An Adaptive Algorithm for Failure Recovery During Dynamic Service Composition*

Xingzhi Feng, Huaimin Wang, Quanyuan Wu, and Bin Zhou

School of Computer, National University of Defense Technology,
410073 Changsha, China
billytree@gmail.com

Abstract. During the execution of Web Service Composition, if one component service fails or becomes overloaded not to be accessed, a failure recovery mechanism is needed to ensure that the running process is not interrupted and the failed service can be replaced quickly and efficiently. Recent researches on this problem have some disadvantages. They don't consider the influence of the number of service candidates or the connection state of the overlay network, so the algorithms are easily disabled. In this paper, we present an adaptive algorithm to find replacement path locally by virtue of the old path during dynamic service composition. We go backward along the execution path to find the branch node, and then construct the sub-graph by the predefined rules. Finally we choose the best path with the highest total utility to replace the failed one. The test's result shows the algorithm performs very well in the vigorousness and availability to dynamic adaptation.

1 Introduction and Related Work

Dynamic Web Service Composition (DWSC) organizes the component services across the different autonomic domains to finish complex application requirements by business planning. But for the incontrollable features of service providers and network connection of the component services in the Internet, the component services along execution path easily can fail during the runtime course while we have laid out the business process well in the design stage. So the running process is interrupted and couldn't be finished along the original execution path. A failure recovery mechanism is needed to ensure that a replacement path is found quickly and the business process can continue. Also the dynamic replacement problem is often considered in software upgrading and maintenance, because dynamic upgrading with minimal disruption to the consumers is very important feature for high-availability, mission-critical, and hard real-time systems.

* This work was supported by the National High-Tech Research and Development Plan of China under Grant Nos. 2003AA115210, 2003AA115410, 2005AA112030; the National Grand Fundamental Research 973 of China under Grant No.2005CB321800; the National Natural Science Foundation of China under Grant Nos. 60603063, 90412011.

A. Ghosh, R.K. De, and S.K. Pal (Eds.): PReMI 2007, LNCS 4815, pp. 41–48, 2007.

The QoS values of the single component services used for computation may be estimations in turn, declared by each service provider or obtained by computing statistics during previous executions of the service. At run-time, the actual measured QoS values may deviate from the estimate ones or, simply, some of the services may not be available (fail or become overloaded not to be accessed). Thus the composite service may have to be re-planned, so to still meet the constraints and maximize the QoS. In this time, quite a few component services perhaps have executed. If we just find an entirely new execution path over again. Although the new path is globally optimal but needs longer time to search, higher cost and delay unfit for the real-time application requirements. In this paper we consider the replacement solution in local path; assure the new path has the largest overlap with the old one with multiple QoS constraints. Thus the replacement process can be quickly finished with lower delay and cost. We select the optimal service candidates with higher utility to construct new replacement path. It makes sure that new path has the largest total utility with multi-dimensional QoS constraints assurance.

Many researchers have worked on the adaptive and dynamic service composition problem. The eFlow system is enacted by a centralized process engine, supports specification, enactment and management of composite services [1]. Service processes can adapt to environment changes and dynamically modify the process definition with minimal or no user intervention. However, eFlow does not have the rules for automatically switching service when service failure occurs. The reconfiguration (process definition modification) requires the user intervention and no quality issues are considered. In [2], authors develop Web Service Offering Infrastructure (WSOI) and proposed the idea of service offering similar to service levels defined in [3][4]. They address dynamic adaptation of service composition using manipulation of service offerings or classes via five mechanisms: switching, deactivation, reactivation, deletion, and creation. These methods are simple and fast but the ability is very limited. If we consider the end-to-end quality constraint issues of business processes, it is usually not enough to just switch among the service offerings within one Web service. The project in [5] majors in dynamic software upgrading with minimal disruption to consumers. In [6] they present the CARIS architecture and introduce application to dynamic composition of on-demand security associations. They address dynamic service composition supports business agility, flexibility, and availability. Same as eFlow, no algorithm has been designed to select replacing services. However, this is one of the most fundamental and important problems for the dynamic adaptation in Web service composition.

In [7-9], they study the end-to-end quality issue of business processes, model the system as a Directed Acyclic Graph (DAG) and convert the service composition problem to the problem finding the highest utility path in the graph theory under the multiple QoS constraints. The business process constructed can produce the highest user-defined utility as well as satisfy user's functional and quality requirements. In [10] [9], they present a replacement path algorithm to find the shortest path from source to target in graph G-{e_i}(G without e_i), where e_i is an edge in the original shortest path of graph G. The algorithm proposed in [10] can produce n replacement paths with the same time complexity as the single-source shortest path algorithm. Our algorithm is

similar to the algorithm in [9]. But our algorithm has no special requirement for the network topology, so it has more flexibility and availability than the algorithm in [9].

2 QoS Evaluation Model and QSC Problem

During the process of service composition, it should consider not only the functionality and behavior of service but also the nonfunctional properties, especially about the QoS metrics, which normally include response time, service time, availability, reliability, service cost and loss probability etc.

Some researchers have studied the replacement algorithm in local path. SpiderNet [8] removes all the other service candidates in the same Service Class that the old service performs well in the execution path. It is not suitable if the output degree of the immediate predecessor of the failed service equals to 1. QCWS [9] provides a backup path for each service node in the original execution path. The immediate predecessor of a failed service may quickly switch to a predefined backup path, so the running process is not interrupted. But this predefined backup path is unfit for the dynamic business process. They assume that if the output of service s_i is used as input of service s_j, add directed edges from all service level nodes in s_i to all service level nodes in s_j. It is too rigorous for each service node in the graph must be included in at least two paths. It's reasonable that not all service level nodes in s_i must be connected with all service level nodes in s_j.

In this paper we propose an algorithm that can solve the above shortages of other algorithms. If the component service fails while the composite service has executed half way, we will find the branch node backward along the execution path. Then we use current branch node as new source node, prune out the irrelevant service nodes by the algorithm and remove all edges whose start or end nodes reside outside of the sub-graph. According to this sub-graph, we construct the composition sub-tree. There is a list in each branch node recording the paths begin from it. We select the optimal path with highest utility as the replacement path with multi-dimensional QoS constraints assurance. We use the backward chaining algorithm to construct the sub-graph. And we make use of the utility function as the evaluation criteria.

The utility function is the evaluation criteria of QoS during service selection. The computation of the utility function should consider each QoS metric of the candidates in order to reflect the QoS of the service candidates deeply and thoroughly, such as response time, service time, availability, reliability, service cost and loss probability, etc. QoS metrics such as availability and reliability are benefit property, which need to be normalized by maximization. And other metrics such as response time and service time are cost property, which need to be normalized by minimization. For there exist some differentiations such as metrology unit and metric type between each QoS metric. In order to rank the Web Services fairly we must do some mathematics operations to normalize the metrics in the range [0,1]. Subsequently we can get a uniform measurement of service qualities independent of units.

Definition 1 (Service Candidate Utility Function). *Suppose there are α QoS metric values to be maximized and β QoS metric values to be minimized. The utility function for candidate k in a Service Class is defined as following:*

$$\mathcal{F}(k)=\sum_{i=1}^{\alpha} w_i \times(\frac{q_{ai}(k)-q_{ai_{\min}}}{d_i})+\sum_{j=1}^{\beta} w_j \times(\frac{q_{bj_{max}}-q_{bj}(k)}{d_j}) \quad (1)$$

$$d_i = q_{ai_{max}} - q_{ai_{\min}},\ if\ q_{ai_{max}} - q_{ai_{\min}} \neq 0,\ else\ (q_{ai}(k) - q_{ai_{\min}})\big/ d_i = 1,$$
$$d_j = q_{bj_{max}} - q_{bj_{\min}},\ if\ q_{bj_{max}} - q_{bj_{\min}} \neq 0,\ else\ (q_{bj_{max}} - q_{bj}(k))\big/ d_j = 1.$$

Where w is the weight for each QoS metric set by user and application requirements ($0 < w_i, w_j < 1$; $\sum_{i=1}^{\alpha} w_i + \sum_{j=1}^{\beta} w_j = 1$, $\alpha+\beta=m$). d_i, d_j *are the difference between maximum value and minimize value in the column. The concrete QoS metrics are defined in the matrix QoS=* $[q_{i,j}]_{n\times m}$. $q_{ai\ max}$ ($q_{bj\ max}$) *is the maximum value among all values on column i (j) in submatrix* $[q_{a\ k,i}]_{n\times\alpha}$ ($[q_{b\ i,j}]_{n\times\beta}$) *and* $q_{ai\ min}$ ($q_{bj\ min}$)*is the minimum value among all values on column i (j) in submatrix* $[q_{a\ k,i}]_{n\times\alpha}$ ($[q_{b\ i,j}]_{n\times\beta}$).

Each row in QoS matrix represents a Web service candidate sl_i ($1\le i\le n$, and n represents the total number of candidates) while each column represents one of the QoS metrics q_v ($1\le v\le m$, and m represents the total number of QoS metrics). As shown in matrix, QoS metrics of a Web Service candidate can be modeled as *m-dimensional* vectors; each Web service candidate can be represented as a point in m-dimensions.

We also consider the network QoS attributes (QoS metrics of the edge) such as bandwidth, network delay and loss probability. Because the QoS attributes of the network are only estimated, we can use indeterminacy reasoning with the degree of confidence and plausibility. Thus each service node has different utility value with different service link.

Definition 2 (Edge Utility Function). *Let LP denotes loss probability, D denotes network delay, BR denotes bandwidth ratio. The utility function of the edge* e_i= (u,v) *in service graph G is defined as follows:*

$$\mathcal{F}(e_i)=w_1 \times \frac{\ln 1/1-LP_{e_i}}{\ln 1/1-LP_e^{min}} + w_2 \times \frac{D_{e_i}}{D^{max}} + w_3 \times \frac{BR_{e_i}}{BR^{max}} \quad (2)$$

where $\sum_{i=1}^{3} w_i = 1,\ 0 \le w_i \le 1, i = 1,2,3$ *the loss probability metric* LP^{min} *denotes the minimum value among all values. And* $D_{e_i} \le D^{max}$, $BR_{e_i} \le BR^{max}$.

Based on the above two definitions, the QoS-aware Service Composition (QSC) problem is defined as follows:

Definition 3 (QoS-aware Service Composition (QSC) problem). *Given overlay network G = (V,E), where V denotes the set of |V| nodes* v_i *and E denotes the set of |E| overlay links* e_i. *Let* $Fn(s_i)$ *denotes the business function provided by the service candidate* s_i, s_i/v_j *denote the service candidate* s_i *provided by the node* v_j, *and* e_i/p_j *denote the service edge* e_i *instantiated on the overlay path* p_j. *Each node* $v \in V$ *is associated with m non-negative QoS values; each edge* $e \in E$ *is associated with n*

non-negative QoS values. Given a user request <ξ, Q^{req}>, the QSC problem is to find the best service graph λ□ {λ_s, λ_e} such that

$$max\,(\mu_s \times \sum_{s_i / v_j \in \lambda_s} \mathcal{F}(s_i) + \mu_e \times \sum_{e_i / p_j \in \lambda_e} \mathcal{F}(e_i)\ ) \tag{3}$$

Subject to $\forall F_k \in \xi, \exists s_i \in \lambda_s, Fn(s_i) = F_k, q_i^{\lambda} \le q_i^{req}, 1 \le i \le m$. *Where* $\mathcal{F}(s_i)$ *and* $\mathcal{F}(e_i)$ *are the utility values of service candidate and edge, which can be computed by Def. 1 and Def. 2 respectively.* μ_s *and* μ_e *are their weights respectively.*

A composite service may have several execution paths. The overall QoS value of the execution path can be calculated by the sum of the QoS values of selected component services. Therefore the optimal execution path is the maximum one.

Since the Service Flow Model is not related to any concrete Web Service, we must assign appropriate services to the corresponding activities to construct execution paths. And this is also the procedure to generate weighted multistage graph. We use the compatible rules to build up the service composition graph that we presented in another reviewing paper. The QoS value of service candidate and edge are used to set the weight of edges in the DAG.

3 QoS-Based Algorithm for Finding Replacement Service

During runtime an established service path can become broken or violate QoS constraints, particularly for the long-lived application session. When the service instance (or link) experiences outage or significant quality degradations, the dynamic service failure recovery mechanism is used to recover all the affected sessions. We define the Dynamic Failure Recovery (DFR) Problem as follows:

Definition 4 (Dynamic Failure Recovery (DFR) Problem). *Let* λ^{old} *and* λ^{new} *denote the broken service graph and the new service graph, respectively. Let* $|\lambda_s^{old} \cap \lambda_s^{new}|$ *denote the number of common service candidates between the old service graph and the new service graph. Given the overlay network G=(V,E) and the user request <ξ,* Q^{req}*>, the DFR problem is to find a new service graph* λ^{new} *such that maximize* $|\lambda_s^{old} \cap \lambda_s^{new}|$, *subject to* λ^{new} *satisfies* $\forall F_k \in \xi, \exists s_i \in \lambda_s, Fn(s_i) = F_k$, $q_i^{\lambda} \le q_i^{req}, 1 \le i \le m$.

The following algorithm is designed according to the above DFR problem. Our algorithm decomposes the candidate graph as composition sub-graph, and then transforms it into composition sub-tree. The replacement path begins from the branch node, backward along the original execution path of the failed service. We consider the special case: If the output degree of the predecessor s_p of the failed service s_f is equal to 1, i.e. $d_{out}(s_p)$ =1, then the algorithm would continue to find the branch node s_b backward sequentially. While this occasion would make the algorithms break down in [8][9]. The WSCPR algorithm is showed in Algorithm 1 in great details.

Algorithm 1. WSCPR(Web Service Composition Partial Re-composition) Algorithm

Step 1. From the failed node s_f backward to find the branch node s_b whose output degree $d_{out}(s_b)>1$;
Else no feasible node found, stop.
Step 2. For each Service Class S_i from branch node to last Service Class
If $s_f \in S_f$
// remain all other service nodes in the Service Class of the failed service s_f
continue;
Else if the predecessor s_p of s_f, $d_{out}(s_p)=1$ **or**
the successor node s_s of s_f, $d_{in}(s_s)=1$
continue;
Else
Remove all the other service nodes in the class S_i but not in the original execution path;
Remove the failed node;
Step 3. Construct the sub-graph;
Step 4. Transform the sub-graph into composition sub-tree;
Add all paths in the sub-tree into $p\_list(s_b)$, the list of the branch node s_b;
Select the best optimal path according to computation of the utility function as the new execution path.
$\mathcal{F}_{max}=0$;
for each $p \in p\_list(s_b)$
$\mathcal{F}(p)=\sum \mathcal{F}(s)$, s is all the services along the path p;
$\mathcal{F}_{max}=max\{\mathcal{F}_{max},\mathcal{F}(p)\}$;
Return the path p with the highest utility in $p\_list(s_b)$ as optimal replacement path.

Using WSCPR algorithm, we suppose the number of QoS requirements is m, the number of Service Classes is N and each class has l candidates. Then the worst case time complexity for WSCPR is $O(N^2lm)$, it performs very well in practice.

4 Validation of the Algorithm

We have performed the simulations and experiments to see the performance of our WSCPR algorithm. We also compare the running time between our WSCPR algorithm and CSPB algorithm in [9].

We use the degree-based Internet topology generator *nem* (network manipulator) [11] to generate a power-law random graph topology with 4000 nodes to represent the Internet topology. Then we randomly select 50~500 (depends on different test cases) nodes as the service candidate nodes in a business process. Some service nodes belong to the same Web service, representing different service levels. Several Web services group together to form a Service Class and several Service Classes in sequential order form an execution path. Each service node maintains the following information: *Service Class, service* and *execution path* it belongs to. The graph is

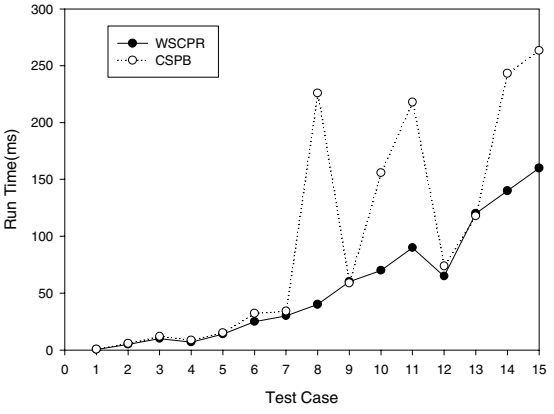

Fig. 1. Performance comparison between WSCPR and CSPB algorithm

constructed by randomly generating edges among service nodes. The generated graph is a DAG. The generated graph contains one or more execution paths.

The performance comparison between WSCPR and CSPB algorithm is showed in Fig. 1. We can see that the running time of WSCPR algorithm is reduced at a rather extent in contrast with CSPB algorithm. CSPB algorithm just uses another backup path when some node fails. And CSPB algorithm doesn't consider the errors in the dynamic environment, such as: the Internet connection fails during the SOAP message round trip; the WS times out because of a network connection failure; the Web Service returns a failure message because of data inconsistency. So the CSPB algorithm is easily disabled as showed in our test. Our WSCPR algorithm excludes quite a number of nodes independent of the final solution from the set of the service candidates. Thus the search space can be reduced at great certain extent.

References

1. Casati, F., Ilnicki, S., Jin, L., et al.: Adaptive and Dynamic Service Composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, Springer, Heidelberg (2000)
2. Tosic, V., Ma, W., Pagurek, B.: On the Dynamic Manipulation of Classes of Service for XML Web Services. In: Proceedings of the 10th Hewlett-Packard Open View University Association (HP-OVUA) Workshop, Geneva, Switzerland (July 2003)
3. H.G., C., Yu, T., Lin, K.-J.: QCWS: An implementation of QoS-capable multimedia web services. In: Proceedings of IEEE 5th International Symposium on Multimedia Software Engineering, Taiwan, pp. 38–45 (December 2003)
4. Tosic, V., Mennie, D., Pagurek, B.: Software Configuration Management Related to Management of Distributed Systems and Services and Advanced Service Creation. In: Westfechtel, B., van der Hoek, A. (eds.) SCM 2001 and SCM 2003. LNCS, vol. 2649, pp. 54–69. Springer, Heidelberg (2003)
5. Feng, N., Ao, G., White, T., et al.: Dynamic Evolution of Network Management Software by Software Hot-Swapping. In: Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM 2001), Seattle, USA, pp. 63–76 (May 2001)

6. Tosic, V., Mennie, D., Pagurek, B.: On dynamic Service Composition and Its Applicability to E-business Software Systems. In: Proceedings of the WOOBS 2001 (Workshop on Object-Oriented Business Solutions) workshop (at ECOOP 2001), pp. 95–108 (2001)
7. Yu, T., Lin, K.J.: Service Selection Algorithms for Web Services with End-to-end QoS Constraints. In: Proceedings of the IEEE International Conference on E-Commerce Technology (CEC 2004), San Diego, California, pp. 129–136 (2004)
8. Gu, X., Nahrstedt, K., Yu, B.: SpiderNet: An Integrated Peer-to-Peer Service Composition Framework. In: Proceedings of 13th IEEE International Symposium on High performance Distributed Computing(HPDC 2004), Honolulu, Hawaii, pp. 110–119 (June 2004)
9. Yu, T., Lin, K.-J.: Adaptive Algorithms for Finding Replacement Services in Autonomic Distributed Business Processes. In: Proceedings of the 7th International Symposium on Autonomous Decentralized Systems (ISADS 2005), Chengdu, China, pp. 427–434 (April 2005)
10. Hershberger, J., Suri, S.: Vickrey Prices and Shortest Paths: What is an Edge Worth? In: FOCS 2001. Proceedings of the 42nd IEEE symposium on Foundations of Computer Science, pp. 252–259. IEEE Computer Society, Los Alamitos (2001)
11. Magoni, D., Pansiot, J.-J.: Internet Topology Modeler Based on Map Sampling. In: ISCC 2002. Proceedings of the Seventh International Symposium on Computers and Communications, Taormina, 1-4 July 2002, pp. 1021–1027. IEEE Computer Society, Los Alamitos (2002)