Towards Dynamic Reconfiguration for QoS Consistent Services Based Applications

Yuyu Yin[1] and Ying Li[2]

[1] College of Computer, Hangzhou Dianzi University,
Hangzhou China
Yyy718@gmail.com
[2] College of Computer Science and Technology, Zhejiang University,
Hangzhou China
cnliying@zju.edu.cn

Abstract. Original overall QoS(quality of service) constraints of services based applications may be violated due to failed services or QoS degradation of component services. Although some methods have been proposed to repair failed services and achieve original overall QoS, few works focuses on recovery of services based applications from QoS degradation of component services. In order to provide QoS consistent service based applications, the paper presents a QoS driven dynamic reconfiguration method. The key to the method lies in replacing only some component services rather than recomposing the entire service based applications. In addition, degradation factor is introduced to find the component services which are replaced to most likely achieve original overall QoS constraint. In this way, reconfiguration overheads are lowered and service disruptions may be reduced. The test shows the effectiveness of our approach.

Keywords: Services based application, Quality of service, dynamic reconfiguration.

1 Introduction

Service Computing has become one of the most promising computing paradigms in the Internet era [1]. As it is now adopted widely, great progress has been made in the research about service composition. Presently, service composition has become an increasingly important way for IT enterprises to rapidly develop their applications[3].

However, developing various service based applications is not the only critical step of service composition. To the best of our knowledge, it becomes an additionally urgent challenge how to adjust service based applications to meet highly dynamic environments (e.g. in a clouding environment) and fast changing business requirements[4,5,9,10]. Up to now, although there existed many valuable works, most of them focus on providing essential services and functions in the presence of runtime environment changes[6-9]. With the ever increasing amount of services based applications now are adopted in wide range of critical domains(such as, real time system, navigation system, and online payment system), it has become increasingly important

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 771–778, 2012.

for enterprises to make service based applications deliver a desirable QoS. Due to many inevitable factors, such as network fault, host exception, and replacement of failed component services, delivered QoS from service based applications may not comply with their original claims at runtime. Once it happens, services based applications should be recovered immediately to continue holding original QoS. Moreover, most enterprises would like to recover their applications with lower cost and better efficiency, so that their customers may not undergo as few unexpected business shutdowns as possible. Thus, providing QoS consistent services based applications has become a huge challenge.

Although services based applications can be recovered by recomposition, service recomposition is extremely time-consuming and may lead to system shutdown, since the optimal service selection is a NP-hard problem[2]. Recently, some researchers have introduced and extended the traditional dynamic reconfiguration technology[8] to services based applications[6-8,12-14]. However, most of them still focus on the function driven dynamic reconfiguration, which is to maintain the pre-defined functions[6-8]. Few researchers[12,13] proposed dynamic reconfiguration methods to maintain the original end-to-end QoS constraints, they only limit to deal with the case of violation of some component service.

In the rest of the paper, the term component QoS and overall QoS are used to refer to QoS of a component service and QoS of an application respectively.

In this paper, to address this issue:

We propose a QoS driven dynamic reconfiguration method. When degradation of component QoS leads to violation of original overall QoS, we always try to replaced component services which have the biggest degradation factor, as long as they are reconfigured to deliver the original QoS. When some component services violate, our method replace them with new services firstly, and then repeat the above process for the rest of component services. In this way, our method can recover overall QoS with less attempts and shorter response time;

Inspired by our previous work[15], the notion of degradation factor is presented to guide us to find the component services which are replaced to most likely achieve the original overall QoS constraints;

The tests are conducted to evaluate the performance of the proposed methods.

The remainder of this paper is organizes as follows. Section 2 presents our method and gives the reconfiguration algorithms. Section 3 gives the tests to show the performance of our method. Section 4 surveys the related works. Finally, we draw a conclusion and discuss the future work in Section 5.

2 Dynamic Reconfiguration Method for Consistent QoS

We now present a QoS driven dynamic reconfiguration method. Different from previous works, our method does not limit to repair overall QoS in the case of failure of component services, it also deals with degradation of component QoS.

2.1 Dynamic Reconfiguration Algorithm

In this section, Algorithm RecQoS is to recovery the original QoS constraints. If there not exist failure component services, The algorithm first computes degradation factor of all component services in a services based application, and sorts all component services according to their degradation factor(Step 4,5). Degradation factor is introduced in Section 2.2. Then, two processes are executed as follows:

1) Single Services Replacement: we try to replace individual component service one by one in the descending order of degradation factor only if there exists some candidate service whose QoS is not worse than pre-defined QoS of the replaced service(Step 6-13). Obviously, original overall QoS constraints can also be satisfied by such replacement.

If such replacement cannot be found in the phase, then 2) Multiple Services Replacement: we begin to try to replace d component services whose degradation factor is the highest among all component services. QoS of the substitutions of the d component services is the best among their respective all candidate services. The range bound of d starts from two and increases gradually until a replacement is found to deliver original overall QoS(Step 14-24).

If there exists failure component services, Algorithm RecQoS first replace each failed service with the substitution whose QoS is the best among all candidate services of failed service(Step 27-30). If delivered overall QoS by the new services based application still do not satisfy original overall QoS, the algorithm RecDeg will be called to reconfigure the rest of component services.

Algorithm. RecQoS

Input: all replaceable component services $\{S_1,\ldots,S_n\}$ in a services based application S.

Output: the replaced services $R_S \subseteq \{S_1,\ldots,S_n\}$ and their substitutions $R_D \subseteq \{CS_i\}$.

Require: candidate services $CS_i=\{CS_{i1},\ldots,CS_{im}\}$ of service S_i.

```
1:  SET R_S = φ, R_D = φ, S_r = φ, D_g[i]=null;
2:  IF(not existing failure service sf_i)
3:      FOR(INT i = 0; i < n; i + +){
4:          D_g[i]=CalDg(S_i);}/*CalDg(S_i) is to calculate degradation factor of S_i */(See Section 2.2)
5:          S' = Sort(S); /*Sort(S) sorts S_1,..., S_n in ascending order according to D_g[]*/
6:          SET j = n;
7:          While(j > 0) {
8:              S_k = Get(S'. j); /* Gets_r(j) is to get the j-th element in S'*/
9:              IF(∃CS_kp ∈ CS_k && QoS of CS_kp is not worse than pre-defined QoS of S_k){
10:                 SET j = 0;
11:                 R_S = R_S ∪ {S_k} ; R_d = R_d ∪ {CS_kp}; Goto 27;
12:                 }Else{ j = j – 1;
13:             }
14:         }
15:         SET d=2;
16:         Do{
17:         FOR(INT l = 0; l < d; l ++){
18:             R_s[l] = R_s[l]  ∪ { Get(S'. j-l) };
19:             R_D[l] =R_D[l] ∪ { Select(S'.j-l) }; /*Select a service CS_kp for M[l] whose QoS is the best
    among all its candidates*/
20:         Replace M [l] with CS_kp
21:         }
```

```
22:          IF(the current QoS of S comply with the original overall QoS of S){ Goto 27;
23:              }Else{ d= d+1;
24:          }
25:      }While(d < n + 1);
26: }ELSE{
27:      RD[l] = Select(Sfl);/*Select a service CSkp for Sfl whose QoS is the best among all its candi-
     dates*/
28:      Replace Sfl with CSkp;
29:      RD[l] = RD[l] ∪ {CSkp};
30:      Rs = Sf;
31:      IF(the current QoS of S comply with the original overall QoS of S){ Goto 35;
32:          }Else{ Goto 3;
33:      }
34: }
35: RETURN RS[],RD[];
```

Once it happens, no such reconfigurations can deliver original overall QoS in current given candidate services repository. A recomposition should be needed to achieve original overall QoS for services based applications. But this goes beyond our current study.

2.2 Degradation Factor

For our method, we would like to find and replace the most promising component services so that the original overall QoS can be delivered as few attempts and as soon as possible. Thus, all component services in a services based application need to be evaluated by *Degradation Factor*.

Degradation factor of a component service shows the degree of its QoS actual degradation relative to other component service. When original overall QoS constraints are violated, the bigger relative degradation value of a component service is, the bigger its contributions to the violation are.

In this paper, we calculate degradation factor of a component service by the following steps:

a) to compute actual QoS degradation rate of a component service by Equation(1). Given a services based application Ω and its all component services $S_1,\ldots,S_n$.

$$\overline{\Delta_K^{S_i}} = \frac{\sum_{j \le m} \Delta_{K,j}^{S_i}}{m} \times \frac{m}{n}, \tag{1}$$

Where $\overline{\Delta_K^{S_i}}$ is the actual degradation rate of QoS property $K \in \{T_{S_i}, C_{S_i}, R_{S_i}, A_{S_i}\}$ from S_i; n is the monitored time in a period of time(users defined) before violation of original overall QoS constrains; $m(m \le n)$ is the degradation time; $\Delta_{K,j}^{S_i}$ is the actual degradation value of QoS property $K \in \{T_{S_i}, C_{S_i}, R_{S_i}, A_{S_i}\}$ from S_i.

b) to sort $S_1,\ldots,S_n$ according to QoS actual degradation rate. Four sorts are gotten as follows: $\Delta_T[], \Delta_C[], \Delta_R[]$,and $\Delta_A[]$. They are the descending sorts of actual degradation value of *Response time*, *Cost*, *Reliability*, and *Availability*. And then a

4**n* matrix *G* is built by the four sorts and denoted as $[\Delta_T[], \Delta_C[], \Delta_R[], \Delta_A[]]^T$. Every column in the matrix is assigned to a weight. The weight of the *j*-th column is set to $(n\text{-}j+1)/n$.

c) to set effective weight of S_1,…, and S_n. Effective weight of S_i is a vector WE_i=(WE_{Ti}, WE_{Ci}, WE_{Ri}, WE_{Ai}). WE_i components are the column weights of S_i in *G*.

Thus, relative QoS Degradation Value DV_i of S_i equals the sum of all components of its effective weights. The formula is as follows:

$$DV_i = \frac{WE_{Ti} + WE_{Ci} + WE_{Ri} + WE_{Ai}}{4} \tag{2}$$

3 Evaluation

In order to evaluate the efficiency and effectiveness of our proposed method, three groups of test are conducted.

We use a service test collection from JTangComponent previously built in [14] where 1056 services have been included to generate the needed application in the test. In addition, in order to support the test, QoS of candidate services is simulated and produced by the following way: *Cost* and *Response Time* are randomly generated with a uniform distribution from 1 to 100, *Availability* and *Reliability* are randomly generated with a uniform distribution from 0 to 1, are assigned to each candidate service.

In our experiment, we have generated one service based application *P* including sequential, parallel, choose, and loop structures in our simulation study. *P* includes 20 nodes and 6 structures in Fig.1. For each service node in *P*, we provide 10 service candidates with four randomly generated QoS values. Randomly select 1 or 2 services in *P* to be failure services.

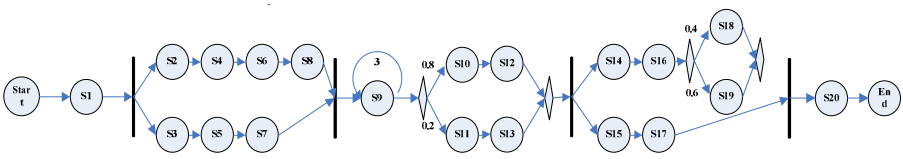

Fig. 1. Service based Application *P*

We compare the proposed approach with two other methods: Region-based method[12] and Random method. The idea of region-based approach is to produce reconfiguration regions that include one or more failed service. By reconfiguring only services in the selected regions, the business process will not be affected significantly. Random method is to select replaceable component services randomly and replace them. In order to study the performance of the three methods, we consider the following factors: attempt time, replaced services number, and recovery time.

Table 1 reports the performance of the three methods. In Table 1, all factors of our approach are better than other two approaches obviously in the given cases.

Table 1. Performance Comprision

Case Study	**1**			**2**		
Method	Our Method	Region based approach	Random Method	Our Method	Region based approach	Random Method
Failure services	s_1			s_{15}		
Attempt times	2	15	14	2	11	10
replaced services number	2	9	14	2	5	10
recovery time(ms)	15	105	71	11	100	57
Case Study	**3**			**4**		
Method	Our Method	Region based approach	Random Method	Our Method	Region based approach	Random Method
Failure services	s_{12},s_{13}			s_{16},s_{17}		
Attempt times	8	17	9	4	12	6
replaced services number	8	7	9	4	6	6
recovery time(ms)	16	88	19	22	105	9
Case Study	**5**			**6**		
Method	Our Method	Region based approach	Random Method	Our Method	Region based approach	Random Method
Failure services	s_1,s_{11}			s_9,s_{12}		
Attempt times	3	37	6	3	36	12
replaced services number	3	8	6	3	8	12
recovery time(ms)	19	349	32	10	160	22

4 Related Work

In dynamic environments, service composition needs to support to recover services based applications from unexpectable violation of not only function but also QoS. Therefore, holding the original overall QoS constraints of services based applications has proposed a big challenge that needs to be addressed.

Many works have studied QoS guarantee of service selection problem[16,17,19]. In addition, many researchers have been studied QoS optimization [18]or reoptimiaztion[11,19] of service compostion. Danilo Ardagna et al. The main difference between the above works and our work is that besides end-to-end QoS constraints, their work focuses more on service composition evolution and selection in order to optimize overall QoS, while we emphasize more on the efficiency of reconfiguration to recovery of services based applications when degradation of overall QoS.

Some researchers have studied dynamic reconfiguration of services based applications but without considering QoS[6-8,14]. Our previous work[15] also tried to dynamically reconfigure services based applications to satisfy customer's QoS constraints. But its goal is to improve overall QoS of services based applications, and it is not able to make services based applications hold their original QoS. Bo Jiang,et.al[10] proposed a statistical framework to assess component services and to identify vulnerable areas called cracks to support service adaptation. Be different than our method, their method is to find potential component services which can lead to failure of key business of services based application. But this may become one of our future research topics.

Recent works on dynamic reconfiguration for service composition has started to study in order to hold the original overall QoS constraints. T.Yu et.al.[13] presented an approach to conduct dynamic process reconfiguration under end-to-end QoS constraints. They use the replacement path idea to reconfigure a business process to avoid only one faulty service. Yanlong Zhai et al. [12] presented an approach for repairing multiple failed services by replacing them with new services and ensuring the new system satisfies the end-to-end QoS constraints. Compared to our work, their works limit to recovery when component services become faulty, while it becomes invalid when the delivered QoS of component services degrades. Furthermore, our test has shown the performance of our method is better than them.

5 Conclusion

Due to failed services or degradation of component QoS, original QoS of services based applications may be broken. Once that happens, it is undesirable to halt and recompose services based applications. Services based applications should be recovered as soon and as efficiently as possible. The paper proposes a QoS driven dynamic reconfiguration method to maintain the original QoS of services based applications. The key of our method is degradation factor of component services which can guide us to find the component services which are the most contribution to the violation of overall QoS. The results of our evaluation show that our method can recover the original overall QoS by reconfiguring only a small number of services with fewer attempts in acceptable time.

Acknoledegments. This research is partially supported by the National Technology Support Program under grant of 2012BAH16B04, Zhejiang Provincial Natural Science Foundation of China under grant of LY12F02029, and the National Natural Science Foundation of China under grant of 61100043.

References

1. Zhang, L., Zhang, J., Cai, H.: Services Computing. Springer & Tsinghua University Press (2007)
2. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. on Software Engineering 30(5), 311–327 (2004)

3. Brian Blake, M., Tan, W., Rosenberg, F.: Composition as a Service. IEEE Internet Computing (INTERNET) 14(1), 78–82 (2010)
4. Aoyama, M., Weerawarana, S., Maruyama, H., Szyperski, C., Sullivan, K., Lea, D.: Web Services Engineering: Promises and Challenges. In: Proc. ICSE 2002, Orlando, pp. 647–648 (2002)
5. Vambenepe, W., Thompson, C., Talwar, V., et al.: Dealing with Scale and Adaptation of Global Web Services Management. Int. J. Web Service Res. (3), 65–84 (2007)
6. Tsai, W.T., Song, W., Chen, Y., Paul, R.: Dynamic System Reconfiguration Via Service Composition for Dependable Computing. In: Kordon, F., Sztipanovits, J. (eds.) Monterey Workshop 2005. LNCS, vol. 4322, pp. 203–224. Springer, Heidelberg (2007)
7. Avgeriou, P.: Run-time Reconfiguration of Service-Centric Systems. In: Proceeding of the European Pattern Languages of Programming, EuroPLOP (2006)
8. Ezenwoye, O., Busi, S., Sadjadi, S.M.: Dynamically Reconfigurable Data-intensive Service Composition. In: WEBIST 2010, pp. 125–130 (2010)
9. Yan, Y., Poizat, P., Zhao, L.: Repair vs. Recomposition for Broken Service Compositions. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 152–166. Springer, Heidelberg (2010)
10. Jiang, B., Chan, W.K., Zhang, Z., Tse, T.H.: Where to adapt dynamic service compositions. In: WWW 2009, pp. 1123–1124 (2009)
11. Xiong, P., Fan, Y.S., Zhou, M.C.: Web Service Configuration under Multiple Quality-of-Service Attributes. IEEE Trans. on Automation Science and Engineering, 311–321 (2008)
12. Zhai, Y.L., Zhang, J., Lin, K.-J.: SOA Middleware Support for Service Process Reconfiguration with End-to-End QoS Constraints. In: The IEEE International Conference on Web Services (ICWS), pp. 815–822 (2009)
13. Yu, T., Lin, K.J.: Adaptive algorithms for Finding Replacement Services in Autonomic Distributed Business Processes. In: Proc. of the 7th International Symposium on Autonomous Decentralized Systems (2005)
14. Yin, Y.Y., Li, Y., Yin, J.W., et al.: Ensuring Correctness of Dynamic Reconfiguration in SOA Based Software. In: 2009 Congress on SERVICES-I, pp. 599–606 (2009)
15. Li, Y., Lu, Y.L., Yin, Y.Y., et al.: Towards QoS-Based Dynamic Reconfiguration of SOA-Based Applications. In: APSCC 2010, pp. 107–114 (2010)
16. Zeng, L., Ngu, A., Benatallah, B., Podorozhny, R., Lei, H.: Dynamic composition and optimization of web services. Distrib. Parallel Databases 24(1-3), 45–72 (2008)
17. Zheng, H., Yang, J., Zhao, W.: QoS Analysis and Service Selection for Composite Services. In: IEEE SCC 2010, pp. 122–129 (2010)
18. Rosenberg, F., Müller, M.B., Leitner, P., Michlmayr, A., Bouguettaya, A., Dustdar, S.: Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions. In: IEEE SCC 2010, pp. 97–104 (2010)
19. Yu, T., Zhang, Y., Lin, K.-J.: Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. ACM Transactions on the Web 1(1), 1–26 (2007)