International Journal of Future Generation Communication and Networking
Vol.7, No.1 (2014), pp.249-260
http://dx.doi.org/10.14257/ijfgcn.2014.7.1.25

ESCA: Evolution-strategy based Service Composition Algorithm for Multiple QoS Constrained Cloud Applications

Xiao Peng* and Liu Changsong

Department of Computer and Communication, Hunan Institute of Engineering
**xtanefn@gmail.com, liuchangsong74@gmail.com*

Abstract

Web services have been widely applied in e-business or organizations to establish versatile applications. However, service composition with constraint to QoS constraints still remains as a challenging issue, because both the service performance and the QoS requirements of applications tend to be dynamically changed at runtime. To address this problem, in this paper we present a novel service composition framework, which takes advantage of flexibility provided by cloud systems and applies evolution strategy to solve the optimal programming problem of multiple QoS constrained service composition. Furthermore, a QoS negation mechanism is proposed to satisfy the dynamical and elastic cloud environments. Massive experiments are conducted to investigate the effectiveness of the proposed framework, and the results show that it can significantly reduce the costs of large-scale service-based application in terms of various QoS metrics.

Keywords: *Web Service; Cloud Computing; Quality of Service; Optimal Programming; Evolution Strategy*

1. Introduction

As the globalizing market is requiring unprecedented interoperability to integrate diverse information systems to share knowledge and collaborate among organizations, web service has emerged as a promising paradigm, which can effectively associate loosely coupled software components for solving complex business/scientific problems [1]. Therefore, it has been identified as the basic technical building block for the next generation of web-based business solutions, which is featured with application, platform, and provider independence [2, 3].

The most important value of web service technologies is the composition of basic services to create value-enhanced performance [4, 5]. However, in the presence of so many services with similar functionality, users need to discriminate these alternatives based on their quality of service requirement. Therefore, services need to be described and understood both in terms of functional capabilities and quality of service properties [5, 6]. Recently, many global solutions are proposed to satisfy the process constraints and user preferences for the whole application. In this way, QoS constraints can predicate at a global level, for example, the constraints posing restrictions over the whole composed service execution can be introduced. However, those global approaches introduce an increased complexity with respect to local solutions, also, their performance tend to be variable since workload on Web services are fluctuated dramatically. So, a dynamic and adaptive composition approach is needed, in which runtime changes in the QoS of the component services are taken into account.

Recently, cloud computing has emerged as a promising technology that allows users deploy their applications in an environment with increased scalability, availability, and fault tolerance [7]. In a consequence, many service servers has been organized as high-performance clusters, which provide standard interfaces and services through cloud systems (also called SaaS infrastructure) [9]. In such environments, available services are dynamically provided in an elastic manner, and the users will be charged according their QoS requirements and the underlying resources. On the other side, service providers began to care about the resource utilization more than before. For example, average enterprise server utilization is around 10-20 %. However, this sub-optimal decision has been made because violating service level agreement (SLA) is more costly. By adapting cloud computing, enterprises can dynamically provision the resource at the current demand level, and no longer has to pay for idling resources.

Motivated by the above observations, in this paper we introduce a novel service composition algorithm which formulates the service composition problem as a multiple choice multiple dimension knapsack problem and applies evolution strategy to obtain an approximate solution. In this way, we can efficiently selective the optimal set of services from a large number of candidates according the user's QoS requirements. Also, it takes into account the benefits of cloud providers by using a flexible SLA mechanism when resource prices are negotiated before the application execution.

2. Related Work

With more and more attentions on web applications, user's QoS requirements such as costs, reliability, security and etc has been considered as important QoS measurement when running Internet-based applications [2, 10, 18]. The standard description of service composition is proposed in ebXML [18] and DAML-S [3]. DAML-S supports the Semantic Web services based on a generic ontology, in which both functional and QoS aspects of services are expressed as rule-based preconditions and post-conditions on service operations. However, neither DAML-S nor ebXML consider user's QoS criteria, nor do they address the issue of dynamic service selection and adaptive service composition. The technique of Web service composition has received a lot of attention. Most of the work can be broadly classified into three categories: manual composition, semi-automatic, and automatic composition. In manual composition, the composite service is modeled manually by using a service flow language such as BPEL4WS [2] and they often requires the knowledge of specific domain. Meanwhile, it is a labor-intensive and error-prone task; so, it is not appropriate for the large-scale web service composition. In [6, 12, 19], some Semi-automatic composition techniques have been proposed, which solve some of the problems of manual composition.

In [13], the authors designed a framework which interleaves planning and execution of complex applications whose functional objectives and QoS requirements are specified by assertions of XSAL languages. In [8], a similar approach is proposed, which built complex applications from a high level workflow specification by applying contingency planning technique. However, the drawback of these techniques is high computation complexity, which often means that only suboptimal solution can be obtained by some heuristic functions. The technique of agent-based service composition has recently received great attention. For example, McIlraith *et al.*, proposed an agent-based web service composition framework [15]; Maamar *et al.*, presented an agent-based and context-oriented approach that supports the composition of web services [14]; Wang *et al.*, proposed an agent-based web service workflow model for inter-enterprise collaboration [22]. During service composition process, software agents engage in conversations with their peers to agree on the web services that participate in this process. However, the agents in the above mentioned approaches are not used to achieve the automatic web service composition but to the coordination and enactment of the composite web services.

3. Cloud-based Service Composition Framework

3.1. System Architecture

In this work, we deployed the underlying resources on cloud computing systems. In this way, we can fully take advantages of features of cloud platforms, such as resource virtualization, server consolidation, load-balance, and virtual machine migration. Therefore, the architecture is illustrated in Figure 1.

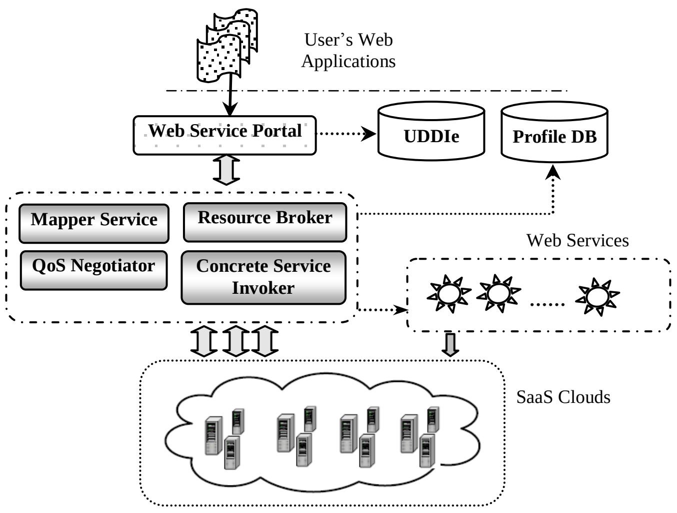


Figure 1. Architecture of QoS-aware Web Service Platform

In the framework, user applications invoke Web services through *Web Service Portal*, in which the service abstract interface requirements and user's QoS constraints are specified. The registry service is an extended version of a UDDI registry, which provides related information on service abstract interface and corresponding QoS characteristics. The *Concrete Service Invoker* is responsible for invoking the concrete service corresponding to application's requirements through a *Mapper* component. Then the *Mapper* selects the best concrete services for each task of the composed service according to the optimization criteria which will be discussed in the remainder of this paper. The *Broker* component is responsible for invoking services that selected by *Mapper*, and QoS negotiation is performed through the *Negotiator* component. In order to provide adaptive QoS for user applications, we designed an application profile database which includes the context of application invocation, application performance profile and service performance profile. As the invocation context can vary over time, by this profile database, different services can be invoked according to the modifications of current context.

For the convenience of representation, we give some notations of web service system in the following. A composed service is specified at an abstract level as a high-level business process, which contains an initial task t_{init} and an exit task t_{exit}. So, an abstract composed service can be noted as a directed graph G=<$\boldsymbol{T}$, $\boldsymbol{E}$>, where $\boldsymbol{T}=\{t_1, t_2, \ldots, t_n\}$ is the set of tasks, $\boldsymbol{E}=\{e_{i,j} | \text{if } <t_i,t_j> \in \boldsymbol{T}\times\boldsymbol{T}\}$. As mentioned above, the *Mapper Service* is responsible for transforming abstract business process into concreted composed service. So, a concreted composed service can also be noted as a directed graph 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. When a task t_i is mapped onto s_j and invokes the k-th operation interface, it is noted as a pair $< t_i, ws_{j,k}>$. 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.

3.2. QoS Measurements of Web Services

Several QoS measurements can be associated when executing a web service. In this paper, we assume that the values of QoS measurements are real numbers that vary in a bounded range with a minimum and a maximum value. If the same operation is accessible from the same Web service and the same provider, but with different quality characteristics, then multiple copies of the same operation will be stored in the registry, each copy being characterized by its quality profile.

We mainly take into account the following subset of QoS measurements, which have been the basis for many QoS-aware service architectures [10, 15, 20]: (1) $Avail_{i,j}$: The probability that the service operation $ws_{i,j}$ is accessible, which is a number in the range [0, 1]; (2) $Cost_{i,j}$: The fee that a user has to pay to the service provider for the service invocation $ws_{i,j}$; (3) $Exec_{i,j}$: The duration between the time when $ws_{i,j}$ is invoked and then time when the result is obtained; (4) $Rep_{i,j}$: It is defined as the ratio of the number of service invocations which comply the negotiated QoS to the total number of invocations, which is in the range [0,1]. With respect to negotiability, the QoS measurement such as costs and execution time are negotiable, while availability and reputation are not negotiable. Therefore, the aggregated QoS measurements of a composed service can be calculated as following.

$$Cost(ip_k) = \sum_{<t_i, ws_{j,l}> \in ip_k} Cost_{i,j} \tag{1}$$

$$Exec(ip_k) = \max\left\{ \sum_{<t_i, ws_{j,l}> \in ip_k} Exec_{i,j} \right\} \tag{2}$$

$$Avail(ip_k) = \prod_{<t_i, ws_{j,l}> \in ip_k} Avail_{i,j} \tag{3}$$

$$Rep(ip_k) = \frac{1}{|ip_k|} \cdot \sum_{<t_i, ws_{j,l}> \in ip_k} Rep_{i,j} \tag{4}$$

3.3. Evolution Strategy based Service Composition Algorithm

In this work, we focus on Web service composition with multiple QoS constraints. Based on the aggregated QoS definition in (1)-(4), the problem can be determined by solving the following optimization programming problem.

$$\max Q = \sum_{i=1}^{k} \left(Cost(ip_k) + Avail(ip_k) + Rep(ip_k) + Exec(ip_k) \right)$$

$$\begin{aligned} s.t. \quad & \sum_{i\in T} \sum_{j\in S,\, m\in ip_k} c^m_{i,j} \cdot Cost_{i,j} \le QoS_{cost} \\ & \max_{i\in T}\{\max_{j\in S}\{\max_{m\in ip_k}\{e^m_{i,j} \cdot Exec_{i,j}\}\}\} \le QoS_{exec} \\ & \frac{1}{|ip_k|} \times \sum_{i\in T} \sum_{j\in S,\, m\in ip_k} r^m_{i,j} \cdot Rep_{i,j} \ge QoS_{rep} \\ & \sum_{m\in ip_k} c^m_{i,j} = \sum_{m\in ip_k} e^m_{i,j} = \sum_{m\in ip_k} r^m_{i,j} = 1 \end{aligned} \tag{5}$$

where $c^m_{i,j}$, $r^m_{i,j}$ and $e^m_{i,j}$, are all boolean 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 user's QoS constraints that specified in the QosConstraint.xml file. It is clear that the above problem is a classic 0-1 integrated programming problem, which is equivalent to the *Multiple choice Multiple dimension Knapsack Problem* (MMKP). MMKP is a well-known NP-hard problem and many heuristic approaches have been proposed to solve it approximately [5, 11, 21]. In this paper, we use evolution-computing technique to solve the above problem, and the algorithm is shown as following.

Algorithm1: Optimal Service Composition with Multiple QoS Constraints

```
Begin
1.  $Q$ := 0;
2.  Generate a random invoking path $ip$;
3.  while evolution iteration $W$ is not reached do
4.   for each $t_i$ in $T$ do
5.     $WS_i$ := {}
6.     for each $ws_j$ in $S$ do
7.       if $ws_j$ is not candidate service of $t_i$ then continue
8.       else $WS_i$ := $WS_i$ + {$ws_j$}
9.     end for
10.      for each $< t_i, ws_{j,m}>$ in $ip$ do
11.        Randomly generate triple $< c^m_{i,j}, r^m_{i,j}, e^m_{i,j},>$
12.        if $< c^m_{i,j},r^m_{i,j},e^m_{i,j},>$ satisfying the QoS constraints then
13.          Calculate $Cost(ip_m)$, $Exec(ip_m)$ and $Rep(ip_m)$
14.          $Q' := Q' + Cost(ip_m) + Exec(ip_m) + Rep(ip_m)$
15.          if $Q' > Q$ then $Q := Q'$
16.          else
17.            Mutate $ip$ using swap-mutation with the probability of 0.5;
18.            go to step 3
19.          end if
20.        end if
21.     end for
22.   end while
end.
```

3.4. QoS Negotiation in Cloud Environment

In elastic cloud environments, QoS negotiation might happen before the service composition or at the application's runtime. In addition, the negotiation process will involve multiple participants and multiple QoS attributes. Thus, the whole negotiation process can be implemented as a set of parallel bilateral bargaining sessions between the negotiation Broker and a set of service providers. In this paper, we assume that service broker *b* and service provider *p* act as autonomous agents. For each QoS attributes, the value of negotiation bound is limited in a range [$q^{max}_{i,j,k}$, $q^{max}_{i,j,k}$], and let $q^{x}_{i,j,k} \in [q^{max}_{i,j,k}, q^{max}_{i,j,k}]$ represent the x^{th} QoS attribute value when task t_i is mapped onto s_j and invokes the *k*-th operation interface. Therefore, each agent utility function can be defined as U_n^x: [$q^{max}_{i,j,k}$, $q^{max}_{i,j,k}$] $\rightarrow \{0, R^+\}$ that indicates the benefit evaluation when the agent *x* is assigned to a quality attribute $q_{i,j,k}$. For convenience of nomination, we define U_n^x as following

$$U_n^x = \frac{q^{max}_{i,j,k} - q^{x}_{i,j,k}}{q^{max}_{i,j,k} - q^{min}_{i,j,k}} \tag{6}$$

The negotiation process between two agents includes a set of succession of offers and counteroffers until an offer is accepted by the other side or one of the agents terminates. So, an agent *x* will accept an offer if the value U_n^x is greater than the value of the next iteration that the agent is ready to send in the next iteration. In order to provide a counter offer, an agent uses a set of tactics to generate new values for each negotiated quality attribute. In this paper, we have implemented time dependent tactics and the offer for the quality dimension $q_{i,j,k}(r)$ at the iteration r is evaluated as

$$q^{x}_{i,j,k}(r) = w^x_i(r) ?(q^{max}_{i,j,k} \quad q^{min}_{i,j,k}) + q^{min}_{i,j,k} \tag{7}$$

where function $\omega_i^x(r)$ is to indicate the urgency of agent *x* to achieve successful QoS negation at the *r* iteration.

In this paper, we adopt polynomial time functions since they concede faster in the beginning of the negotiation and this increases the probability of success of the negotiation process. So, the function $\omega_i^x(r)$ is defined as

$$w^x_i(r) = \left(\frac{\min(q^{x}_{i,j,k})}{q^{k}_{i,j,k}} \right)^{1/C} \tag{8}$$

where *C* is a constant value in [0, *R*].

In our service composition framework, the negotiation is automatic and is executed by the Broker according to service negotiation parameters that are stored in the UDDIe registry when concrete services are deployed. So, the negotiation process complexity is proportional to the number of negotiated quality attributes and to the agent deadlines. The QoS negotiation algorithm is listed as following.

Algorithm 2: QoS Negotiation for Service Composition or Re-optimization

```
Begin
1. $r$ := 0;
2. while $r < r_{max}$ and not find feasible solution of problem (5) do
3.   find a possible invoke path $ip_x$ which satisfies the maximum number of QoS
     constraints;
4.   for each service j do
5.     start a QoS negotiation process with each service provide $p_j$, and calculate each
       attributes according (6)
```

```
6.       if negotiation is successful with one provider then
7.          break out while loop;
8.       else  r:= r+1;
9.       end if
10.    end for
11.  end while
12.  if a feasible solution is obtained then
13.     return TRUE
14.  else
15.    return Failue
end.
```

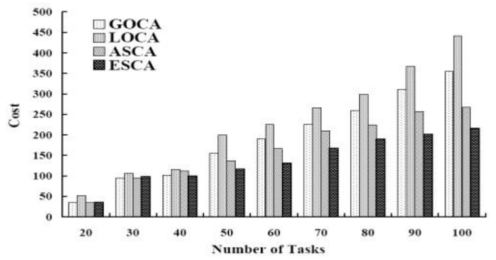
4. Experiments Evaluation and Analysis

4.1. Experimental Settings

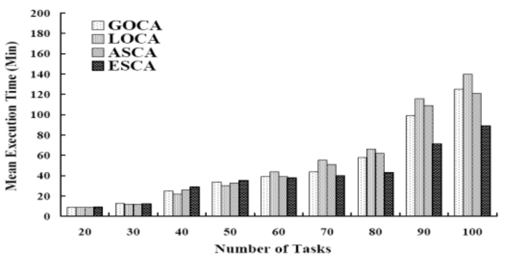
In our experiments, Web services were developed using IBM's Web Services Toolkit and deployed on a cluster of PCs. All PCs had the same configuration: Pentium IV 2.8 MHz with 2G RAM, Windows 2000, Java 2 Enterprise Edition, and Oracle XML Developer Kit. Host nodes are connected through 100Mbits/sec LAN. The target Web application in our experiments is an extended version of service-oriented numerical optimization project, which is deprived from the prototype that designed by University of Southampton [23]. In our experiments, QoS data is retrieved by the service execution engine in different ways depending on the QoS dimension. We investigate the performance in two situations: (1) In a static situation, the QoS of any Web service will not be changed during a given composite service execution, and services are able to execute the tasks successfully and in conformance with their expected QoS; (2) In a dynamic environment, the QoS of services may undergo changes during the execution of a composite service, which means that existing component services may become unavailable, new component services with better QoS may become available.

4.2. Performance Comparison

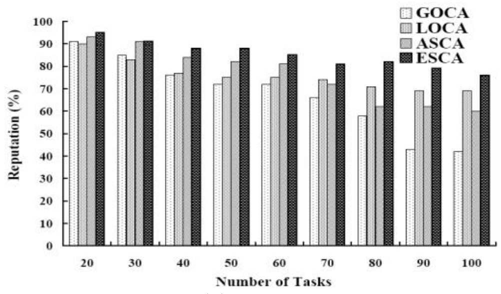
The first series of experiments aimed at evaluating the QoS of composite service executions in both static and dynamic environments. The performance of the proposed algorithm (Evolution-strategy based Service Composition Algorithm, ESCA) are compared with three classic composition policy, which includes Global Composition Optimization Algorithm [8] (GCOA), Local Composition Optimization Algorithm [16] (LCOA), and Agent-based Service Composition Algorithm [20] (ASCA). Four QoS measurements are all considered separately in our experiments, and experimental results are shown in Figure 2 and Figure 3.



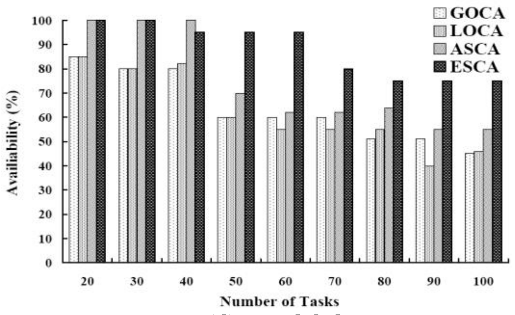
(a) Cost



(b) Execution time



(c) Reputation



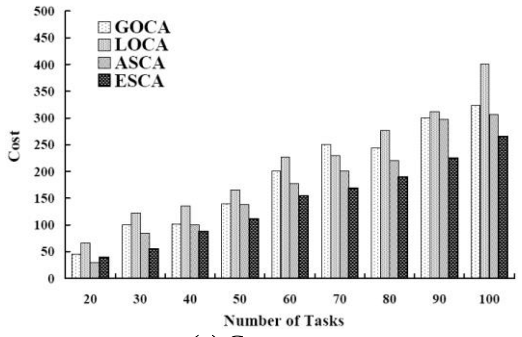
(d) Availability

Figure 2. QoS Performance Comparison in Static Environment

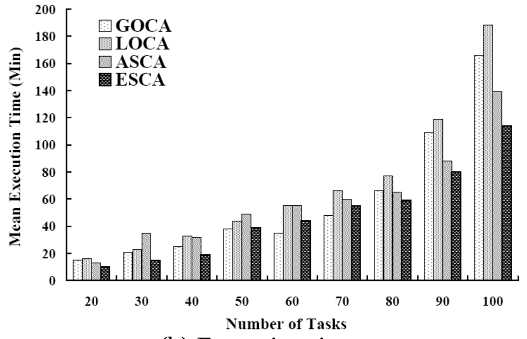
When the service composition occurs in static, the QoS of any Web service will not be changed during a given composite service execution, and services are able to execute the tasks successfully and in conformance with their expected QoS. So, QoS negotiations only are performed before the concrete service broker starts, which mean that re-negotiation never happens. In this case, we assume that reputation of a service is measured by its historic log that indicates the degree of its QoS compliance. As shown in Figure 2, the performance of GCOA is significantly better the LCOA policy. For example, the execution duration is consistently shorter when using GCOA than that of LCOA. It is because that the computation costs of GCOA is much higher that that of LCOA, which will be evaluated in the second experiment in details.

As to ASCA and ESCA, their performances are very similar when the number of tasks is less than 40. However, their variants become significant when we increase the task number of the experimental application. For example, the execution time of ESCA is about 25% shorter than that of ASCA when number of tasks is 100. As the analysis in [20], ASCA policy is based on the automotive characters of multiple agents, which is defined by administers of the system. When the size of an application is small, the ASCA can find an optimal solution by linear programming, which is similar to the proposed ESCA. However, when the number of tasks increases significantly, the solution of ASCA depends on the interaction of a large set of autonomous agents and the optimal solution can not be figured out any more.

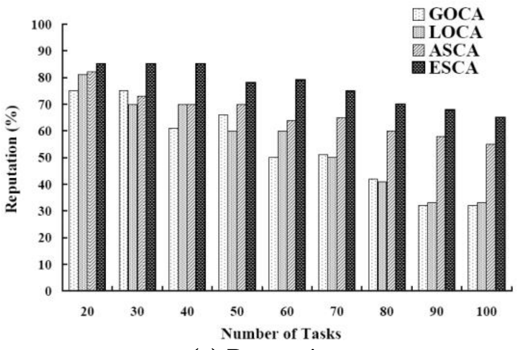
When in dynamic environment, the QoS of services may undergo changes during the execution of a composite service, which means that existing component services may become unavailable, new component services with better QoS may become available. As shown in Figure 3, the most differences between the two cases are that performance on reputation and availability measurements. The performance of both of QoS measurements are reduced about 12%~20% when in dynamic environments. Since the availability of Web services is dynamic, some services that are selected by the optimal execution plan may become unavailable when the task needs to be executed. Although the system can re-negotiate the unexecuted part of composite services, the executed part may become suboptimal, making the entire composite service execution suboptimal. There, execution time of a solution becomes uncertain, because re-negotiation will increased the delay of execution, but if better candidate service is available the benefits might compensate the delay brought by re-negotiation or re-optimization. So, in our experiments, we find that the performances of cost and execution time are almost the same whether in static environment or in dynamic environment.



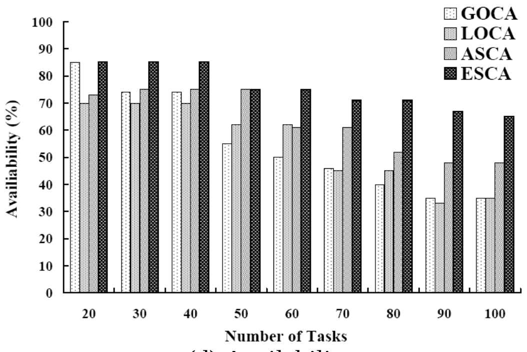
(a) Cost



(b) Execution time



(c) Reputation

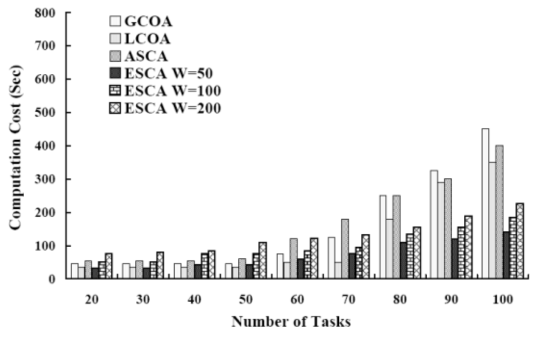


(d) Availability

Figure 3. QoS Performance Comparison in Elastic Environment

4.3. Efficiency Comparison on Composition Algorithms

In this experiment, we will investigate the computation cost of the proposed service composition algorithm. The aim is to provide a basis for determining the overhead for obtaining an optimal solution. For each test case, we executed the composite service 10 times and computed the average computation cost. Same as the first experiment, we choose the GCOA, LCOA and ASCA for comparison. As the complexity of ESCA is related to the parameter *W* (Evolution Iteration Number), so, we conducted the ESCA algorithm three with *W*=50, *W*=100, *W*=200. It is clear that bigger *W* will result in more computation cost but also bring about more optimal solution for ESCA. The experimental results are shown in Figure 4 and Figure 5.



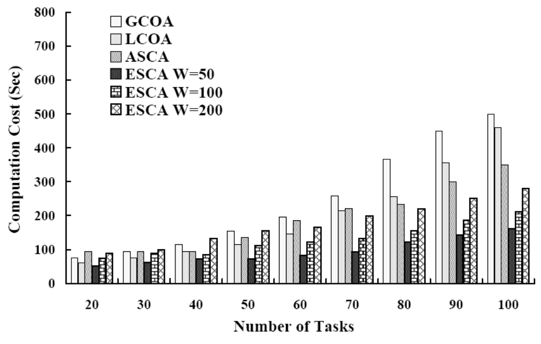


Figure 4. Computation Cost in Static Context

Figure 5. Computation Cost in Elastic Context

The mean computation cost of GCOA is the highest, and LCOA is the second highest. It is especially true when the number of task become very large. In addition, we noticed that computation costs of GCOA and LCOA increases faster than that of ESCA and ASCA. It is because that implementation of GCOA and LCOA requires travailing the abstract service graph and candidate service set several time before a concrete service composition is obtained. The computation cost of global planning by exhaustive searching is very high even in very small scale in aspect of the number of tasks and candidate service. So the computation costs of both of the two composition policies are multi-linear related to the size of the target graph of an application. In addition, when in dynamic environment, re-negotiation often results in the change of candidate service set, which in turn will increase the computation cost of the two policies. The computation cost of ASCA policy is better than GCOA and LCOA, and the reason have been mentioned in the first experiments.

As to ESCA policy, the experimental results indicate that its computation cost does not increasing linearly with the increasing of the number of tasks. By analyzing the detailed experimental data, we noticed that most of optimal solution is obtain before the evolution iteration is increased to the maximum account. That is why the computation cost of ESCA is not linear to the *W* value, for instance, when *W* value is increased 100% then computation cost of ESCA only increased about 30%~45%. It is noteworthy that communication delay of QoS negotiation is not taken into account when calculating the computation cost of these policies, since this overhead is heavily depended on the network traffic and topology of the tested environment. More detailed study on negotiation delay when compositing large-scale application will be our next work.

5. Conclusion

To address the issue of Web service composition with multiple QoS constraints, we introduce a service composition algorithm which formulates the service composition problem as a multiple choice multiple dimension knapsack problem and applies evolution strategy to obtain an approximate solution. Extensive experiments are conducted on a set of practical applications. Experimental results compare our method with other solutions and demonstrate the effectiveness of our approach toward the identification of an optimal solution to the QoS constrained Web service selection problem. In presence of large-scale application, the proposed ESCA algorithm is more suitable than ASCA to find global optimal solution with multiple QoS constraints. Also, the stability of ESCA is better than the other three policies when in dynamic environment. At present, the implementation of ESCA only considers four QoS measurements. Our future work will focus on more QoS requirement of business applications. Meanwhile, we will take efforts to improve the performance of QoS negotiation in our service composition framework.

References

[1] G. Alonso, F. Casati, H. Kuno and V. Machiraju, “Web Services”, Springer Verlag, **(2003)**.
[2] T. Andrews and F. Curbera, “Business Process Execution Language for Web Services”, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf, **(2003)**.
[3] A. Ankolekar, “DAML-S: Web Service Description for the Semantic Web”, Proc. First Int’l Semantic Web Conference, **(2002)**.
[4] D. Ardagna and B. Pernici, “Global and Local QoS Guarantee in Web Service Selection”, Proc. Business Process Managment Workshop, **(2005)**, pp. 32-46. Int’l J. Business Performance Managment, vol. 1, no. 4, **(2005)**, pp. 233-243.

[5] G. Bartoli, "An Optimized Resource Allocation Scheme Based on a Multidimensional Multiple-Choice Approach with Reduced Complexity", Proc. of IEEE International Conference on Communications, **(2011)**, pp. 1-6.
[6] S. Koulouzis, R. Cushing and K. A. Karasavvas, "Enabling Web Services to Consume and Produce Large Datasets", Ieee Internet Computing, vol. 16, no. 1, **(2012)**, pp. 52-60.
[7] C. Y. Lai, D. H. Shih and H. S. Chiang, "A study of cloud computing affecting switch intentions of information service in junior and elementary schools", International Review on Computers and Software, vol. 6, no. 3, **(2011)**, pp. 371-383.
[8] D. Zhang, P. Coddington, and A. Wendelborn., "Web services workflow with result data forwarding as resources", Future Generation Computer Systems, vol. 27, no. 6, **(2011)**, pp. 694-702.
[9] R. K. Jena and P. K. Mahanti. "Computing in the cloud: Concept and trends", International Review on Computers and Software, vol. 6, no. 1, **(2011)**, pp. 1-10.
[10] S. Hwang, "Dynamic Web Service Selection for Reliable Web Service Composition", IEEE Trans. Services Computing, vol. 1, no. 2, **(2008)**, pp. 104-116.
[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, **(2009)**, pp. 157-161.
[12] L. Ai, M. Tang and C. Fidge, "Partitioning composite web services for decentralized execution using a genetic algorithm", Future Generation Computer Systems, vol. 27, no. 2, **(2011)**, pp. 157-172.
[13] A. Lazovik, M. Aiello and M. Papazoglou, "Associating Assertions with Business Proesses and Monitoring Their Execution", Proc. Int'l Conf. Service Oriented Computing, **(2004)**, pp. 94-104.
[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, **(2005)** May, pp. 686-697.
[15] S. McIlraith and T. C. Son, "Adapting Golog for Composition of Semantic Web Services", Proc. Int'l Conf. Principles of Knowledge Representation and Reasoning, **(2002)**, pp. 482-496.
[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, **(2007)**, pp. 1-23.
[17] A. Patil, "METEOR-S Web Service Annotation Framework", Proc. 13th Int'l Conf. World Wide Web, pp. 553-562, **(2004)**.
[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, **(2006)**, pp. 50-61.
[20] H Tong, et al. "A Distributed Algorithm for Web Service Composition Based on Service Agent Model", IEEE Trans. on Parallel and Distributed Systems, vol. 2, no. 12, **(2011)**,pp. 2008-2021.
[21] D. C. Vanderster, N. J. Dimopoulos and R. J. Sobie, "Metascheduling Multiple Resource Types using the MMKP", Proc. of IEEE/ACM International Conference on Grid Computing, **(2006)**, pp. 231-237.
[22] Y. Ni and Y. Fan, "Model transformation and formal verification for Semantic Web Services composition", Advances in Engineering Software, vol. 41, no. 6, **(2010)**, pp. 879-885.
[23] G. Xue, W. Song, S. J. Cox and A. Keane, "Numerical Optimisation as Grid Services for Engineering Design", Journal of Grid Computing, vol. 2, no. 3, **(2004)**, pp. 223-238.
[24] M.-W. Zhang, B. Zhang and Y. Liu, "Web Service Composition Based on QoS Rules", Journal of Computer Science and Technology, vol. 25, no. 6, **(2010)**, pp. 1143-1156.

Authors

Xiao Peng received the Ph.D degree in computer science in CSU at 2010. He is currently an associate professor in the Hunan Institute of Engineering. His research interests include grid computing, distributed resource management. He is a member of ACM and IEEE.

Liu Changsong received his master degree in Xi'An University of Technology in 2004. Now, he currently works in Hunan Institute of Engineering as lecturer. His research interests include service computing, quality of service management, distributed intelligence. He is now a student member of IEEE and ACM. Also, he is a member of CCF in China.