Dynamic Resource Selection For Service Composition in The Grid

William K. Cheung[+], Jiming Liu[+], Kevin H. Tsang[+], Raymond K. Wong[++]

Department of Computer Science[+]
Hong Kong Baptist University
Hong Kong
{william,jiming,hhtsang}@comp.hkbu.edu.hk

School of Computer Science & Engineering[++]
University of New South Wales
Australia
wong@cse.unsw.edu.au

Abstract

While numerous efforts have focused on service composition in the Grid environment, service selection among similar services from multiple providers has not been addressed. In particular, all service composition work done so far are based on a given selection of services under a well set environment. As a result, uncertainty (e.g., server load, network traffic, computation time of the services due to changing memory and other unexpected conditions) under a real, dynamic environment has never been considered. This paper prototypes the service selection under a Grid environment and proposes an uncertainty framework to address the issue. Experimental results show that our considerations are valid and our preliminary solution works well in our Globus Grid network.

Keywords: Services composition, resource and performance ontologies, Grid services, uncertainty management

1. Introduction

Although Web services are becoming the backbone for electronic business and interoperable applications, standards such as WSDL [1], UDDI [2], and SOAP [3] do not address the issues of service re-use and composition, especially dynamic composition of existing services from multiple sources. Various efforts on addressing this issue including the recent initiative of BPEL4WS [4] focus on representing compositions but they do not address the actual process of selecting and composing the services. Furthermore, the benefits of embracing Web services on Grid [5] have been recently realized in the Open Grid Services Architecture (OGSA) of Globus (GT3) – the de-facto standard of Grid middleware [6]. Since Grid platform is in general more conscious regarding the utilization and reliability of resources, services composed in Grid need to be planned in an optimized way. This problem is even more challenging if various uncertainties caused by changing server load, network traffic, memory conditions and hence runtime for each services are considered. Along this line and inspired by the resource matching proposal [16], this paper attempts to investigate the uncertainties during service selection and composition, propose an initial solution and finally realize its significance by implementing it (called BU-Grid) and running series of experiments.

1.1 Related Works

Due to the increasing attention to Web services from the research and industry communities, there have been lots of recent works addressing various issues of Web services (e.g., [7]). To name a few, for example, in [8], the issue of service composition is addressed in the context of Web components, as a way for creating composite Web Services by re-using, specializing and extending existing ones. McIlarith and Son [9] proposed an approach to building agent technology based on the notion of generic procedures and customizing user constraints. They argue that an augmented version of the logic programming language Golog provides a natural formalism for programming Web services. Prototypes that guide a user in composing Web services in a semi-automatic manner have been proposed in [10,11]. The semi-automatic process is facilitated by presenting matching services to the user at each step of a composition and filtering the possibilities by using semantic descriptions of the services. While there are numerous papers describing specifications and methods for service composition, seldom of them have addressed the issues of choosing services based on its costs and resources (which is an important issue in utilizing resources in a Grid environment). For instance, [12] mentioned a simple scoring service based on the summation of the services' weighted scores. However, the details of estimating the scores and evaluating criteria (which are crucial in the actual implementation and system evaluation, again, especially in Grid) have been left out. Blythe *et al*., in

[13], used limited state information (the current data storage of the distributed hosts) for optimizing services compositions for e-Science applications. The work closest to ours is due to Sample *et al.* [14] that incorporated services uncertainty (e.g., costs, performance, reliability) via probabilistic modeling in the composition process.

1.2 Paper Organization

The remaining of the paper is as follow. Section 2 gives a typical environment for autonomous services composition. Section 3 describes in detail the overall system architecture of BU-Grid. Section 4 provides in detail some bidding mechanisms for services selection in a dynamic Grid environment. Experimental results and the lessons learnt are found in Section 5 and 6, respectively. Section 7 concludes the paper with a number of future research directions.

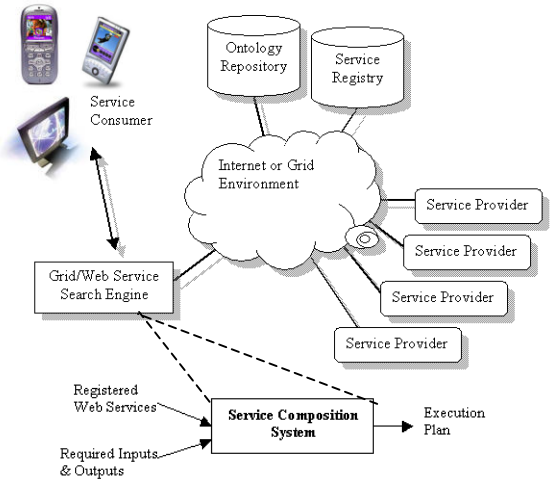


Figure 1. A typical service composition environment.

2. Dynamic Services Composition

2.1 A General Environment

A typical environment for supporting Grid/Web service composition is illustrated in Figure 1. A collection of *service providers* expose, via the Internet, the services they support as Web services. The services are *registered* at a *service registry* (e.g. UDDI) for *service discovery*. The semantics of the available Web services (e.g., the semantics of the input/output parameters) are described by some machine understandable semantic Web language (e.g. OWL-S). Relationships and concepts of the vocabularies used to enable semantic matching of services are shared in an *ontology repository*. A *service consumer* is a client program which sends service requests (e.g., in terms of desired input/output relationships) to the Grid/Web service broker which bears the duty of selecting suitable primitive services, composing them as well as monitoring their execution.

2.2 An Illustrated Example: e-Finance Services

Suppose there exists the need of decision support for stock trading which involves a typical service flow (Figure 2) from clarifying the specific goal of the user – *Service-1*, to fetching stock information from heterogeneous information sources – *Service-2*, to analyzing the information for a summarized report – *Service-3*, and eventually to buying stock on-line – *Service-4*. In the service-oriented computation market, different companies start providing implementations of the services with different degrees of performance and cost. The service consumers then compare and select specific implementations for each service along the flow. As the e-Service industry develops, the demand of the services will become more complicated and diversified. Some services used to be provided by one company start splitting for further specialization and some used to be separated start merging for providing one-stop "canned" consulting services. Dynamic composition of a suitable plan out of the set of diversified services available in the computational market becomes non-trival and the architecture proposed in this paper is for addressing this issue.

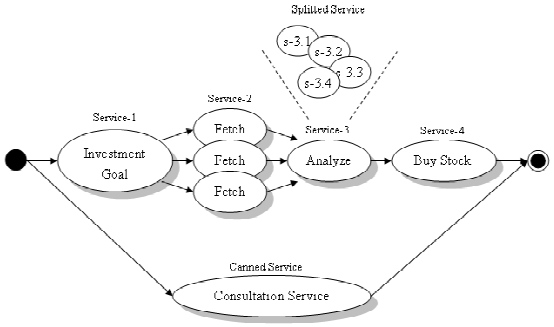


Figure 2. A particular e-Finance service flow.

2.3 Uncertainties in The Grid Environment

To contrast with the traditional distributed computing environments, the Grid aims to coordinate *dynamically* resources distributed in the Internet which is an environment with its performance subject to diverse sources of uncertainties (e.g., transient system load and network bandwidth, dynamically cached data files, etc.). In addition, the dynamic coordination requirement results in a great need for Grid related systems to maintain the trust relationship among the distributed resources.

3. BU-Grid System Architecture

The architectural design of the proposed BU-Grid, to be further described in the following (also see Figure 3 for an overview), contains components that are common in most of the service composition systems. In addition, it is featured by the incorporation of a) bidding services and bid evaluation components for dynamic service selection, as well as b) a plan base and a plan retriever for plan re-use support.

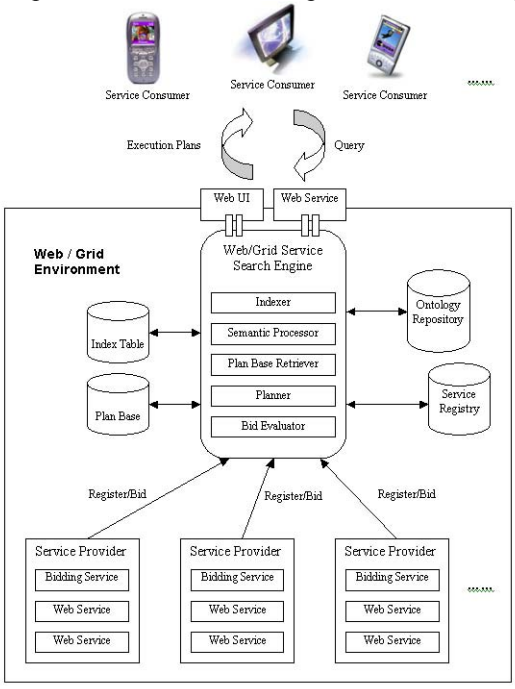


Figure 3. The system architecture of BU-Grid.

3.1 Service Registration and Indexing

Semantic descriptions of Grid services are stored at the Service Registry, which may include:

- High-level services descriptors: E.g., for e-business applications, they can be company name, business nature/categories, contact person, phone number, email address, etc.
- Low-level services interface descriptors: E.g., service name, functional description, URL of the WSDL file or Grid Service Handle (GSH), semantics of the input/output parameters, etc.

To support efficient access of GSHs from the Service Registry and efficient update of the services' state

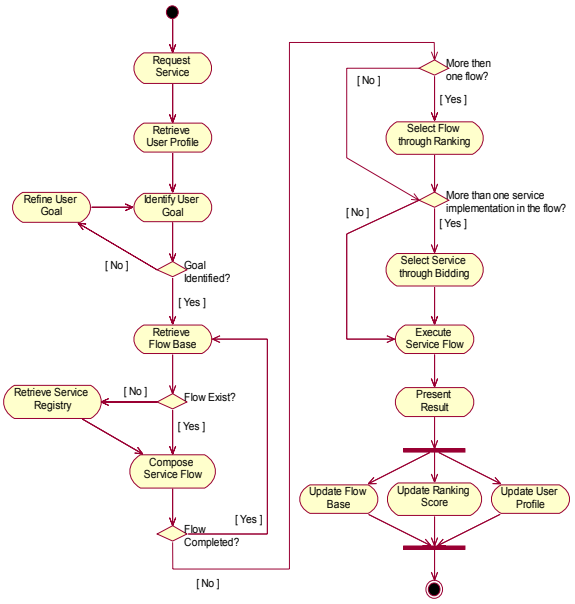


Figure 4. Activity diagram of service request use case.

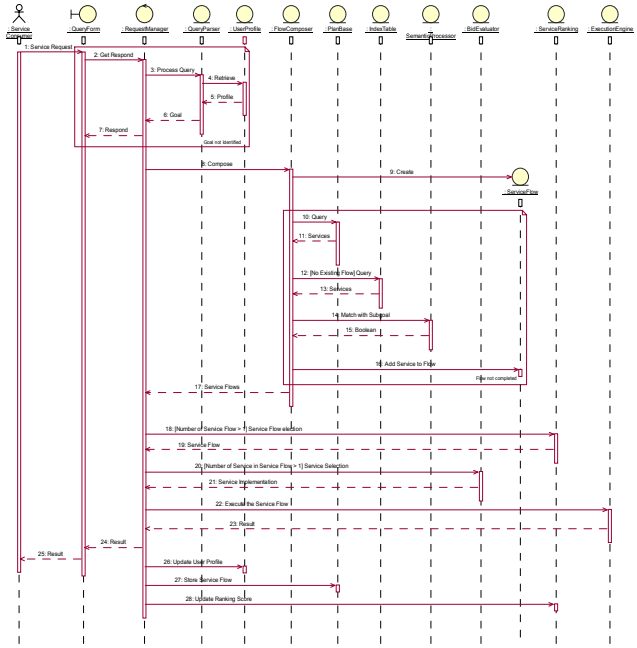


Figure 5. Sequence diagram of service request use case.

information, both the high-level and low-level service semantics are indexed and categorized. Furthermore, to extend the service discovery capability to go beyond the simple keyword-based approach, different domain-specific ontologies are maintained in Ontology Repository to support semantic matching.

3.2 Task Specification & Service Composition

In BU-Grid, a task is represented by specifying the required input and desired output. To solve the task (or to satisfy the specification), a meta-level service is to be composed using the primitive services available in the Service Registry.

By treating the input as the initial state, the desired output as the goal, and the available services as the operators, service composition can readily be formulated as an AI planning problem [14]. Under the Grid context, one challenge is that the planning has to be performed in a dynamic environment, containing multiple functionally equivalent operators (services) but with possibly different implementations and time-varying resources. Besides, services matchmaking based on semantics is also a non-trivial task.

3.2.1 Services Matchmaking

To enable correct matchmaking between Grid services, we need to well-define services compatibility. There exist at least two types of compatibility measures, namely data type compatibility as well as semantic compatibility. Eq.(1) and Eq.(2) give two possible forms of compatibility in terms of data type and semantics between an output of a service and an input of a matching service.

a) Data Type Compatibility

$$Compatibility_t(type_{output}, type_{input}) = \begin{cases} 1 & same / upcast \\ 0.5 & downcast \\ 0 & otherwise \end{cases} \quad (1)$$

where "upcast" means the output has to be upcasted (e.g., from `int` to `float`) so as to be fed into the next input, and similarly for "downcast".

b) Semantics Compatibility

$$Compatability_s(semantic_{input}, semantic_{output}) = \begin{cases} 1 & equivalent \\ 0.8 & subclass \\ 0 & otherwise \end{cases} \quad (2)$$

where "subclass" means that the output is a subclass of the input and the need of ontology is explicitly implied.

3.2.2 Planning

Based on the services compatibility measures defined, planning for service composition can be proceeded using different planning paradigms. One example is regression planning which is based on backward chaining. Starting from the output of the specified task as the ultimate goal, the planner can search the Service Registry for the services with their outputs compatible with that of the specified task. It is possible that the set of compatible services can be categorized into several distinct *service interfaces,* each contains a unique input/output pair. One can then use those distinct service interfaces as sub-goals and continue to search for the best plan. Sometimes, for efficiency purpose, one may want to use a local search strategy by choosing one of the interfaces and continuing the search. The selection can be done based on a local performance estimation of the interfaces. See Figure 4 for an overview and refer to Section 6.2 for more discussion on dynamic plan optimization.

As one service interface is in fact representing a group of functionally identical services, its performance estimation should be characterized by the best service under the same interface. So, under this scenario, the remaining question is how to select the best service under the dynamic environment.

3.2.3 Service Selection

Services with identical input/output interfaces can have different implementations, time-varying system load, time-varying cached data, etc. Specifying these a) performance related and b) state related information via ontology is important to support more robust task planning.

A) Resource and Performance

Via the GT3 middleware, one can obtain state information of a Grid node from the service `MasterForkManagedJobFactoryService` (see Table 1). For network related states, they have not yet been available in GT3 and one can install the Ganglia system for obtaining them in an XML format which is based on a particular DTD for Ganglia (see Table 2 for a complete list). Based on the specific requirement of the application, different scoring schemes derived from those system state information can be adopted for service selection. Besides, some current/aggregated performance statistics obtained from the past history should also be an additional determining factor.[1]

B) Ontologies

Various proposals have been published in using RDF-Schema for ontology specification. Without re-inventing the wheels, this paper assumes the three ontologies in RDF-Schema as proposed in [16], namely resource ontology (e.g., `OperatingSystem.TotalPhysicalMemory = 512MB`), resource

[1] Note that the service cost is another orthogonal factor to be considered in a computation market, which however is not the focus of this paper.

request ontology (e.g., `MinPhysicalMemory = 1024MB`) and policy ontology (that capture the resource authorization and usage policies). The actual resource matching based on these ontology were described in [16]. Besides, we believe that performance related ones (e.g., `ResponseTime = 14min.`) should also be useful. In this paper, we propose to make use of both types of information for service selection (see Section 4).

C) Selection

With the service resource and performance ontologies ready, particular service implementation can then be wisely picked for better Grid resource utilization. In particular, a bidding-like mechanism based on a dynamic scoring scheme (Figure 4) is proposed for this service selection task, as detailed in Section 4.

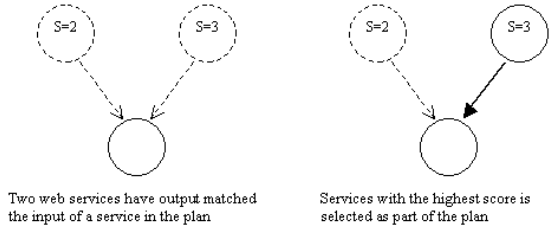


Figure 4. Service selection.

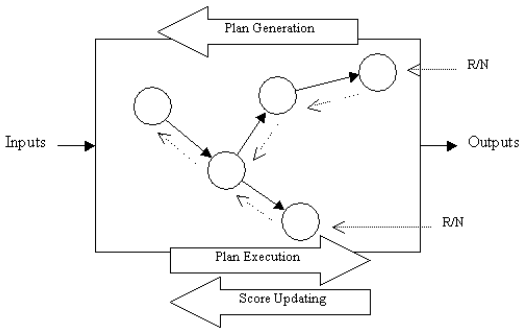


Figure 5. An overview of service composition and execution process.

4. Dynamic Service Selection Via Bidding

Here we propose a bidding-like mechanism for the aforementioned service selection problem with the hope of balancing the load among a set of Grid nodes in a virtual organization.

4.1 Notations

Let I denote a particular service interface, $E_i(I)$ denote the estimated service time of the i^{th} implementation for the service interface I, $B_i(I)$ denote the value sent to the broker by the i^{th} implementation for bidding the interface I to be performed.

4.2 Bidding Process

The broker (search engine) first notifies each of the service providers that host the required service implementations. Being notified, each service implementation will make use of the current estimated service time $E_i(I)$ as well as the current system load to compute a bid value as in Eq.(3) and send the bid back to the broker:

$$B_i(I) = (1 - L_i) \times \frac{1}{E_i(I)} \tag{3}$$

where L_i is the CPU usage of the node hosting the i^{th} service implementation. Note that L_i is a state information and $E_i(I)$ is a performance prediction.

The broker then selects a service implementation according to the probability distribution:

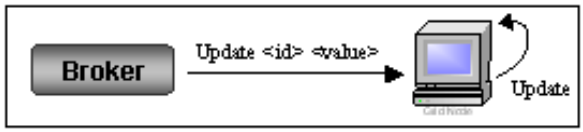
$$P(i) = \frac{B_i(I)}{\sum_i B_i(I)} \tag{4}$$

4.3 Estimation of Service Performance

After the selected implementation finished the assigned job, it will notify the broker the result. The broker will then return the actual service time A_i and the estimated service time of the i^{th} service implementation will be updated as

$$E_i^{t+1}(I) = (1 - \alpha) \times E_i^t(I) + \alpha \times A_i \tag{5}$$

where α is the updating rate. In our experiment, its value is set to 0.8. The responsiveness of the system is determined by the value of α. Such an updating rule is able to capture smooth variation of service performance.



4.4 A MiniMax Bidding Strategy

For cases where worst-case performance has to be controlled, the minimax strategy can be used for computing the bidding score. In particular, we can store the actual service times for a time window of size N (=10 for our experiment) and we take the maximum instead of average to avoid selecting node with large

service time fluctuation. The formula for the bidding score can be reformulated as

$$B_i(I) = (1 - L_i) \times \frac{1}{\max_{t \in \{i, i-1, \ldots i-N+1\}} (A_t)} \quad (6)$$

Other than the minimax scheme, one can also compute directly the standard deviation as uncertainty and incorporate it into the bidding score (later on called the uncertainty scheme), for example,

$$B_i(I) = (1 - L_i) \times \frac{1}{E_i(I) + \beta s_i} \quad (7)$$

where

$$s_i = \sqrt{\frac{1}{N-1} \sum_{t=i-N+1}^{i} \left(A_t - \overline{A_t}\right)^2}$$

and β is used for controlling the degree of tolerance for performance fluctuation.

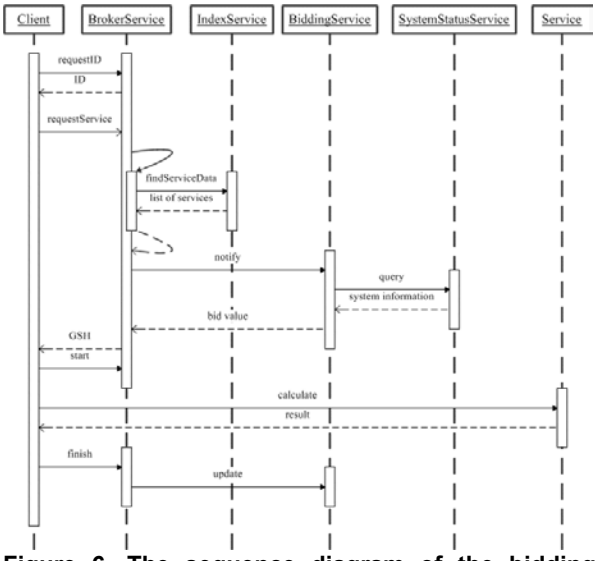


Figure 6. The sequence diagram of the bidding process.

5. Experiments

In order to study in detail the effectiveness of the proposed bidding process on the Globus platform and the behavior at each grid node, we have set up a small grid environment with six grid nodes, one being the Service Broker and the other five being the Service Providers. Their properties are shown in Table 1. Figure 6 shows the sequence diagram of the overall bidding process. All of the Grid services are running in the service container provided by GT3. The BrokerService queries the IndexService of each Grid node to get the list of available service implementations. BiddingService consults SystemStatusService of its own node to get the current system information. Three experiments have been conducted for evaluating different virtual organization scenarios on the grid platform. Without loss of generality, we assume homogeneous service type for all the grid nodes for testing the service selection process. For experiments related to heterogeneous service types and composite types, readers are referred to one of our previous works [17].

	node-1	node-2	node-3	node-4	node-5
CPU	P4 2.4GHz	P3 .7GHz	P3 .65GHz	P3 .65GHz	P3 .43GHz
RAM size	512MB	256MB	256MB	256MB	256MB

Table 1 The properties of the five grid nodes used in the experiments.

Service Time	Single Node	Three Nodes	Five Nodes
Minimum	17.6s	17.5s	18.1s
Maximum	23.3s	49.7s	96.2s
Average	19.0s	31.8s	41.3s
Total	1047.7s	704.9s	552.1s

Table 2. Service time for different number of nodes in the Grid environment.

5.1 The Effect of Increasing Number of Nodes

The first experiment tries to test the overall gain when an increasing number (1, 3 & 5) of Grid nodes are being used during the bidding process. The inter-arrival time for the service requests is 5 seconds. The results are tabulated in Table 2. It is noted that the overall service time is significantly reduced and individual service request should experience shorter latency but occasionally longer service time. The corresponding job schedules of the three cases are shown in Figures 7-9 and the distribution of the job assignments for the case with five nodes is shown in Figure 10. Note that the job requests can successfully be assigned to the five nodes according to their computational resources.

5.2 Adaptability Towards Unexpected Loading

The second experiment tries to test the effectiveness of the proposed bidding process for adaptive balancing of the service requests under the situation with some unexpected loading experienced by some of the Grid nodes. We used five nodes for this experiment. In particular, node-1 and node-2 are set to experience additional loading from 100 sec to 300 sec and from 400 sec to 500 sec, respectively; starting from the time the first service request arrives. The situations are

highlighted in Figure 12. The inter-arrival time is 5 sec. By comparing the distribution of the job assignments between the case with unexpected loading (Figure 13) and that without the loading, it is noted that more job assignments are shifted from node-1 and node-2 to the others via the bidding process.

5.2 Adaptability Towards Service Reliability

The third experiment tries to test the effectiveness of the use of the minimax scheme and the uncertainty scheme. For the experiment setting, we only used three nodes with similar computational power (i.e., node-2, node-3, & node-4) to clearly demonstrate the effect and programmed node-2 to have a large fluctuation in service time to simulate the situation of an unreliable service provider. The inter-arrival time for this experiment is 20 seconds. First, we tried a uniformly distributed fluctuation within the range of [-10,10] sec. for the service time of node-2. With the use of the proposed uncertainty scheme with β=3, the bidding process can successfully assign less jobs to node-2 which is supposed to be less reliable in terms of performance, as shown by comparing Figures 15 and 17. The minimax scheme, however, is not very effective for that scenario. Then, we replaced the uniform fluctuation by a binomial process with {+10, -10} the only possible outcomes (so much more severe than the uniform one). Under this scenario, the minimax can effectively migrate the job assignments to those with more stable service performance.

6. Discussion and Future Works

6.1 Semantic Service Matching

Semantics based service matching, such as those ontology-based matchmaking techniques, has been active in the domain of Web services. Recently, researchers have tried to extend the techniques to Grid services. Since Grid is resource conscious, resource consideration during service matching is useful and important (e.g., [16]). Our ongoing work is to extend the techniques presented in this paper into the ontology-based framework to facilitate a better and more practical service matching in Grid.

6.2 Dynamic Plan Optimization

The next obvious step of this work is to integrate the bidding mechanism one step upward to the planning step. By assuming that each Grid service interface keeps a table of scores *S* to indicate its desirability to use some other services, where the scores can be some statistics computed during the bidding for services selection (Section 4). Then, the setup will be similar to that of the PageRank algorithm [15] used by Google search engine for indicating Web page importance. For example (see Figure 5), let *R* denote the reward for a selected plan (can be a constant equal to, say, 1), *N* denote the number of the outputs of the specified task, *n* denotes the current updating service interface, *m* denotes the service interfaces that use the output of current service interface *n,* and α denotes the updating rate (can be a constant equal to some value less than 1). For service interfaces with their outputs form the outputs of the specified task (i.e., the ultimate goal),

$$S_n^{t+1} = (1-\alpha)\cdot S_n^t + \alpha \cdot \frac{R}{N}$$

Then, for the subsequent planning steps,

$$S_n^{t+1} = (1-\alpha)\cdot S_n^t + \alpha \cdot \sum_m S_m^t$$

Such a scoring scheme implies implicitly that frequently selected (good track records) service interfaces will be updated more frequently. Also, those interfaces often appear near to the final output of the selected plans (bringing you faster to the goal) will have higher scores. Also, those interfaces provide more outputs (more resourceful) will have a higher score. We are currently studying the effectiveness of such a scoring scheme.

6.3 Plan Base

Performing service composition from scratch can be a time-consuming process for time-critical applications. One can use a plan base for storing plans that have been executed. A similar idea has been echoed in [13]. The archived plans (as some options of pre-composed services) can then be used for the construction of new plans. The reuse of plans should be can increase the efficiency of plan construction. For better use of the storage resource, there can also be some related policies for deleting plans that appear obsolete.

7. Conclusion

This paper focused on service selection under a dynamic environment, i.e., with various uncertainties due to changing server loading, network traffic etc. Although the issue is critical and practical, it has been disregarded by previous works in Web service composition. While Web services embraced in Grid platforms is getting popular, we demonstrated that service selection could make significant performance and resource utilization differences during service composition in the Grid, especially in a dynamic environment. Although the experimental results were encouraging, we believe that further investigation on selecting services for large scale service composition

will encourage more Web service usages, especially for Grid environments where resource utilization and service performance are concerned.

Acknowledgement

This work is supported by Centre for E-Transformation Research, Hong Kong Baptist University under the RGC Group Research Grant (HKBU 2/03/C).

References

1. WSDL, http://www.w3.org/TR/wsdl
2. UDDI, http://www.uddi.org
3. SOAP, http://www.w3.org/TR/SOAP/
4. BPEL4WS, http://www-106.ibm.com/developerworks/library/ws-bpel/
5. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, Vol. 15, pp. 200-222, 2001.
6. I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, "Grid Services for Distributed System Integration," IEEE Computer, June 2002.
7. IEEE Internet Computing, Special issue: Middleware for Web services, 2003.
8. J. Yang and M. Papazoglou, "Web components: A substrate for web service reuse and composition," Advanced Information Systems Engineering, Proceedings of the 14th International Conference, CAiSE 2002 Toronto, Canada, May 27-31, 2002.
9. S. McIlraith and T. Son, "Adapting Golog for composition of semantic Web services", Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning, 2002.
10. E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic composition of Web services using semantic descriptions," Proceeding of Web Services: Modeling, Architecture & Infrastructure workshop in ICEIS, April 2003
11. L. Chen, N.R. Shadbolt, C. Goble, F. Tao, S.J. Cox, C. Puleston, P.R. Smart, "Towards a Knowledge-based Approach to Semantic Service Composition," 2nd International Semantic Web Conference (ISWC2003), 20-23 October 2003, Florida, USA, Lecture Notes in Computer Science, LNCS 2870, pp 319-334.
12. B. Benatallah, Q. Sheng, and M. Dumas, The Self-Serv environment for Web services composition, in *IEEE Internet Computing*, pages 40--48, 7(1), 2003.
13. J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, K. Vahi, "The Role of Planning in Grid Computing," Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS), June 9-13, 2003, Trento, Italy.
14. N. Sample, P. Keyani, G. Wiederhold, "Scheduling Under Uncertainty: Planning for the Ubiquitous Grid," Proceedings of the Fifth International Conference on Coordination Models and Languages (Coord2002).
15. R. Motwani, S. Brin, L. Page, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford Digital Libraries Working Paper, 1998.
16. H. Tangmunarunkit, S. Decker, C. Kesselman, "Ontology-based Resource Matching in the Grid – The Grid meets the Semantic Web", Workshop on Semantics in P2P and Grid Computing, May 2003.
17. W. Cheung, J. Liu, K. Tsang, R. Wong, "Towards Autonomous Service Composition in A Grid Environment," to appear in Proceeding of 2004 IEEE International Conference on Web Services, San Diego, California, July, 2004

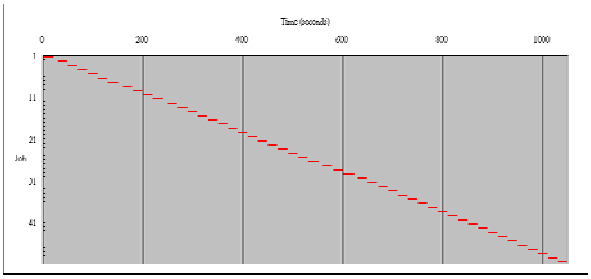


Figure 7. Job schedules with one node.

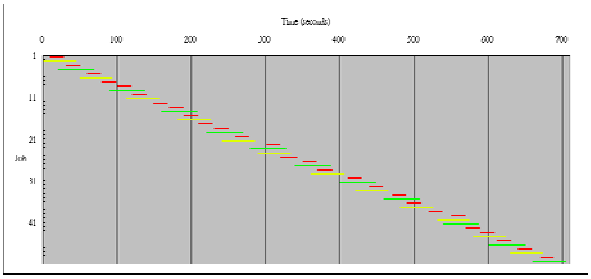


Figure 8. Job schedules with three nodes (red for node-1, yellow for node-2, green for node-3).

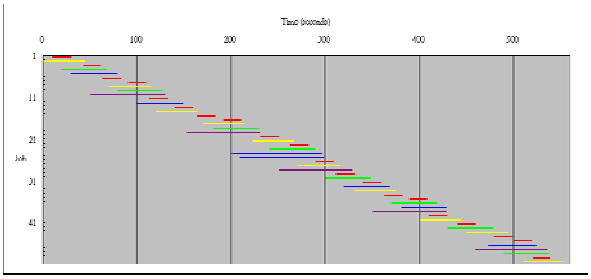


Figure 9. Job schedules with five nodes (red for node-1, yellow for node-2, green for node-3, blue for node-4, purple for node-5).

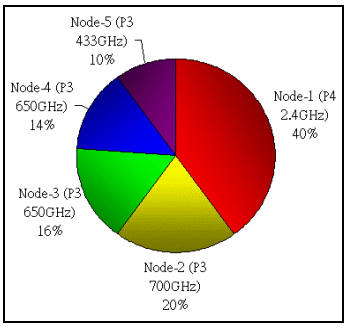
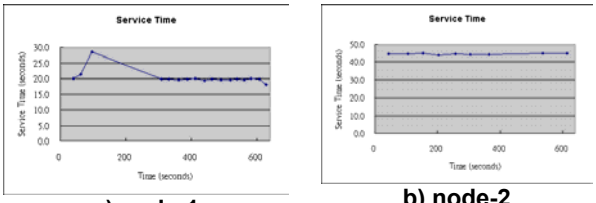
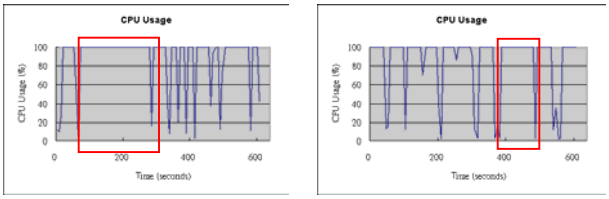


Figure 10. Job assignment distribution with five nodes.



a) node-1 **b) node-2**

Figure 11. Job service time for node-1 and node-2 under system load variation scenario.



a) node-1 **b) node-2**

Figure 12. CPU usage for node-1 and node-2 under system load variation scenario.

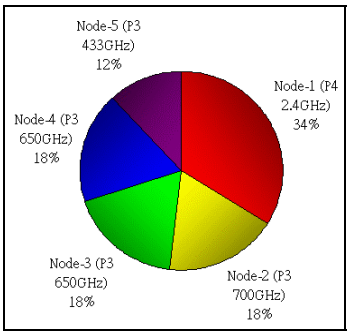
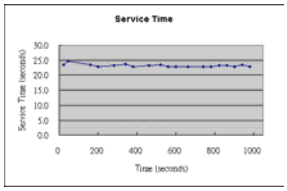
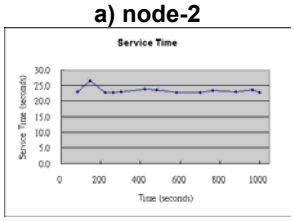


Figure 13. Job assignment distribution with five nodes under the unexpected system loading scenario.

a) node-2



b) node-3



c) node-4

	Average
node-2	24.1s
node-3	23.1s
node-4	23.4s

Figure 14. Job service time for node-2, node-3 and node-4 under unreliable service scenario with Eq. (5) used as the updating rule.

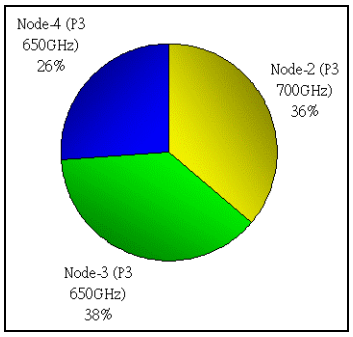
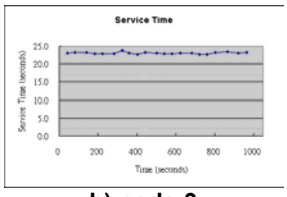
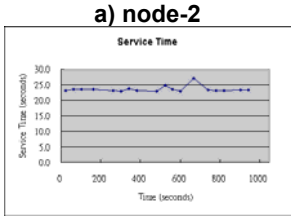


Figure 15. Job assignment distribution under unreliable service scenario with Eq. (5) used as the updating rule.

a) node-2



b) node-3



c) node-4

	Average
node-2	23.8s
node-3	23.0s
node-4	23.4s

Figure 16. Job service time for node-2, node-3 and node-4 under unreliable service scenario with the uncertainty scheme used.

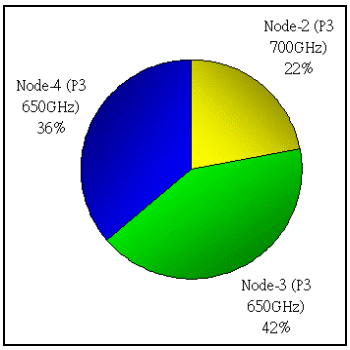
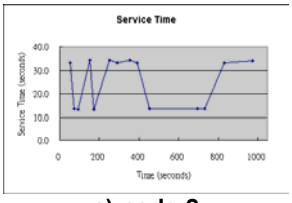
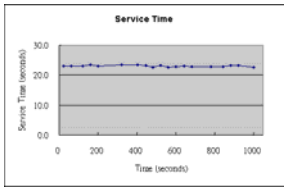


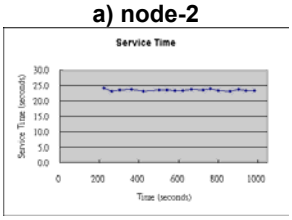
Figure 17. Job assignment distribution under unreliable service scenario with the uncertainty scheme used.



a) node-2



b) node-3



c) node-4

	Average
node-2	25.1s
node-3	23.0s
node-4	23.4s

Figure 18. Job service time for node-2, node-3 and node-4 under unreliable service scenario with the minimax scheme used.

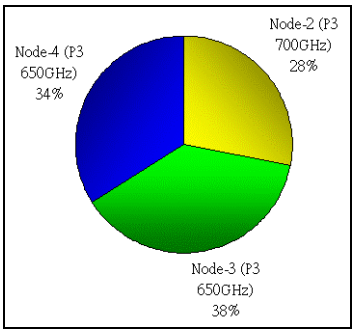


Figure 19. Job assignment distribution under unreliable service scenario with the minimax scheme used.