A Service-Clustering-based Dynamic Scheduling Algorithm for Grid Tasks

Zhoujun HU, Zhigang HU, Zhenhua LIU
School of Information Science and Engineering, Central South University, Changsha 410083, China
E-mail: zghu@mail.csu.edu.cn

Abstract

Task scheduling in Grid environment is a challenging problem because of NP-complete nature of scheduling issue and dynamic characteristic of the environment. Not constrained by local scheduling policy of Grid site, a dynamic service evaluation method based on cloud model is presented. Then we obtain performance metrics of dynamic service. According to dynamic service evaluation, an adaptive and dynamic service clustering method is derived from PSO (Particle Swarm Optimization)-based clustering algorithm. It gathers the services with similar or same QoS (Quality of Service) into one cluster. A dynamic meta-task scheduling algorithm is proposed in light of service clustering. We conduct extensive experiments on the implemented prototype. The results show that our algorithm outperforms prior well-established algorithms in terms of time complexity and user QoS guarantee.

Keywords: *service Grid, performance evaluation, dynamic clustering, scheduling algorithm*

1. Introduction

As a large-scale networked computing system, Service-Oriented computing Grid [1] combines OGSA (Open Grid Service Architecture) with WSRF (Web Service Resource Framework). Through encapsulation of diverse high performance resources in multiple (virtual) organizations, it provides unified Grid service for users to solve significant scientific problems or e-business applications. Due to NP-complete nature of scheduling problem [2] and extremely large amount of resources in Grid, decreasing amount of candidate resources is an effective way for lowering computational complexity of task scheduling. In service Grid environment, there exist many services providing similar or same QoS. Putting these services into a cluster can reduce the problem-scale of task scheduling, which has a great significance for designing efficient and effective scheduling algorithm.

With the purpose of service clustering in dynamic Grid environment, there are two issues needing to be solved. Firstly, we need to know how to evaluate whether the QoS provided by the services is similar or same considering dynamic characteristic and autonomy of Grid resource. That is to say, how to get the performance metrics of dynamic service. The second issue is how to group the services with similar or same QoS together. However, since service performance variation or service provider (resource) may publish or terminate a service at any moment, the service may belong to different clusters or join in or depart from a cluster. And these variations may happen frequently. Thus, how to choose an adaptive and dynamic clustering method is a challenge.

We are devoting ourselves to resource performance evaluation in dynamic computational Grids all the time. Following our previous works [4-6], this paper mainly focus on a low-complexity meta-tasks scheduling algorithm. To the best of our knowledge, to date, there has

been little or no work on dynamic service clustering. Listed below are our original contributions.

(1) Irrespective of resource local administrative policy, a dynamic service evaluation method based on cloud model is presented. And dynamic service performance metrics are obtained.

(2) An adaptive and dynamic service clustering method is derived from PSO-based clustering algorithm. Then, the services with similar or same QoS are put into a cluster.

(3) A dynamic meta-task scheduling algorithm is proposed according to service clustering. Through analysis and simulation, our algorithm is theoretically sound and has a real potential.

The rest of the paper is organized as follows. Section 2 gives the related work and makes a comparison between our work and previous work; Section 3 describes the definitions and basis of our work; Section 4 builds a service-clustering-based scheduling framework; Section 5 presents a dynamic service evaluation method from cloud model; an adaptive and dynamic service-clustering method grounded on PSO is proposed in Section 6; Section 7 introduces the new dynamic scheduling algorithm; experiment results and evaluation are discussed in Section 8; and finally we conclude the paper and outline the future directions.

2. Related work

In this section, we describe some of the works that are relevant to the problems addressed in this paper.

(1) Resource or service performance evaluation

Based on some distributions that the number and execution time of arrival task follows, the distribution function of task completion time is gained [3] from queuing theory. It only uses expectation of task completion time for measuring resource serving capacity, leading to great uncertainty. To solve this weakness, we [4] employ entropy to characterize resource uncertainty. Our previous work [5, 6] and Berten [7] evaluate resource availability according to related statistic of resources and tasks, and present some availability metrics including average waiting queue length and average number of used CPUs, etc. However, the values of the metrics in these papers are also only average value. The works in [3-7] aim at evaluation of resource with “FCFS (First Come First Serve)” local scheduling policy, while [8] focuses on cycle-sharing resources. A multi-state model is presented according to the factors causing the availability or unavailability of resource in [8], which applies semi-Markov Process models to the prediction of resource future state. However, the predicted state can only be classified into two types: the available and the unavailable (caused by various reasons), without indicating specific resource capacity. The above-mentioned methods are limited to specific local administrative strategy of a resource, while our method is not. A resource-performance-fluctuation-aware workflow scheduling algorithm for Grid Computing is reported in [9]. The algorithm improved from DCP (Dynamic Critical Path) algorithm only utilizes spare time-slot of resource and needs to update ranks of unscheduled tasks after each task is scheduled. The problem of dynamic memory capacity evaluation and a memory-conscious scheduling algorithm is addressed in [10, 11], which use a simple weighted average method to evaluate resource memory capacity based on some historical data.

(2) Clustering-based task scheduling algorithm

There are merely several works in this field. The workflow tasks are scheduled to different VO (Virtual Organization) according to CCR (Computation-to-Communication Ratio) of the task [12, 13]. Data-intensive tasks are scheduled to a VO in which resources are linked by high bandwidth, while computing-intensive tasks mainly consider computing power of a VO. They reach some balance on the trade-off between computation time and communication

time. On the basis of the force field and potential energy theory in physics, a clustering-based resource aggregation scheme [14] can reduce the problem scale efficiently under the background of iVCE (Internet based virtual computing environment) for Memory.

Motivated by but differing from these works, our clustering-based algorithm is on the basis of service performance evaluation, and then gathers services with similar or same QoS together adaptively to dynamic Grid environment.

3. Problem definition

Definition 1: Computing service. From WSDL (Web Service Description Language) specification, a service can be represented by a triplet:

$CS = \langle Name, Perf, Function \rangle$.

Therein, $Name$ and $Function$ is the name and function description of computing service, respectively. $Perf$ is interpreted as follows.

$Perf$: The performance of a dynamic service. Due to resource load variance, the service performance is dynamic after resource encapsulation, resulting in providing different QoS. Through performance evaluation, a triplet $\langle Ex, En, He \rangle$ is employed to describe dynamic service performance.

Utilizing the definition of service capacity in [15], we give it directly.

Definition 2: Servslot. A servslot quantifies the level of resources needs necessary to handle a single service request. It represents the aggregated capacity of the collection of software and hardware resources required for the successful operation of a single service instance.

Therefore, the capacity of a dynamic service can be represented by the number of servslots. Due to dynamic nature of service, we assume the current available capacity of CS_i is $i(0 \le i \le N_i)$. N_i is static capacity of CS_i .

With the assumption that services with same performance provide same QoS, we gather services providing similar or same QoS into one cluster, and then the following definition is demonstrated.

Definition 3: Service cluster. A group of services provide similar or same QoS. It is expressed as $CSC = \{CS_1, CS_2, \cdots, CS_u\}$. CS_i is the member of CSC . The service cluster broker records detail information of CSC (e.g. name, function, capacity and interface specification, etc.) as well as the mapping between CSC member services and their providers.

4. Service-clustering-based scheduling framework

Figure. 1 depicts the service-clustering-based scheduling framework. Firstly, the unit of measurement for service capacity is obtained by exploiting method in [15] and the number of servslots of each service is obtained. Because of workload variation of resource, state of service capacity is needed to be monitored. Then, the performance of dynamic service is evaluated. How to determine the value of parameters including monitor cycle and samples length for evaluation and adjust them according to the results is out of the scope of this paper. When a service registers, the evaluation results are added to service description grammar. And, as the evaluation result is changed, update information in UDDI (Universal Description Discovery and Integration) actively and record the interval between two updatings. The monitor and evaluation work is done on resource end completely and needn't centralized management. Thus, pressure of centralized management and performance bottleneck can be eliminated.

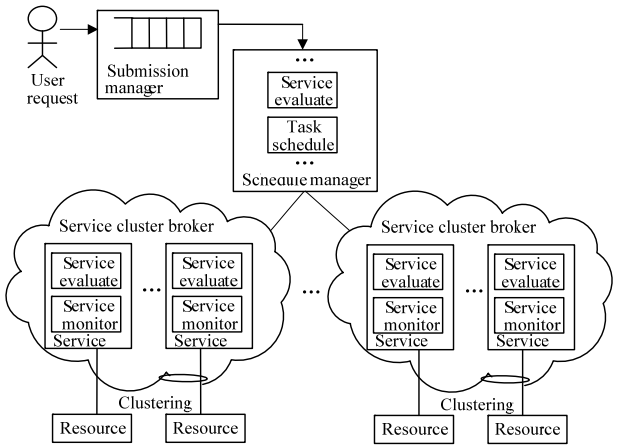

Figure. 1 Service-clustering-based scheduling framework

Then, gather similar or same QoS services into one *CSC*. The *CSC* broker maintains mapping between its member services and their service providers. In UDDI, it not only records *CSC* ID and provider's address (such as URL) of its member service, but also manages namespace, interface, operation and parameters information of *CSC*. Scheduling manager makes scheduling decision based on user QoS requirements and *CSC* information in UDDI. When binding service, based on WSDL, as the meta-information of *CSC*, the PortType and agent of member service are generated dynamically through DDI, reflection mechanism and dynamic compiling mechanism.

5. Service performance evaluation based on cloud model

5.1. Brief introduction to cloud model

Cloud model [16, 17], presented by Deyi Li, is a model for uncertainty reasoning and transformation between qualitative object and quantitative object. It has been widely applied in many domains such as intelligent control, data mining and knowledge discovery, etc. It is very appropriate to model Grid resource with volatile performance. In this paper, we firstly use cloud model for evaluation and present a performance evaluation method based on cloud model.

Definition 4: Cloud and Cloud Drop. Assuming that U is a quantitative set, as the universe of discourse, and C is a term associated with U. For a quantitative value $x \in U$ which is a random implementation of qualitative notion C, the determination degree of the membership degree of x in U to the term C, $\mu(x) \in [0,1]$, is a random variable with a stable tendency. That is, $\mu : U \to [0,1] \quad \forall x \in U, x \to \mu(x)$. Then, we call distribution of x on U Cloud represented by $C(x)$. x is called Cloud Drop [17].

The conception that the cloud model expresses can be reflected by the unique vector $< Ex, En, He >$, where Ex, En and He are expected value, entropy and hyper-entropy of cloud, respectively. The expected value Ex is the position at U corresponding to the gravity center of the cloud. The entropy En reflects the fluctuation extent of the samples. The bigger En is, the more uncertainty of qualitative notion. The hyper-entropy, He, is a measure of uncertainty of the entropy En.

The transformation from quantitative value to qualitative conception can be implemented by backward cloud algorithm, which is to say a group of quantitative data can be transformed

to $< Ex, En, He >$-described qualitative notion. Algorithm 1 is the backward cloud algorithm [17].

Algorithm 1: Backward cloud algorithm [17].

Input: N cloud drops $\{x_1, x_2, \cdots, x_N\}$.

Output: the expected value Ex, the entropy En and the hyper-entropy He of the qualitative conception reflected by the N cloud drops.

Steps:

(1) According to x_i, average value $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} x_i$, first order sample central moment $\frac{1}{N}\sum_{i=1}^{N}\left|x_i - \bar{X}\right|$, and sample variance $S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{X})^2$ are computed.

(2) Estimated value of Ex is $\hat{E}x = \bar{X}$.

(3) Estimated value of He is $\hat{H}e = \sqrt{\frac{\pi}{2}} \cdot \frac{1}{N}\sum_{i=1}^{N}\left|x_i - \hat{E}x\right|$.

(4) Estimated value of En is $\hat{E}n = \sqrt{S^2 - \frac{1}{3}\hat{H}e^2}$.

5.2. Dynamic service evaluation based on cloud model

We view performance variation of dynamic service as a random process and leverage cloud model to evaluate the performance of dynamic service.

We directly use the 3 elements in the vector as performance metrics of dynamic service. According to the number of available servslots of a service, a triplet $\langle Ex, En, He \rangle$ obtained from backward cloud algorithm is employed to describe dynamic service performance.

Here, the expected value Ex is the average service capacity and reflects resource idle processing power; the entropy En is fluctuation level of service's dynamic capacity; and the hyper-entropy He is a reflection of stability of the entropy En.

Assuming that during a period of time T, the capacity series of 4 services $CS_i (1 \le i \le 4)$ are shown in Table 1.

Table 1 Service capacity of 4 services

Service	Number of servslots									
CS_1	3	2	2	2	3	2	2	3	2	3
CS_2	6	5	6	5	6	5	6	5	6	5
CS_3	4	8	3	10	2	9	1	4	9	2
CS_4	4	3	4	4	3	3	4	4	3	4

From Algorithm 1, the evaluation eigenvalue $Perf$ of the 4 services are calculated and are respectively $< 2.4, 0.382, 0.602 >$, $< 5.5, 0.383, 0.627 >$, $< 5.2, 2.626, 3.810 >$ and $< 3.6, 0.382, 0.602 >$. Through dynamic performance evaluation, we can easily see that, though CS_2 has the average capacity as much as CS_3, CS_2 is more stable than CS_3. Thus, if only the two services can cater to a task's QoS requirement, CS_2 may be a better choice for executing the task because of its greater stability. CS_1 and CS_4 have the same stability, but the average value of CS_4 is larger. A task may need a service with larger capacity accounting for its deadline. Thus, the expected value Ex is also an important performance metric of dynamic service.

When a service is published, we need to add the service performance metrics $\langle Ex, En, He \rangle$ to the unified service description grammar. Compatible with UDDI specification, the metrics information can be added by extending UDDI.

6. QoS-oriented dynamic service-clustering

Based on dynamic service performance evaluation, the services with similar or same 3 metrics are gather into one CSC. There are many solutions to service clustering such as similarity measurement in web searching domain and clustering algorithm. However, as a service provider may join in or depart from Grid and service performance varies, and these variations may happen frequently, an adaptive and dynamic clustering method is needed. Thanks to swarm intelligence paradigm with dynamic aggregation nature [18] which exactly satisfies our needs, the paper applies PSO to dynamic service-clustering.

6.1. Brief introduction to PSO

PSO [18] is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling.

The system is initialized with a population of random solutions and searches for optimum by updating generations. The potential solutions, called particles, fly through the problem space by following the current optimum particles. Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called $PBest$. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbors of the particle. When a particle takes all the population as its topological neighbors, the best value is a global best and is called $GBest$.

The particle swarm optimization concept consists of, at each time step, changing the velocity of each particle towards its $PBest$ and $GBest$ locations (according to formula (Ⅰ)& (Ⅱ)).

$$v_{k+1} = c_0 v_k + c_1(pbest_k - x_k) + c_2(gbest_k - x_k) \quad (Ⅰ)$$

$$x_{k+1} = x_k + v_{k+1} \quad (Ⅱ)$$

Therein, v_k is the velocity vector of a particle; x_k is the current location; c_0, c_1 and c_2 represents cognitive coefficient, c_0 is a random number within (0,1), c_1 and c_2 are random numbers within (0,2). v_{k+1} is the vector summation of v_k, $pbest_k - x_k$ and $gbest_k - x_k$.

The velocity of each dimension is constrained within $v_{\max}$. That is to say, when $v_k > v_{\max}$, then $v_k = v_{\max}$; and when $v_k < -v_{\max}$, then $v_k = -v_{\max}$.

6.2. PSO-based dynamic service-clustering

From service performance evaluation, the PSO-based clustering algorithm is employed to address the problem of service clustering which groups the services with similar or the same QoS together. Our method is dynamic and environmentally adaptive.

Algorithm 2: PSO-based dynamic service-clustering algorithm.

Input: n services, each has a three-dimensional eigenvector $\langle Ex, En, He \rangle$; m CSC; maximum number of iterations $MaxIter$; the number of particles $PNum$; $v_{\max}$, c_0, c_1 and c_2.

Output: m service clusters and centroid of each CSC.

Steps:

(1) Initialization. The expression of service is replaced by $Perf$ (abbreviated as Pf).The services to be clustered are $Pf = \{Pf_i, i = 1,2,\cdots,n\}$, each element is a three-dimensional vector $\langle Ex, En, He \rangle$. Each particle denotes m cluster centroids. Generate initial position and velocity of each particle uniformly distributed between 0 and $v_{\max}$.

(2) Generate particle's new position according to its current position and velocity.

(3) Based on centroids represented by each particle, clustering all services according to Euclid distance:

$$\|Pf_i - Z_j\| = \min_{k=1,2,\cdots,m} d(Pf_i, Z_k) = \min_{k=1,2,\cdots,m} \|Pf_i - Z_k\| = \min_{k=1,2,\cdots,m} (\sum_{h=1}^{3} (x_{hi} - x_{hk})^2)^{1/2} , \quad (\text{III})$$

where $Z_k(x_{hk})(1 \le k \le m)$ is the centroid of service cluster, $x_{hi}(1 \le h \le 3)$ is the performance metric of dynamic service.

Consequently, Pf_i is put into CSC_j. The fitness function of particle is the sum of dispersion measurement of all clusters (see formula (Ⅳ)). Calculate the fitness value of each particle and compare it with the particle's $PBest$. If the new fitness value is smaller than $PBest$, set the new value to $PBest$.

$$Fitness(PIndex) = \sum_{k=1}^{m} \sum_{Pf_i \in CSC_k} d(Pf_i, Z_k) \quad (\text{Ⅳ})$$

(4) Find out $GBest$ according to $PBest$.

(5) Update velocity of each particle from formula (Ⅰ) with constraint of $v_{\max}$.

(6) Update position of each particle from formula (Ⅱ).

(7) The cycle from step (3) to (6) continues until its iteration number reaches $MaxIter$. The centroids denoted by particle with $GBest$ are the optimal service cluster centroids.

By referring to methodology in [19-21], parameter value of c_0, c_1, c_2, $v_{\max}$, m and $PNum$ can be determined. Usually, $m = \sqrt{n}$, $PNum \in [20,40]$.

Because Algorithm 2 searches for optimum in course of iterations, when service performance changes or resources join in or depart from Grid, as long as the fitness value larger than $PBest$ is detected, the algorithm continues and new clustering result is formed. Therefore, the algorithm has a good feature of self-adaptiveness to Grid environment. Meanwhile, the services clustering and task scheduling can be processed in parallel. The computational complexity of scheduling through service-clustering is no higher than that without service-clustering.

7. Service-clustering-based dynamic scheduling algorithm

All member services in a CSC provide similar or same QoS after service clustering. Consequently, the task scheduling involves two steps: initial cluster selection from service clusters and further service selection from the selected CSC. In service Grid environment, deadline and cost are two commonly considered factors of task scheduling. Built on performance metrics of dynamic service and clustering result, scheduling decision is made according to user's QoS requirements. When selecting CSC, we aim at minimizing task execution makespan; and when selecting service in CSC, the cost is the objective.

Algorithm 3: Service-Clustering-based meta-Task Dynamic Scheduling Algorithm (SCTDSA).

Input: meta-task set $TS = \{t_1, t_2, \cdots, t_i, \cdots\}$; task length l_i, deadline D_i and cost C_i of t_i; service cluster set $CSCT = \{CSC_1, CSC_2, \cdots, CSC_m\}$, service cluster $CSC_i = \{CS_{i1}, CS_{i2}, \cdots, CS_{iu}\}$.

Output: mapping between tasks and services.

Steps:

(1) Judge whether a service cluster can satisfy user's deadline requirement according to task length (l_i) and average capacity of the service cluster.

$$l_i / En \le D_i \quad . \qquad (\text{V})$$

(2) Find the service cluster with the biggest Comprehensive Performance value (see formula (VI)) in the candidate service cluster set.

$$CP(CSC_i) = \omega_1 Ex + \omega_2 \frac{1}{En} + \omega_3 \frac{1}{He} \quad , \qquad (\text{VI})$$

where ω_1, ω_2 and ω_3 indicates weight of Ex, En and He, respectively. And $\omega_1 + \omega_2 + \omega_3 = 1$. Specific relationship among Ex, En and He will be investigated in future work. In this paper, $\omega_1 = \omega_2 = \omega_3 = 1/3$.

(3) Map task t_i to the service with the minimum cost in the candidate service cluster selected in (2).

Yang *et al*. [21] found out that, when cluster number $m \le \sqrt{n}$, the dispersion inside a cluster is optimal. Therefore, the time complexity of Algorithm 3 is $O(\sqrt{n})$, much smaller than $O(n)$ of without clustering.

8. Experiment and evaluation

8.1. Experiment setup

We use two simulation tools in our experiment. One is GridG [22], which is a synthesizing computational Grid generator with realistic workload, network topology and many other rules discovered in real-life. Another is Lublin Workload Model [23], whose author analyzed workload logs from many large-scale parallel systems and exploited good-fitness distributions to model the parameters such as arrival-time, runtime of task, etc. Our experiment environment consists of 4 PCs. PC1, equipped with Pentium 4 CPU 2.8GHz, RAM 512MB and OS Fedora Core 6, runs GridG to simulate the Grid environment. We base on Lublin Workload Model to develop a task submitter running on PC2 with the same conFigureuration as PC1. The evaluation and clustering broker is implemented on PC3 conFigureured with Celeron CPU 1.8GHZ, RAM 512MB and OS Fedora Core 6. ConFigureured the same as PC3, PC4 is the scheduling manager in which our algorithm and other two algorithms for comparison are conducted. The 4 PCs communicate with each other through message passing mechanism. Thereby, these 4 PCs make up of a Grid prototype.

We compare our algorithm with QoS guided Min-min Heuristic [24] proposed by Prof. Sun and the prediction-based algorithm designed in the scheduler EMPEROR [25]. The evaluation metrics contain runtime consumed by scheduling algorithm, task makespan and service rejection ratio (SRR). 4 scenarios are setup: (1) Because SCTDSA is a clustering-based algorithm which can lower time complexity of task scheduling, we wish to verify runtime consumed by SCTDSA and unclustered QoS guided Min-min Heuristic. (2) For EMPEROR, different time series data leads to different prediction precision. Thus, the performance-evaluation-based SCTDSA is compared with EMPEROR with different prediction accuracy. (3) SCTDSA is a dynamic algorithm taking dynamic performance into consideration. Then, how SCTDSA outperforms Sun's static algorithm under different workload is exposed. We model varying workload through changing the inter-arrival time of tasks in Lublin Workload Model. (4) Because SCTDSA is a clustering-based algorithm, how the number of service clusters and the dispersion degree of a cluster influence scheduling results is also revealed. For each scenario, we repeat the simulation 30 times and take the average value as the final result.

As QoS parameters (deadline and cost) are not reco- rded in the trace, we follow a similar experimental methodology in [26] to model these parameters throu- gh two task classes: (1) high urgency and (2) low urgency. Each task in the high urgency class has a deadline of low D_i / R_i value and cost of high $C_i / f(R_i)$ value. R_i is runtime of the task, and $f(R_i)$ is a function representing the minimum cost the user will quote with respect to R_i. Conversely, each task in the low urgency class has a deadline of high D_i / R_i value and cost of low $C_i / f(R_i)$ value. This model is realistic since a user who submits a more urgent task to be completed within a shorter deadline is likely to offer a higher cost for the task to be finished on time. The arrival sequence of tasks from the high urgency and low urgency classes is uniformly distributed. Values are normally distributed in the range of each deadline and budget parameters. The ratio of the means for each parameter's high-value and low-value is thus known as the high:low ($h:l$) ratio.

All parameters in the experiment are listed in Table 2. ($D/C\ h:l$ means Deadline/Cost high:low ratio; $D/C\ mean$ denotes Deadline/Cost low mean.)

Table 2 Parameter information

Parameter	c_0	c_1	c_2	*PNum*	v_{max}
Value	(0,1)	(0,2)	(0,2)	30	10
Parameter	*MaxIter*	$D\ h:l$	*D mean*	$C\ h:l$	*C mean*
Value	500	4	2	4	2

8.1. Results and analysis

(1) Scheduling runtime compared with QoS guided Min-min Heuristic

Figure.2 plots comparison of scheduling runtime traced on PC4 between SCTDSA and QoS guided Min-min Heuristic. When service number is small, SCTDSA is a little better than QoS guided Min-min Heuristic. With service amount rising, scheduling runtime of QoS guided Min-min Heuristic goes up rapidly, but SCTDSA's runtime increases only a little. There are two reasons for this phenomenon. Firstly, time complexity of SCTDSA is lowered to $O(\sqrt{n})$ through service clustering. Secondly, service performance information needed by SCTDSA is evaluated in advance. SCTDSA uses service information directly. However, QoS guided Min-min Heuristic spends much time in calculating the minimum completion time of each task for each resource. Hence, through service clustering, our algorithm can save much time and more fit for the large-scale Grid environment.

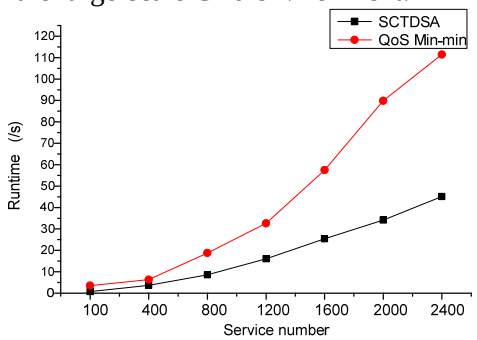

Figure.2 Comparison of scheduling runtime between SCTDSA and QoS guided Min-min Heuristic

(2) Compared with EMPEROR with different prediction accuracy

As shown in Figure. 3, a comparison is made between SCTDSA and EMPEROR with prediction precision of 0.8, 0.85 and 0.9 on makespan and SRR. When the prediction precision is 0.8, SCTDSA has greater advantage. It saves 26% makespan and reduces 53% SRR compared with EMPEROR, because the target resource chosen by EMPEROR may be heavy-loaded due to its prediction inaccuracy. With increasing of the prediction precision, the difference between the two algorithms becomes smaller. Makespan (SRR) of SCTDSA is only 15% (34%) less than that of EMPEROR when prediction precision is 0.9. However, even prediction precision of EMPEROR reaches maximum, SCTDSA remains better than EMPEROR, because our algorithm considers service stability. The main reason for service rejection caused by SCTDSA may be that user QoS requirement is too tight. Thus, our algorithm provides better QoS than EMPEROR and the effectiveness of our performance evaluation method is validated.

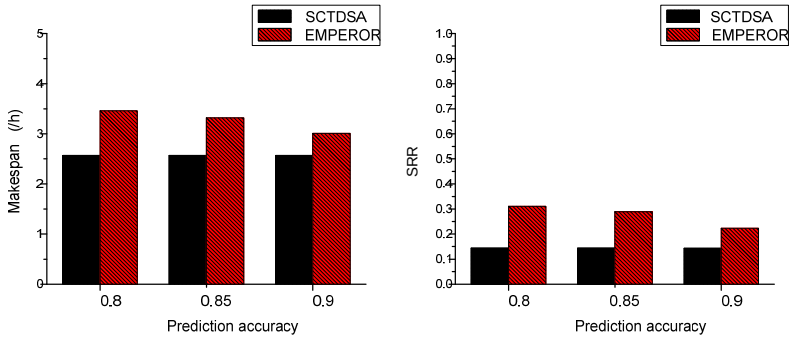

Figure. 3 Comparison between SCTDSA and EMPEROR with different prediction precision

(3) Compared with QoS guided Min-min Heuristic under different workload

Figure. 4 illustrates comparison between SCTDSA and QoS guided Min-min Heuristic under different workload on makespan and SRR. W_1, W_2 and W_3 denotes different workload from light to heavy. When the workload is heavier (W_3), the advantage of SCTDSA is obvious. It decreases 30% makespan and 56% SRR of QoS guided Min-min Heuristic, because Sun's algorithm makes scheduling decision according to static performance of resource. While in a light-loaded condition, the superiority becomes smaller. Makespan only drops by 11% and SRR drops by 39%. Assuming that all resources are absolutely unloaded, our algorithm may have little advantage. However, this assumption is really too restrictive. Therefore, our algorithm adapts to the dynamic Grid environment.

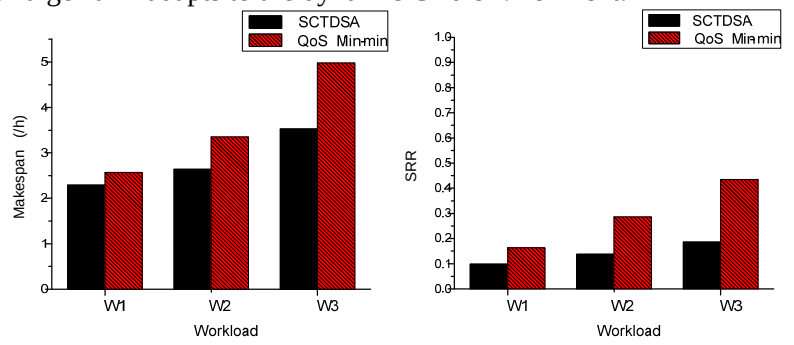

Figure. 4 Comparison between SCTDSA and QoS guided Min-min Heuristic under different workload

(4) Impact of related clustering parameters on scheduling results

The impact of the amount of service clusters on makespan and SRR is demonstrated in Figure.5. The number of *CSC* is set to $0.6\sqrt{n}$, $0.8\sqrt{n}$, $\sqrt{n}$, $1.2\sqrt{n}$ and $1.5\sqrt{n}$. When the services are clustered, the dispersion of a cluster is measured (see dispersion axis). We can see that, when *CSC* number is smaller than $\sqrt{n}$, with decreasing of *CSC* number, the makespan and SRR ascend fast, though, the candidate resource for scheduling is less; when *CSC* number is larger than $\sqrt{n}$, with increasing of *CSC* number, the makespan and SRR descend smoothly, though, the candidate resource for scheduling is more. Therefore, there is trade-off between scheduling complexity and QoS. And the balance point is roughly $\sqrt{n}$ discovered through extensive simulations, which conforms to conclusion of literature [22]. Under the optimal condition, the scheduling complexity of our algorithm is lowered to $O(\sqrt{n})$ and it provides better QoS guarantee than previous ones.

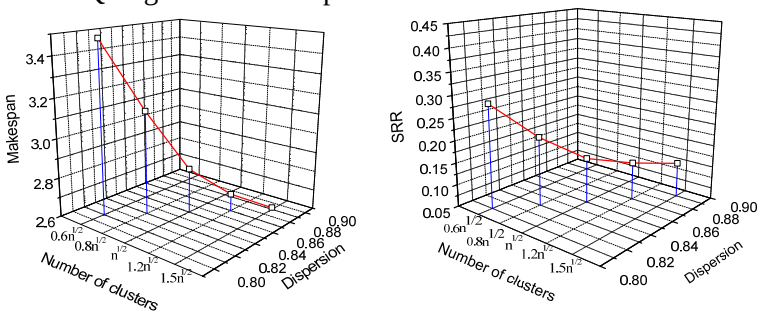

Figure.5 Impact of clustering parameters on SCTDSA

9. Conclusion

Reducing problem scale is an effective solution for solving task scheduling in Grid environment. We evaluate dynamic service performance on condition that local scheduling policy of Grid site is respected, and obtain performance metrics of dynamic service. According to dynamic service evaluation, an adaptive and dynamic service clustering method is proposed in light of swarm intelligence. Then the services with similar or same QoS are clustered. A concise dynamic meta-task scheduling algorithm is presented from service clustering. Basing on implemented prototype, we simulate our algorithm and compare it with two related algorithms. The experiment results demonstrate that our algorithm features low time complexity and good QoS guarantee. We focus our future works on further performance evaluation considering relationship among *Ex*, *En* and *He* and resource communication capacity.

10. Acknowledgments

This research was supported by the National Nature Science Foundation of China under grant No. 60970038 and 60673165.

11. References

[1] I. Foster, C. Kesselman, M. Nick et al. “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” 2002. http:// www.globus.org/research/papers/ogsa.pdf

[2] O.H. Ibarra, C.E. Kim. “Heuristic Algorithms for Scheduling Independent Tasks on Nonindentical Processors,” Journal of ACM, 1977, 24(2): pp.280-289

[3] M. Wu, X.H. Sun. “Grid Harvest Service: A Performance System of Grid Computing,” Journal of Parallel and Distributed Computing, 2006, 66: pp. 1322-1337

[4] Z.G. Hu, Z.J. Hu, L. Li. “A Performance-Evaluation-based Grid Task Scheduling Algorithm,” Journal of Simulation, 2009, 21(12): pp.3542-3548

[5] Z.J. Hu, Z.G. Hu, Z.H. Liu. “Resource Availability Evaluation in Service Grid Environment,” 2007 IEEE Asia-Pacific Services Computing Conference (APSCC'07), Tsukuba Science City, Japan, 2007, pp. 232-238

[6] Z.G. Hu, Z.J. Hu. “Prediction-based Resource Matching Algorithm,” Computer Application, 2007, 27(10): pp. 2391-2394

[7] V. Berten, J. Goossens, E. Jeannot. “On the Distribution of Sequential Jobs in Random Brokering for Heterogeneous Computational Grids,” IEEE Transactions on Parallel and Distributed System, 2006, 17(2): pp. 113-124

[8] X. Ren, S. Lee, R. Eigenmann et al. “Prediction of Resource Availability in Fine-Grained Cycle Sharing Systems Empirical Evaluation,” Journal of Grid Computing, 2007, 5: pp. 173-195

[9] F. Dong, S. G. Akl. “PFAS: A Resource-Performance-Fluctuation-Aware Workflow Scheduling Algo- rithm for Grid Computing,” 21st Parallel and Distributed Processing Symposium (IPDPS'07), Long Beach, Canada, 2007, pp. 1-9

[10] M. Wu, X.H. Sun. “Memory-Conscious Task Partition and Scheduling in Grid Environments,” Proceeding of Fifth IEEE/ACM International Workshop Grid Computing (Grid'04), 2004, pp. 138-145

[11] S. Viswanathan, B. Veeravalli, T. G. Robertazzi. “Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems,” IEEE Transactions on Parallel and Distributed Systems, 2007, 18(10): pp. 1450-1561

[12] Y. Zhang, A. Mandal, H. Casanova et al. “Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling,” Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), Singapore, 2006, pp. 568-575

[13] W. Zhang, B. Fang et al. “Multisite Resource Selection and Scheduling Algorithm on Computational Grid,” Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, 2004, pp. 105

[14] R. Chu, X. Lu, N. Xiao. “A Clustering Based Resource Aggregation Method for Virtual Computing Environment,” Journal of Software, 2007, 18 (8): pp. 1858-1869

[15] Y. Derbal. “A Model of Grid Service Capacity,” Journal of Computer Science and Technology, 2007, 22(4): pp. 505-514

[16] D. Li. “Artificial Intelligence with Uncertainty,” National Defense Industry Press, 2005, pp. 171-177

[17] D. Li, C. Liu, Y. Du, X. Han. “Artificial Intelligence with Uncertainty,” Journal of Software, 2004, 15(11): pp. 1583-1594

[18] J. Kennedy, R. Eberhart. “Particle swarm optimization,” Proceeding of the IEEE International Conference on Neural Networks, Perth Australia, 1995, pp. 1942-1948

[19] R. Eberhart, Y. Shi. “Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization,” Proceeding of the 2000 Congress on Evolutionary Computation. Piscataway, 2000, pp. 84-88

[20] T. Ioan Cristian. “The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection,” Information Processing Letters, 2003, 85(6): pp. 317-325

[21] S.L. Yang, Y.S. Li, X.X. Hu et al. “Optimization Study on k Value of K-means Algorithm,” Systems Engineering--Theory&Practice, 2006, 26(2): pp. 97-101

[22] D. Lu, P. A. Dinda. “Synthesizing Realistic Computational Grids,” ACM/IEEE Supercomputing 2003 Conference (SC'03), Phoenix, USA, 2003, pp. 16

[23] U. Lublin, D. G. Feitelson. “The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs,” Journal of Parallel Distributed Computing, 2003, 63: pp. 1105-1122

[24] X. He, X.H. Sun et al. “QoS Guided Min-Min Heuristic for Grid Task Scheduling,” Journal of Computer Science and Technology, 2003, 18(4): pp. 442-451

[25] L. Adzigogov, J. Soldatos, L. Polymenakos. “EMPEROR: An OGSA Grid Meta-Scheduler Based on Dynamic Resource Predictions,” Journal of Grid Computing, 2005, 3: pp. 19-37

[26] C. S. Yeo, R. Buyya. “Pricing for Utility-driven Resource Management and Allocation in Clusters,” International Journal of High Performance Computing Applications, 2007, 21(4): pp. 405-418

Authors

Zhoujun Hu

He received the M.S and Ph.D degrees in Computer Science from Central South Univ. in 2007 and 2010 respectively. His research fields include grid computing, resource performance evaluation and meta-scheduling.

Zhigang Hu

He is a Ph.D supervisor and professor of Central South Univ. in P.R.C. He received the M.S and Ph.D degrees in Computer Science from Central South Univ. in 1988 and 2002 respectively. His research fields include parallel and distributed computing, grid computing and cloud computing.

Zhenhua Liu

He received the B.S and M.S degrees in Computer Science from Central South Univ. in 2006 and 2009 respectively. He is currently a Ph.D. student of Department of Computer Science and Engineering in Univ. of South Carolina. His research fields include wireless networking and its security, location privacy.