QoS-oriented Web Service Framework by Mixed Programming Techniques

Changsong Liu, Dongbo Liu
School of Computer and Communication, Hunan Institute of Engineering, Xiangtan, 411104, China
Email: liuchangsong74@126.com

***Abstract*—Web service based applications have been widely applied in various kinds of domains with the development of service-oriented architecture. However, service selection and composition under user's QoS constraints still remains to be a challenging issue because of the changing of user's requirements. In this work, we present a QoS-oriented web service framework and its implementation, which is aiming to optimize the performance of service-based application with constraints to user's QoS requirements. In this framework, the optimum mapping between abstract web services and application's processes is implemented through mixed programming technique. In addition, an embedded QoS negotiation mechanism is also implemented in this framework for refining the execution of service-based application at runtime. Massive experiments based on real workloads are performed to evaluate the effectiveness of the proposed framework in both static environment and dynamical environment. The results indicate that the proposed framework and its service selection/composition algorithm can significantly improve the user's QoS satisfaction in terms of five most-mentioned QoS parameters.**

***Index Terms*—service selection, mixed programming, workflow, heterogeneous systems**

I. INTRODUCTION

Conventionally, web services are defined as autonomous components that can be located, advertised, and queried by standard protocols such as XML [1, 2]. With the development of network-based service computing, web service systems are creating many opportunities for various domains to cooperate with their potential consumers or business partners. In service-based platforms, complex applications are generally formulized as processes which invoking distributed services and compositing them in dynamical manner [3]. As a set of functionally equivalent services may implement the same functionality, more importantly, applications are required to discriminate those candidates based on their quality of service (QoS) requirements. So, selecting the most suitable set of services available at execution time and compositing them with desirable performance and costs plays a key role in the web service systems [4, 5].

In typical web service systems, service selection and composition is often implemented by mapping the running activities to the best set of candidate services [6]. Unfortunately, such approaches can only guarantee the local QoS constraints such as the cost of a service [7]. To overcome these shortcomings, researchers have proposed many global-optimization techniques with aiming to satisfying the performance constraints as well as application's preferences [8,9,13,15]. Nevertheless, such approaches will result in significantly increasing in terms of computational complexity compared with those local solutions. At the same time, the performance of global optimization approaches tend to be unpredictable because the workloads on the target system are often fluctuating and unpredictable in runtime [16, 17]. For example, when a business application is required to run for long period, selected web services might adjust their QoS attributes even in the duration of execution, some of them even can be un-available any more. Therefore, dynamic and adaptive technique for service selection and composition is required in such scenarios, which requires taking into account the runtime changes in terms of service's QoS attributes.

In this work, we proposed an integrated framework which can optimize the performance of service-based application with constraints to user's QoS requirements. At the same time, the proposed framework also allows the system flexibly deploy the underlying services with constraints to resource provider's expectation. In our framework, the optimum mapping between abstract web services and application's processes is implemented through programming technique [18]. In addition, QoS negotiation functionality is also implemented for providing better service through Service-Level-Agreement (SLA) negotiation.

II. RELATED WORK

Conventionally, service selection and composition can be categorized into 3 classes: automatic, semi-automatic, and manual. As to the manual approaches, the process of service composition is described by using the standard service description language such as BPEL4WS [2], and this procedure often requires that the designers have the knowledge of the target domain. Clearly, it is a labor-intensive and error-prone job, which is not appropriate for the large-scale applications. In the studies of [6,12,19], some semi-automatic approaches have been presented with aiming to deal with the problems of manual approaches.

Multi-agent based service selection and composition is one of the popular techniques and has be widely studied in many previous studies. In [15], an agent-based service composition model is designed and implemented for

doi:10.4304/jcp.8.7.1763-1770

automatic service selection and composition. To support adaptive service computing, a context-oriented service composition approach is presented in [14], which is incorporated with a adaptive mechanism for dynamically adjusting the composition parameter when the execution context is changed. In [22], an agent-based workflow model is presented with aiming to supporting collaboration for multiple enterprises. In this model, a set of software agents are designed for taking participant in the conversations with other peers so as to obtain global agreements in terms of user's performance requirements as well the system's performance constraints.

Recently, dynamic Web service composition has attracted many attentions by many researchers, including planning-based service composition and process-based composition optimization [8,16,19]. The former approaches often investigate the problem of service selection by translating the desirable performance into certain utility functions and then solve these problems through heuristic policies. The later approaches tend to solve the problem from the perspective of specific domains, which makes them able to obtain optimal performance for specific applications.

In many practical web systems, semantic-based web has been widely deployed because of its automatic and flexible features. For instance, the service composition process can be automatically programmed by a domain-level specification, which only contains the required functionality of the given domain. For example, study in [13] presented an integrated service framework, which uses the assertions of XSAL languages to specify the application's objectives and QoS requirements. Similarly, the study of [8] proposed a semantic service planning technique, which also constructs large-scale web application by applying contingency planning technique. Nevertheless, the semantic-based approaches often have very high complexity in terms time and space. So, many of these approaches resort to approximate technique for obtain the suboptimal solution by certain heuristic-based techniques.

In order to execute large-scale BPEL processes based applications in high-performance distributed systems, i.e. grid or cloud, workflow technique has been widely investigated for implementing service composition [24]. However, workflow applications are often not enable to perform QoS negotiation as well SLA mechanism on per-service basis. Therefore, it can only be applied in relative closed systems, instead of Internet-oriented open systems. In our paper, we propose an optimization technique that is applicable to the abovementioned application with fined-grained QoS negotiation mechanism.

III. The Framework and Descriptions

The architectural framework of a typical service-based system is presented in Fig. 1. In such a framework, there are three critical components including web services, service broker, and service compositor. The broker is designed for allowing providers to register their services specifications onto the UDDI registry, in which each service is described in terms of their functionality, capability, performance, and other QoS metrics. The service compositor is designed for application's service selection and execution engine. When a service-based application instance is initiated, the execution planner interacts with the service broker for retrieving candidate services, then it generates an abstract execution plan for upper-level applications. When invoking the concrete services, the execution engine also monitors the component services in case the availability and performance of these services violates the execution plan.

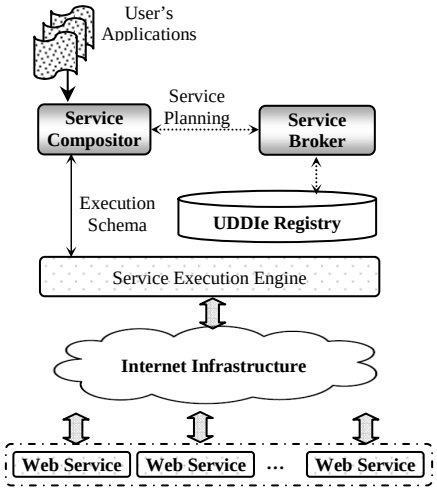


Fig. 1. Framework of Web Service Systems

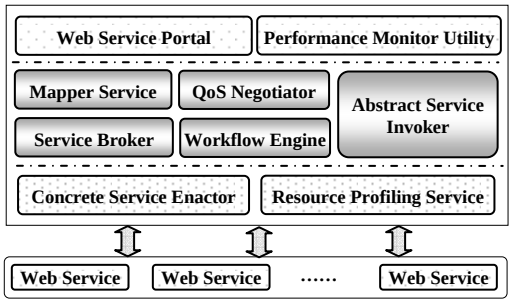


Fig. 2. Framework of Integrated Web Service Platform

In this work, we present a novel web service framework which is shown in Fig. 2. In our integrated web service framework, applications firstly submit their specifications through the portal component, which will translate the application-level descriptions into middleware-level requirements containing service's abstract interfaces and application's QoS constraints. There is another user-level component called *Performance Monitor Utility*, which is designed to monitor the application's runtime performance including responsive time, executing progress, web service status and etc.

The mapper component is designed for selecting the most suitable services for user's applications based on the QoS constraints as shown in the next sections. It is noteworthy that application's QoS constraints are different in various scenarios. In this paper, we mainly focus on the five most mentioned QoS measurements including availability, execution time, reliability, trustiness and costs. As to the broker component, it is designed for invoking services that selected by the

mapper component. Also, it is responsible for executing QoS negotiation by interacting with the negotiator component. Unlike the existing web service systems which generally evaluating the performance metrics by static information, we designed performance profiling subsystem in our framework, which periodically collects the runtime performance logs including both applications' execution and the services' dynamic attributes. Based on these runtime logs, our framework provide an adaptive mechanism to provide better QoS performance for user's applications. The component called abstract service invoker is designed for calling the abstract service corresponding to application's requirements through the mapper component. After the service schema is decided, the component called concrete service enactor is responsible for enabling the schema by interacting with the underlying services. The workflow engine is designed for managing the execution for complex applications which is designed in the form of precedent-constrained task graph.

IV. SEVICE SELECTION AND COMPOSITION ALGORITHM

As mentioned previously, when there are multiple services which provides identical or similar functionality. Therefore, it is the responsibility of users to discriminate these alternatives based on certain QoS constraints. Typically, the concept of QoS measurement involves a large set of non-functional attributes, such as responsive time, throughput, availability, execution time, reliability, security, trustiness, costs and so on. These QoS properties may apply to standalone web services, composition of multiple services, or the whole system. To formulate the QoS properties of individual services, we first design the QoS model which mainly concentrates on the requirements from the perspective of upper-level users. In this QoS model, multiple dimensions are taken into account for service composition in the next step.

Based on the above QoS model, we design the web service execution model, which is aiming to generate concrete service execution schema under QoS constraints of users. When building a complex service-based application, a large number of services will be involved for successfully completion. More importantly, the set size of the available candidate services is much larger and the finally selected set. On the other side, the finally obtained QoS performance depends on too many factors. For instance, the application may require minimizing the execution time as well as meet the budget and trustiness constraints, while other application may put more weight on the real-time responsiveness such as the interactive-intensive applications. To satisfying the various requirements of so many applications, QoS-aware approach to service selection and composition is of significant importance. While, how to evaluating so many QoS measurements is a challenging task, not mentioned their overlapping and interaction with each other. So, the first step for QoS-aware service selection and composition is to distinguish these measurements in terms of evaluation model. And then certain optimization techniques can be applied to maximize/minimize the total QoS utility of the service-based application by taking the constraints or preferences of users into account.

A. QoS Model of Individual Web Service

To differentiate the performance of available web services, we need to define a quantitative model that is applicable to most of the web services. Typically, there are multiple measurements can be associated when running a service-based application. In this work, we select five most mentioned QoS measurements including availability, execution time, reliability, trustiness and costs as the basic measurement to evaluate the system performance. To facilitate quantitative model, those QoS measurements are all defined as real numbers which can be changed in certain range. So, if two services are providing identical functionalities, we can easily differentiate them by these QoS quantitative models. In our framework, the static measurements are assumed to stored in the registry database and the dynamical measurements can be retrieved through the performance profiling component as shown in Fig. 2. Here, we firstly list the considered QoS measurements for individual services as following:

- *Availability*. The availability a service is the probability that the service is accessible. It is noteworthy that the value of availability may vary depending on a particular application. For example, if a service is frequently accessed, it should be assigned a small availability value from the perspective of the application; If the service is less frequently accessed, using a larger availability value is more appropriate. Therefore, we define the availability of a service as following:

$$Avail(S) = \sum T_i^{access}(s) \Big/ T_{exec}(s) \tag{1}$$

  where $T^{access}(s)$ is the accessing time during the running the service , $T_{exec}(s)$ is the total execution time the service, which is described in the next.
- *Execution Time*. Generally speaking, this metric should measure the expected delay in seconds between the moment when a request is sent and the moment when the results are received. However, it is common that specific operations that have various execution for most services. To take this into account, we use the average execution time of all operations that give service exposed, which is evaluated as following

$$T_{exec}(S) = \frac{1}{n}\sum_{i=1}^{n} T_{exec}^{i}(op_i, s) \tag{2}$$

- *Reliability*. This metric is to evaluate the successful execution rate of the given service. In common sense, the successful execution rate is related to hardware and software configuration of web services. In addition, the network connections between the service requesters and providers are also of significant importance with this metric. To accurately evaluate it, the service system should log all the execution results as well as the resources reliability. In this work, we simple use the historical logs of the execution results obtained from user's feedback because of its objective.

- *Trustiness*. This metric is to measure the trustworthiness of a given service in terms of the differences between the proposed QoS and the practical QoS performance. Therefore, it mainly depends on end user's experiences of using the service. Different end users may have different opinions on the same service. Therefore, we define the value of the trustiness as the average ranking given to the service by end users, which is shown as following

$$Trusty(S) = \frac{\sum Trusty_i(s)}{N(t_2 - t_1)} \tag{3}$$

where $N(t_2\text{-}t_1)$ is the number of users who accessed this service during time interval $[t_1, t_2]$

- *Costs*. It is the fee that the user has to pay for invoking the service. For many service provider, they often charge user on basis of invoking operations. So, cost of a service is the function of the invoking interface noted as $Cost(s, op_i)$.

B. Composition QoS Model of Applications

Typically, a web service will expose its functionalities by a set of interfaces (also called operations). After the procedure of service composition, the composed services is often described as a set of activities, which consist of of precedent-constrained execution graph. In this work, we note the execution grapgh as a directed graph $G=<\boldsymbol{T}, \boldsymbol{E}>$, where $\boldsymbol{T}=\{t_1, t_2, \ldots, t_n\}$ is the set of tasks, $\boldsymbol{E}=\{e_{i,j}|\text{if } <t_i,t_j>\in \boldsymbol{T}\times\boldsymbol{T}\}$. The first task is noted as t_{init} and the final task is noted as t_{exit}. After mapping operation, an abstract application execution graph will be translate into concrete execution schema. Here, we note a concreted execution graph as $G^*=<\boldsymbol{S}, \boldsymbol{P}>$, where $\boldsymbol{S}=\{ws_1, ws_2, \ldots, ws_n\}$ is the set of Web services, $\boldsymbol{P}=\{p_1, p_2, \ldots, p_n\}$ is the set of execution path. If a task t_i is mapped onto s_j and invokes the k-th operation interface, such a mapping relationship is noted by $< t_i, ws_{j,k}>$. Invoking path $ip_k=\{p_i,\ldots,p_j\}$ is a set of order execution path, which starts from p_i and ends at p_j. Based on the above QoS model of individual service and the execution model, we model the application QoS requirements (or constraints) by following equations:

$$Avail(ip_k) = \prod_{<t_i, ws_{j,l}>\in ip_k} Avail_{i,j} \tag{4}$$

$$T_{Exec}(ip_k) = \max\left\{ \sum_{<t_i, ws_{j,l}>\in ip_k} T_{exec}(s, ip_k) \right\} \tag{5}$$

$$Cost(ip_k) = \sum_{<t_i, ws_{j,l}>\in ip_k} Cost_{i,j} \tag{6}$$

$$Trusty(ip_k) = \frac{1}{|ip_k|} \cdot \sum_{<t_i, ws_{j,l}>\in ip_k} Trusty_{i,j} \tag{7}$$

$$Rel(ip_k) = \max\left\{ \prod_{<t_i, ws_{j,l}>\in ip_k} Rel(s) \right\} \tag{8}$$

According to the above QoS models as shown in (4)~(8), the QoS-aware selection and composition problem can be described as the following optimization programming problem.

$$\begin{aligned}
&\max \sum_{i=1}^{k} \left(Cost(ip_k) + Avail(ip_k) + Trusty(ip_k) + Exec(ip_k) + Rel(ip_k) \right) \\
&s.t. \quad \sum_{i\in T}\sum_{j\in S}\sum_{m\in ip_k} c_{i,j}^m \cdot Cost_{i,j} \le QoS_{cost} \\
&\max_{i\in T}\{\max_{j\in S}\{\max_{m\in ip_k}\{e_{i,j}^m \cdot Exec_{i,j}\}\}\} \le QoS_{exec} \\
&\frac{1}{|ip_k|}\cdot \sum_{i\in T}\sum_{j\in S}\sum_{m\in ip_k} r_{i,j}^m \cdot Trusty_{i,j} \le QoS_{trusty} \\
&\prod_{i\in T}\prod_{j\in S}\prod_{m\in ip_k} j_{i,j}^m \cdot Rel_{i,j} \le QoS_{rel} \\
&\sum_{m\in ip_k} c_{i,j}^m = \sum_{m\in ip_k} e_{i,j}^m = \sum_{m\in ip_k} r_{i,j}^m = \sum_{m\in ip_k} j_{i,j}^m = 1
\end{aligned} \tag{9}$$

where $c^m_{i,j}$, $r^m_{i,j}$ and $e^m_{i,j}$, are all 0-1 flag variants which indicate that whether $< t_i, ws_{j,m}>$ is in the invoking path ip_k; QoS_{cost}, QoS_{exec} and QoS_{rep} are the QoS constraints which is specified by applications and submitted through the portal services. Clearly, the problem described in (9) is a multiple-dimension constrainted 0-1 integrated programming problem, which can be solve by commercial solver tools.

It is noteworthy other QoS measurements can be easily incorporated in the problem (9), only if it can be defined as range-bounded real number like the previous QoS models. Adding too many measurements will significantly increase the complexity of the solving algorithms, which in turn reduces the application's execution performance. So, we merely choose the most mentioned measurements in our study.

C. QoS Negotiation Model and Algorithm

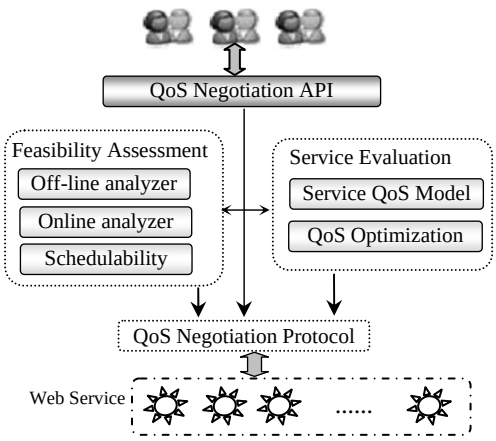


Fig. 3. QoS Negotiation Framework

The QoS negotiation process requires multiple interactions between users and the service broker as shown in Fig. 2 until they reach an agreement. The goal is to generate abstract mapping schema as optimal as possible with aiming to minimize the communication costs between service providers and applications and maximize the resource utilization as much as possible. The service broker will accept application's request by WS-Agreement protocol, and then generates a set of schemes in the WS-Agreement format. In this work, we

designed and implemented a QoS negotiation service to support the proposed QoS-negotiation model as shown in Fig. 3. This negotiation service is responsible for managing a distributed pool of web services to guarantee user's QoS requirements. It employs a WS-agreement extension protocol to keep track of service pool and report service's status. Schedulability analysis is used to provide service-mapping related guarantees. We assume that user's request arrival patterns are not known a priori. Therefore, user QoS requirements may be characterized off-line. Clients of the QoS negotiation service are user's service brokering requests.

Each request for guaranteeing a request includes its rejection penalty and the negotiation options of the client's requirements that specify different QoS levels and their respective rewards. A client task's QoS level is specified by the parameters of its execution model, which is described by XML files as shown in the following. With respect to negotiability, the QoS measurement such as costs and execution time are negotiable, while availability and reputation are not negotiable. An example of QoS constraints specified by a user is shown as following. The individual QoS measurement in the negotiation options describes the user's corresponding requirements when their applications are executed in the system. This allows users to define different versions of the task to be executed at different QoS levels or to compose tasks with mandatory and optional modules. User's QoS requests for guaranteeing tasks may arrive dynamically at any time including the execution period.

QosConstraint.xml

```
<QosContraints name=task1>
      <CostConstraint value=100 />
      <ExecTime value=2.5 />
      <Reliability value=0.8 />
      <Trusty value=0.9 />
</QosConstraints>
<QosContraints name=task2>
      <CostConstraint value=70 />
      <ExecTime value=1.5 />
       <Reliability value=0.6 />
      < Trusty value=0.75 />
</QosConstraints>
<QosContraints name=task3>
      <CostConstraint value=110 />
      <ExecTime value=5.0 />
      <Reliability value=0.6 />
      < Trusty value=0.95 />
</QosConstraints>
```

To guarantee a request, QoS optimization component is responsible for conducting QoS-optimization procedure by calculating the QoS levels for user's requirements and evaluate the potential benefits of guarantee them. The request may be rejected if the sum of benefits is lower than the pre-defined threshold, or the difference between the current and previous calculation is larger than the request's rejection penalty. In other cases, the user's QoS requirements can be considered as guaranteed for now. It is noteworthy that some typical admission controlling polices often defined high request rejection penalty if the system can not actually guarantee the user's requirements at runtime. In our framework, we provide an adaptive negotiation mechanism, which allows users decreasing their requirements not only before the execution, but also during the execution of their applications. This mechanism is implemented by incorporating a set of admission control rules into the service broker component, which will communicate with QoS negotiation subsystem at runtime. In this way, our proposed QoS framework achieves higher successful rate as well as decreased rejection penalties.

V. Experiments and Performance Evaluation

A. Experiments Parameters

In order to examine the effectiveness of our service framework and corresponding selection and composition algorithm, a series of experiments in the real-world platform are performed. In the experimental platform, we use the IBM WS-Toolkit as the basic deployment middleware. The benchmark application used in the experiments is deprived from the classical service-oriented numerical optimization project that originally developed by the University of Southampton [23]. In the testing platform, the underlying computing nodes are homogeneous in both hardware and software, each being configured with Pentium IV 2.8 MHz, 2 GB memory, Windows 2K as OS, J2EE as the execution environment, and Oracle XML Developer Kit. The network bandwidth between these computing nodes is 100 MB/s LAN. As to the QoS data, we set that all of them are retrieved by the service execution engine. As mentioned in Section III, the dynamical QoS measurements are logged by the performance profiling component, in which the individual measurements are calculated by the models described in the Section III.A. The static QoS measurements, such as availability, are collected when there are some services are boot up or shut down in the service provider's side.

B. Service Selection and Composition Algorithms

In the first set of experiments, we want to compare the performance of proposed MPWS (Mixed Programming based Web Service Selection) with two well-known heuristic algorithm: WFlow [25] and RWSCS_KP [26]. When using WFlow algorithm, service selection problem are categorized into two classes: service selection with sequential structure and the other is to composite services with a general flow structure. When the application belongs to the latter, its structure between function nodes might be of complex recursive calling chain (i.e., loop calling). To deal with such a problem, WFlow is designed to remove the loop operations by consequentially un-folding the loop. To generate the benchmarks, we randomly generate the abstract processes, each containing multiple control flows. Also, we set that each abstract service has the same number of candidate service operations. For each candidate service, five QoS parameters are noted as Q_{exec}, Q_{trusty}, Q_{rel}, Q_{cost} and Q_{rep}. Each quality value is randomly generated with a uniform distribution in a range.

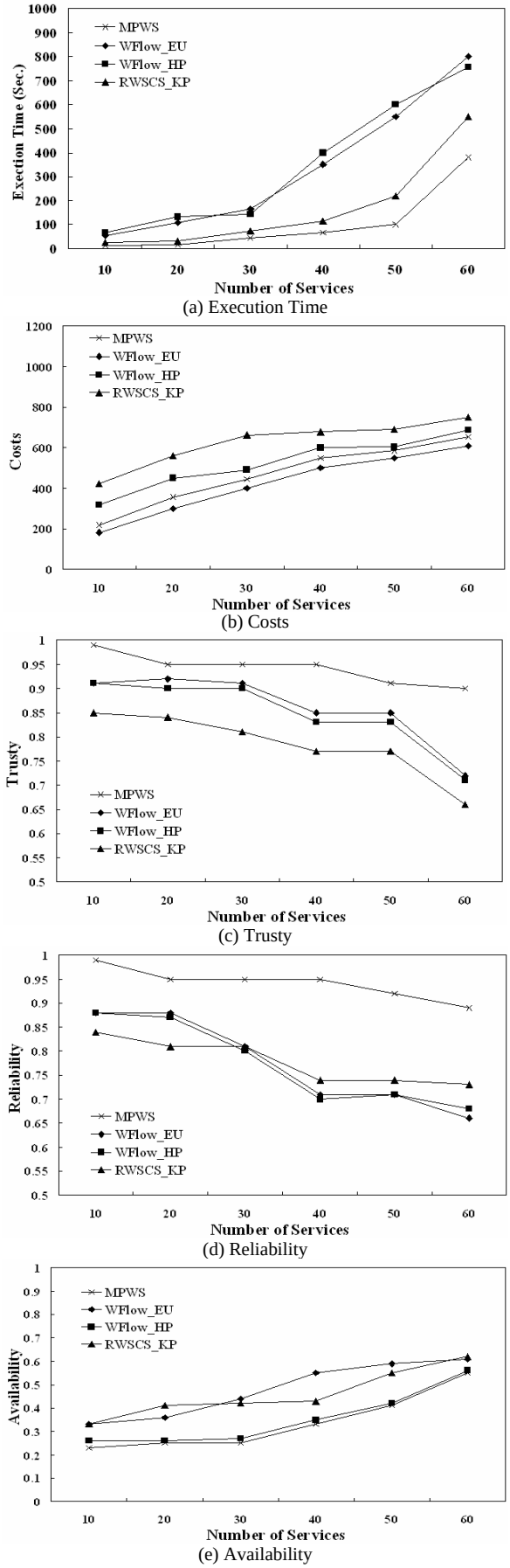


Fig. 4. Comparison of Five QoS Measurements

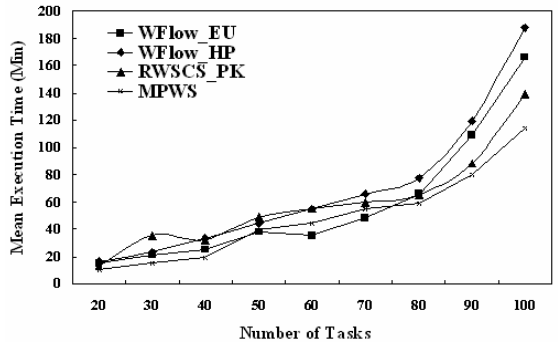
In this work, we mainly concentrate on the analysis of the impact of the number of services on the performance of QoS parameters. Since the number of services will affect the failure of finding a feasible solution. The more the abstract services are, the lower failure rate in finding a feasible solution is. Similarly, the number of candidate service also has effects on the other parameters in the same way. In the experiments, the numbers of services are set from 10 to 60 to analyze the influence over the four algorithms. As depicted in Fig. 4, PMWS has higher reliability and trusty when generating a feasible service composition schema than the other three algorithms for almost of the test cases. As to the execution time metric, we noticed that PMWS will select those services with better RPC (ratio of performance to costs) as the candidate. In this way, it obtained short execution time as well as low cost. Unlike PMWS, both two WFlow algorithms are aiming for improving the execution performance. So, when the number of service is small, they can obtain better performance in term of execution time; While the number of service is high, they tends to significantly increase the costs.

In the four algorithms, we found that RWSCS_KP seems performs worst in all cases except for the execution time. It is because that our test platform has twelve service container distributed in four locations. Between the different service containers, the network bandwidth is very unstable. For example, the average available bandwidth is only 23% of the maximal limitation when in 10:00 ~ 12:00 am. Unfortunately, the test applications require massive remote data transferring when executing on the system. Since the RWSCS_KP algorithm uses dataset location as its main heuristic for service selection, it performs better than WFlow algorithms in our experimental settings. At the same time, we found that its performance can not be maintained when the service number is increased to above 40. The reason may be the available candidate pool is not bigger enough in our testing platform.

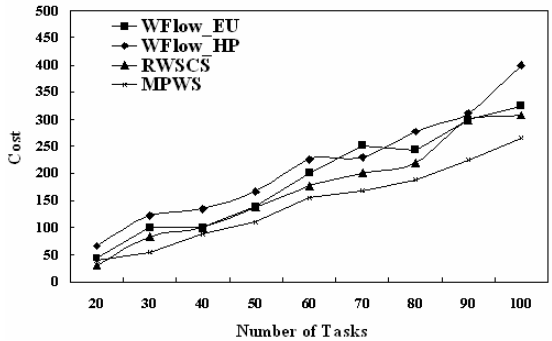
Peformance Comparison in Dynamical Environments

In the static environment, the QoS measurements of all services will not be alternated during the execution of the given composite application, and services are capable of executing the tasks successfully under the pre-required QoS constraints; In the dynamical environments, the QoS measurements of underlying services are assumed to be changeable during the execution of the application. It means that some of the selected services might be unavailable although it is no so at first; also, new services with better QoS measurement might be emerging; or the selected services might not be able to complete the execution of application in time. Therefore, in this experiment we mainly concentrate on the dynamical environment. Unlike static environment, the availability measurement is of no meanness any more. So, we only test four QoS measurements in this experiment. It is noteworthy, in static environment we can adjust the number of services manually; however, it can not be done

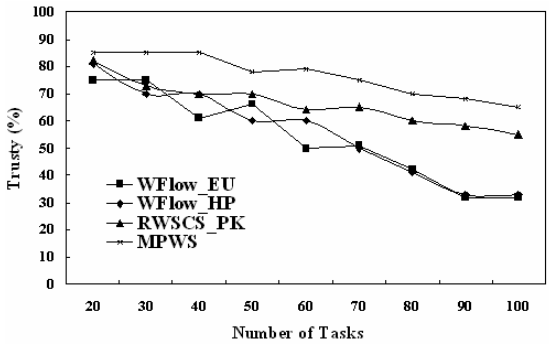
any more in dynamical environment. So, we change the size of subtask in the target application so as to simulate this situation. The experiment results are shown in Fig.5.



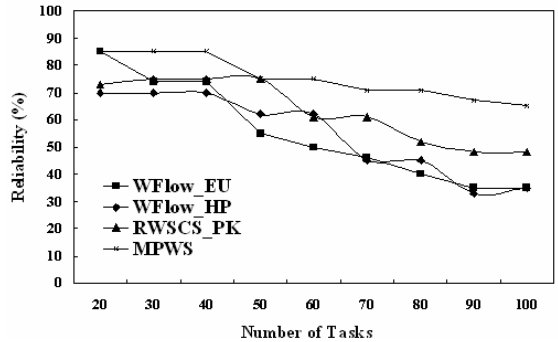
(a) Execution Time



(b) Costs



(c) Trusty



(e) Reliability

Fig. 5. QoS Performance in Dynamical Environment

As shown in Fig. 5, the performance of reputation and availability measurements are reduced about 12 percent to 21 percent when in dynamic environments. Since the availability of services is unpredictable in dynamical environments, the selected services according to optimal QoS-aware strategy might be un-available at the time when they are invoked. In our service framework shown in Fig. 2, re-negotiation mechanism is incorporated for dealing with such a problem. However, the re-negotiation operations can not be guaranteed to be successfully. So, the practical execution of the application might be sub-optimal as the candidate services with sub-optimal QoS measurements will be picked out for execution. At the same time, the re-negotiation operation requires extra costs, which will significantly increase the delay of underlying application. Therefore, the whole execution time the practical execution time is very uncertain. Fortunately, most of the re-negotiation operations can obtain the candidate services with better QoS measurements, in these cases, the extra benefits will compensate the execution latency caused by re-negotiation or re-optimization operations. These mechanisms make our service framework exhibiting better adaptive whether in static or dynamical environments. For example, combing Fig. 4 and Fig. 5, we can see that the QoS measurements of both cost and execution time are very similar whether in static environment or in dynamical environment.

Summarizing the previous experimental results, we draw the following three conclusions on the proposed service platform: (1) Generally, MPWS and WFlow_HP outperforms WFlow_EU and RWSCS_PK in terms of all QoS measurements; (2) when the system is facing some large-scale applications, MPWS is more adaptive than other existing algorithms to obtain global optimal/sub-optimal solution under user's QoS constraints; (3) In the dynamical environments, MPWS performs far more stably than other three strategies, which is the major improvement of the proposed framework.

V. CONCLUSION

In this work, we proposed an integrated framework which can optimize the performance of service-based application with constraints to user's QoS requirements. At the same time, the proposed framework also allows the system flexibly deploy the underlying services with constraints to resource provider's expectation. In our framework, the optimum mapping between abstract web services and application's processes is implemented through programming technique. Experimental result based on static and dynamical environments shown that the proposed framework and its QoS-based algorithm an significantly improve the user's QoS satisfaction in terms of five most-mentioned QoS parameters.

In the future work, we are planning to incorporate more emerging user's QoS parameters into our systems, i.e., security and accountability. Also, we are planning to migrate it for virtualization-based web-servers so as to be adaptable for cloud-based systems.

ACKNOWLEDG

This work was supported by Scientific Research Fund of Hunan Provincial Education Department (No. 09c270).

REFERENCES

[1] Y. Li, Z. Zhu. "A User-oriented and Context-aware Web services Composition". *Journal of Software*, Vol. 7, No. 4,

pp. 878-883, 2012.
[2] T. Andrews and F. Curbera, "Business Process Execution Language for Web Services (version 1.1)," http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf, 2003.
[3] A. Ankolekar, et al., "DAML-S: Web Service Description for the Semantic Web," *Proc. First Int'l Semantic Web Conf. (ISWC 02)*, 2002.
[4] B. Zhou, K. Yin, H. Jiang, S. Zhang, A.J. Kavs. "QoS-based Selection of Multi-granularity Web Services for the Composition". *Journal of Software*, Vol. 6, No. 3, pp. 366-373, 2011.
[5] G. Bartoli, et al. "An Optimized Resource Allocation Scheme Based on a Multidimensional Multiple-Choice Approach with Reduced Complexity", *Proc. of IEEE International Conference on Communications (ICC)*, pp. 1-6, 2011.
[6] W.-L. Lin, C.-C. Lo, K.-M. Chao et al., "Multi-group QoS consensus for web services," *Journal of Computer and System Sciences*, vol. 77, no. 2, pp. 223-243, 2011.
[7] S. Ceri, F. Daniel, M. Matera, and F. Facca, "Model-Driven Development of Context-Aware Web Applications," *ACM Trans. Internet Technology*, vol. 7, no. 2, 2007.
[8] Z. He, L. Wu, H. Li, H. Lai, Z. Hong. "Semantics-based Access Control Approach for Web Service". *Journal of Computers*, Vol. 6, No. 6, pp. 1152-1161, 2011.
[9] A.B. Hassine, S. Matsubara, and T. Ishida, "A Constraint-Based Approach to Horizontal Web Service Composition," *Proc. Fifth Int'l Semantic Web Conf. (ISWC)*, pp. 130-143, 2006.
[10] S. Hwang et al., "Dynamic Web Service Selection for Reliable Web Service Composition," *IEEE Trans. Services Computing*, vol. 1, no. 2, pp. 104-116, Apr.-June 2008.
[11] M.I. Islam and M.M. Akbar, "Heuristic algorithm of the multiple-choice multidimensional knapsack problem (MMKP) for cluster computing", *Proc. of International Conference on Computers and Information Technology (ICCIT'09)*, pp.157-161, 2009.
[12] K. Kritikos, and D. Plexousakis, "Requirements for QoS-Based Web Service Description and Discovery," *IEEE Transactions on Services Computing*, vol. 2, no. 4, pp. 320-337, 2009.
[13] H.-Y. Jeong, and Y. S. Lee, "CSP Based Web Service Composition Model with Buffer at the Business Logic Process Level," *Journal of Internet Technology*, vol. 13, no. 3, pp. 501-508, 012.
[14] Z. Maamar, S.K. Mostefaoui, and H. Yahyaoui, "Toward an Agent-Based and Context-Oriented Approach for Web Services Composition," *IEEE Trans. Knowledge and Data Eng*., vol. 17, no. 5, pp. 686-697, May 2005.
[15] H. Tong, J. Cao, S. Zhang et al., "A Distributed Algorithm for Web Service Composition Based on Service Agent Model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2008-2021, 2011.
[16] S. Oh, D. Lee, and S.R.T. Kumara, "Web Service Planner (WsPr): An Effective and Scalable Web Service Composition Algorithm," *J. Web Services Research*, vol. 4, no. 1, pp. 1-23, 2007.
[17] W. Niu, G. Li, H. Tang et al., "CARSA: A context-aware reasoning-based service agent model for AI planning of web service composition," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1757-1770, 2011.
[18] S. Patil and E. Newcomer, "ebXML and Web Services," *IEEE Internet Computing*, vol. 7, no. 3, pp. 74-82, 2003.
[19] J. Schaffner, H. Meyer, and C. Tosun, "A Semi-Automated Orchestration Tool for Service-Based Business Processes." *Proc. Second Int'l Workshop Eng. Service-Oriented Applications: Design and Composition (WESOA)*, pp. 50-61, 2006.
[20] D. Chiu, Y. Yueh, H.-f. Leung et al., "Towards ubiquitous tourist service coordination and process integration: A collaborative travel agent system architecture with semantic web services," *Information Systems Frontiers*, vol. 11, no. 3, pp. 241-256, 2009.
[21] D.C. Vanderster, N.J. Dimopoulos, R.J. Sobie, "Metascheduling Multiple Resource Types using the MMKP", *Proc. of IEEE/ACM International Conference on Grid Computing*, pp.231-237, 2006.
[22] S. Wang, W. Shen, and Q. Hao, "An Agent-Based Web Service Workflow Model for Inter-Enterprise Collaboration," *Expert Systems with Applications*, vol. 31, no. 4, pp. 787-799, 2006.
[23] G. Xue, W. Song, S.J. Cox, A. Keane. Numerical Optimisation as Grid Services for Engineering Design. *Journal of Grid Computing*, vol.2, no.3, pp.223-238, 2004.
[24] V. Chifu, and I. Salomie, "A Fluent Calculus Approach to Automatic Web Service Composition," Advances in Electrical and Computer Engineering, vol. 9, no. 3, pp. 75-83, 2009.
[25] T. Yu, Y. Zhang, K.J. Lin. "Efficient algorithms for web services selection with End-to-End QoS constraints", *ACM Transactions on the Web*, vol. 1, no. 1, 2007.
[26] H. Cao, X. Feng, Y. Sun, Z. Zhang, Q. Wu. "A service selection model with multiple QoS constraints on the MMKP", *Proc. of the IFIP International Conference on Network and Parallel Computing Workshops*, pp.584-589, 2007.

Changsong Liu was born in 1974. He received his master degree in Xian University of Technology in 2004. Now, he is a Ph.D candidate in Central South University, and currently works in Hunan Institute of Engineering as lecturer. His research interests include service computing, quality of service management, distributed intelligence.

Dongbo Liu was born in 1974. He received his master degree in Jiangsu University of in 2004. He obtains the Ph.D degree in Hunan University in 2010, and currently works in Hunan Institute of Engineering as associate professor. His research interests include Web service, distributed intelligence, cloud computing and etc.