Scheduling in Grid Computing. Journal of Concurrency and Computation: Practice and Experience. vol.14, pp. 1507-1542, Dec. 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] C. Hu. M. Wu, G. Liu, et al. QoS Scheduling Algorithm Based on Hybrid Particle Swarm Optimization Strategy for Grid Workflow. The Sixth International Conference on Grid and Cooperative Computing, pp. 330-337, Urumchi, Aug. 2007.

[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] B. Wu, C.H. Chi, Z. Chen et al., Workflow-based resource allocation to optimize overall performance of composite services. Future Generation Computer System. vol.25(3), pp.199-212, 2009.

[11] M. Wieczorek, S. Podlipnig, R. Prodan, et al. Bi-criteria Scheduling of Scientific Workflows for the Grid, Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), IEEE Computer Society Press, Lyon, France. May 2008.

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

[13] A.K.M. K.A. Talukder. M. Kirley, R. Buyya, C.K. Tham: Multi-objective Differential Evolution for Workflow Execution on Grids. Proceedings of the 5th international workshop on Middleware for grid computing: at the ACM/IFIP/USENIX 8th International Middleware Conference, ACM, California, 2007.

[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] 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