Optimized Composite Service Transactions through Execution Results Prediction

Jiuyun Xu, Zhaotong Li,Huanxing Chi
Muhan Wang, Chao Guan
School of Comp. & Commun. Eng.
China University of Petroleum
Qingdao, Shandong 266580
Email: jiuyun.xu@ieee.org
lizhaotong@s.upc.edu.cn

Stephan Reiff-Marganiec
Department of Computer Science
University of Leicester
LE1 7RH, Lecester, UK
Email: srm13@le.ac.uk

Huilin Shen
School of Geoscience
China University of Petroleum
Qingdao, Shandong, China 266580
email:hlshen@upc.edu.cn

***Abstract*—Traditional web services transaction processing mechanism handle exception by forward recovery and backward recovery. These compensation mechanisms often lead to waste of resources and time. In this paper, we propose a framework for predicting outcomes of service executions as part of service compositions which allows to choose service instances that are likely to lead to a successful result in the first instance and thus reduces the need for invoking costly recovery mechanisms. The framework makes use of watchdogs to maintain an awareness of service availability and a pre-coordinator which has oversight of the whole composite Web service and acts as a control center. An analysis of a scenario shows that we cannot only provide users with a more satisfactory result, but also can reduce the overhead costs of resources and waste.**

***Keywords*-Transaction processing; Flow prediction; Skyline;**

I. Introduction

Web service technology has become a widely deployed basis for distributed applications. It provides users with a loosely coupled, cross-platform heterogeneous application integration mechanism allowing for rapid changes to systems and easy integration of third party functionality. Complex systems are an economy of independently provided Web services that collaborate to achieve a larger goal – referred to as composite (Web) services. However, this kind of collaborative, distributed computing and resource sharing approach to running systems can inevitably lead to faults and unexpected results occurring at runtime; which in turn leads to a decrease of the systems reliability and consistency. In order to ensure that the composite Web service is executed correctly and achieves the overall desired result, it is necessary to provide transactions support for web services. At present, Web transaction mechanisms have been developed to mimic traditional transaction mechanisms, however accommodations have been made to address specific needs in Web service transactions. To name a few such needs: complex services can be long running; 'undoing' is not always possible (at least not without a potential cost), and transactions might be nested in various ways across systems that operate in parallel. In general this means that the ACID properties that are seen as highly desirable for database transactions do not all hold in this space. However, the quite flexible nature of the systems also allows for new approaches to overcome problems, such as replacing a failed service with a different one. In the light of these common recovery mechanisms in Web service transactions include forward and backward recovery. Forward recovery attempts to reach the original goal of the composite service by retrying or replacing components and continuing the process; backward recovery is essentially a form of rollback that unrolls the transaction and restores the original state of the system – or at least as closely as possible. The last point refers to the fact that sometimes a compensation mechanism has to be invoked to undo a step which might come at a cost itself.

To exemplify these issues, let us consider a composite service for booking a holiday: An airline ticket might be booked in an early stage followed by a hotel booking attempt – it turns out that no hotel is available and hence the airline ticket needs to be cancelled. Such a cancellation might mean that a percentage of the original cost is lost to the user as a penalty. There might also be other change to the state, especially if physical tickets have been posted – maybe these need to be returned which is a different aspect that was not part of the original process.

As should be obvious from the above explorations, the cost of a failed transaction can be significant; even if in the case of forward recovery the original aim is achieved there might have been a cost incurred due to recovery issues in the process and there certainly would have been extra time added to achieve the aim.

In this paper, we present a novel approach to increase the success of transactions. The proposed framework allows to gather an understanding of the likely success of the transaction, which is used for instantiating the abstract services in the process and to guide the execution. Data about individual service candidates will be provided watchdog runtime monitoring modules and the search space for the instantiation problem is pruned using a skyline approach [1] to ensure that we can efficiently find the optimal instantiation.

We implemented and tested the approach with an online ticket booking system in the travel domain as example. Travel booking is often used as an example for composite services and has drawn criticism. However, for our purpose

the travel booking is ideal as it shows the complexities required (it is heavily dependent on the output data from services and there are many alternatives and the operating environment is very dynamic) while being easy to explain.
The remainder of the paper is organized as follows: Section 2 provides an overview of the framework, Section 3 describes the proposed transaction mechanism for composite web service. A case study is presented in section 4 together with some experimental analysis. Section 5 discusses the related work and Section 6 concludes the paper.

II. The Proposed Framework

The proposed framework aims to achieve a higher success rate in successfully completing composite services at the first attempt, thus reducing reliance on compensation mechanisms. With a focus on transactions, we are looking at runtime executions and problems occurring at that stage rather than planning type problems.
When executing composite services in the framework the input is an abstract process instance, that is a process that describes the structure of the problem, but in which no concrete service instances for fulling the tasks have been bound. For our work we have used an abstract BPEL process, but the methods presented are independent of a specific implementation mechanism.
We further assume that for each Web service (abstract and concrete) a profile exists which – as a specification – describes the precondition, results, postcondition and some declared constrains expressed as logical expressions. Additionally, concrete Web services, also have a description of their QoS attributes.
The next two subsections will discuss the structure of the framework as well as the process when a composite service is triggered respectively.

A. Structure of the Framework

The main components of the framework are users, services, watchdogs and the pre-coordinator – however users and services are clearly not deeply embedded. An overview of the framework structure is shown in Figure 1.
A user interacts with the system by providing their abstract composite service and a description of their expectations (overall costs, time constraints, and other configuration aspects for the composite service). The process, once instantiated, will be executed through Web services which are expected to already exist and to be generally available.
The two core components of the framework are the pre-coordinator and the watchdog modules, detailed as follows:

- **Pre-coordinator module**: this module forms the central part of the system. It is in charge of receiving the user request, deciding on concrete services and instantiating the process, and finally starting the concrete process with the parameters provided by the users.
- **Watchdog module**: each service has an associated runtime monitoring module, which can record the data of the service and the restrictions. The watchdog module interacts with the Pre-coordinator to provide information on the service and can temporarily hold resources of the service.

Note that the mentioned holding of resources by the watchdog is only in relation to information available to the pre-coordinator – if the service is invoked through a different avenue then the service information will change in the watchdog, and might become temporarily out of sync with the information passed to the pre-coordinator.

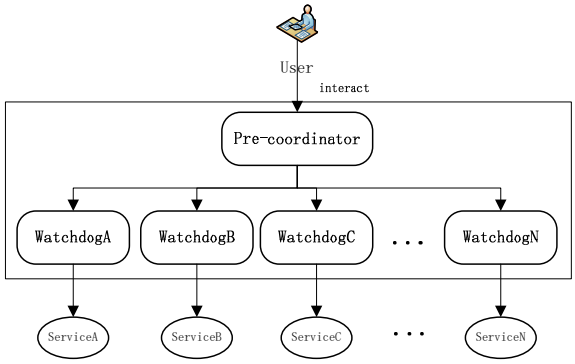

Fig. 1: Chart of the module structure

B. Process for Managing a Composite Service

The main communications in the framework are between the watchdogs and the Pre-communicator, the former interact with the services and the latter with the user. Services might directly interact with the user (as they would if they were not part of the framework) and we do not wish to take control over that. More importantly there is no need for the individual watchdogs to coordinate with each other, keeping dependencies at a minimum.
The overall process for managing a composite service consists of two parts: a background process between each watchdog and its associated service and the process of receiving a new user instruction. Dealing with a user instruction involves getting the composite service instantiated and executed.
Background Process:
Every Web service has an associated watchdog process which is monitoring the service. We are not concerned with the exact details of the monitoring in this paper as work on runtime monitoring exists. We rather assume that a watchdog is able to provide, when asked, details about the service's situation, which includes the logical expression of precondition, postcondition, constrain and possible other aspects of the service.
Note that run-time monitoring of web services is non trivial; some work in this area has been conducted (e.g. [2]) and some tools are available (usually additions to network monitoring tools).
User Instruction:
Step 1:. A user provides a new abstract composite service description together with invocation parameters and requirements (such as maximal cost and duration) to the Pre-coordinator (PC), thus starting a new process.

TABLE I: Requirements of four Users

Name of demond	User1	User2	User3	User4
Account()	1300	1005	1220	400
Num_ticket()	1	5	4	2
Num_room()	1	3	2	1
$WantTime()$	6	7	6	5
$WantWaitTime()$	2	3	1	2
$WeightofTime$	1	2	1	2

TABLE II: Description of Precondition and Potential Result of Web services used in the example

ServicesName	Preconditions&Results	guarantees
ServC	$Num\_acc() \geq 700$, $Num\_ticketCF() = 9$, $Num\_ticketCG() = 12$, $Num\_ticketCH() = 33$,	$Account() \leq 300$, $Time() \leq 5$
ServF	$Num\_acc() \geq 500$, $Num\_roomFI() = 13$, $Num\_roomFJ() = 4$	$Account() \leq 250$
ServG	$Num\_acc() \geq 500$, $Num\_roomGJ() = 1$,	$Account() \leq 250$
ServH	$Num\_acc() \geq 500$, $Num\_roomHJ() = 8$,	$Account() \leq 250$

Step 2: The PC contacts watchdogs and extracts information about the relevant services.
Step 3. The PC applies a skyline algorithm with transaction pro-detecting to quickly filter service candidates to identify the most promising ones for closer analysis.
Step 4. The PC selects concrete service candidates to optimize the overall selection and instantiates the service.
Step 5. The PC starts the concrete process instance.

III. Case Study

We used the travelling scenario as an example, as it contains the main aspects of concern to us namely a wide range of services available, an easy to explain need to compose services and a dependence on the results which are based in real resources (e.g. limited seats on a plane).
Typical user requirements are shown in Table II: how much money is available for the trip, numbers of rooms and tickets needed and criteria related to time are upper limits on the timing QoS criteria of the composed service. Typical service descriptions are provided in Table II showing the description of the precondition, potential results and the guarantees the service gives (in terms of upper limits on costs and time).
Service selection methods in the literature are either approaches aiming to achieve global optimal solutions (selecting services for a whole process with a view to optimizing the overall QoS) or approaches that try to make the best decision for an individual service in the environment and context that it is being invoked – which computationally is much simpler. In this paper, we adopt the latter to select the suitable concrete web services for each abstract service.
We use skyline optimization to identify suitable service candidates quickly. As it is a multiple objective optimization method it allows us to consider the criteria we like to optimize over. For our example we frame the candidate service set as a double objective optimization problem based on travel time and cost. We obviously know that time and money interact in such a way that faster approaches are usually more expensive.

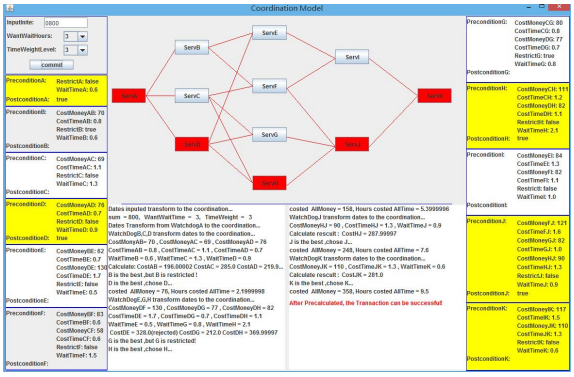

Fig. 2: Transaction flow nodes

Each transaction node has a watchdog module, which records parameters about the node. Let us look at an example for node F where the data might be the cost in terms of time or money from one node to the next $CostTimeCF$, $CostTimeBF$ (time overhead from C or B respectively to F), $CostMoneyCF$, $CostMoneyBF$ (financial overhead) and waiting time or restrictions at F $WaitTimeF$, $RestrictF$. Restrictions could be no availability of tickets, traffic jams and other emergencies. $RestrictF$ is a boolean indicating whether a restriction exists or not.
Note that we use a simple variant for $RestrictF$ here; in general a more detailed type could be used and evaluated as maybe not all limiting conditions are quite as black and white as captured here.
Suppose the Pre-coordinator has calculated to select the node $ServC$ from $ServA$ and thus now makes node $ServC$ the basic point for further considerations and service $ServF$, $ServG$, or $ServH$ are potential next steps.
First, we calculate the skyline ($CostTime$, $CostMoney$) for Nodes $ServF$, $ServG$ and $ServH$. If the values for $ServF$ and $ServG$ are quite similar but both smaller than those for $ServH$, we give up node $ServH$, and we proceed with nodes $ServF$ and $ServG$.
Second, the pre-coordinator calculates the optimal solution service to identify the best candidate from the set from the last step. Let us compute the QoS for F on the basis of C: $QoSCF = CalCostMoneyCF + CalCostTimeCF$. $CalCostTimeCF$ and $CalCostMoneyCF$ are weighted calculations using the watchdog data on service F in relation to C. These can employ complex user requirements were users have to define weights for all factors. Similar calculations are performed for $ServG$ and service with the minimal score is chosen as candidate. Let us assume it is $ServF$.
Last, we determine the feasibility of $ServF$. If $RestrictF = true$ or $CostMoney = CostMoneyAC + CostMoneyCF > sum$ or $WaitTimeF > WantTime$, that is if the limiting conditions of F are true, or the account balance is insufficient or the time we have to wait in F is larger than desired, then the node F cannot be chosen and

the Pre-coordinator has too choose the second best node.
If all nodes can be successfully instantiated then the process will be executed; if no instantiation can be found feedback is provided to the user and the composed service is not executed (execution would lead to failure and thus a need to backward recover and hence undesirable costs).
To demonstrate the proof of concept, we have implemented the prototype using Java. The snapshot of our prototype system is shown in Figure 2. The experimental data was derived from travel booking data and guest room price data obtained according to the scenario about travelling and accommodations from Beijing to Qingdao[1]. We only used a subset of the available data for the demo.

IV. Related Work

Research such as [3] has shown that there are crucial distinctions between Web Service transactions and traditional transactions, hence new ideas were introduced, crucially compensation. These ideas manifest themselves in forward recovery e.g. [4] and backward recovery e.g. [5] techniques. Realising that not every step can be returned to its original situation and that for some steps this is not even desirable from a business perspective [6] allows to specify what steps are part of a transaction.
Work in web service transactions is usually showing the need of effort to 'repair' transactions when exceptions occur and this effort is costly in terms of resources. The work presented here attempts to predict success of the diverse steps in a process and selects services in order to reduce the time overhead and resource waste by trying to ensure that more steps succeed in the first instance.
Prediction of execution results has been considered in [7] and [8] who consider predicting the quality of a service that can be achieved based on past execution history and criteria such as availability and reliability of services. They do not consider transactional concepts.
Much work has been conducted in the area of service selection. Global optimizations approaches (e.g. [9]) achieve solutions that consider the interplay of service QoS values, but are less fault tolerant and flexible as the whole instantiation is preplanned. Local selection methods (e.g. [10], [11]) consider criteria effecting the selection of an individual service and essentially rank services. They are computationally quite efficient and can be applied 'just in time' increasing the chance of a relevant service being selecting.
In this paper, the primary selection criteria is the anticipated success of the transaction and the secondary criteria consider ensuring good quality of service, so in that sense it is not trying to make a specific contribution in the area of service selection.

V. Conclusion and Future work

In this paper, we propose a framework to improve and optimize the success rate of transactional composite services by predicting how likely a service will lead to desired results before including them as a candidate service in the process. We used a case study based on realistic data and the experimental result demonstrated that the approach improves success rate of composite services completing successfully and thus reducing the need to employ forward or backward recovery approaches in many cases (they will still be needed if services fail for other reasons).
In the future, we will investigate performance and scalability of our approach. We will also investigate different approaches to watchdogs: at the moment we are monitoring each concrete service through its own watchdog; an alternative might be to use watchdogs at the level of abstract services which will decentralize the decision making (which at the moment is conducted in the Pre-coordinator) and thus could bring efficiency gains as well as allowing easy implementations of load balancing.

[1]This data is partially from http://www.12306.cn.

Acknowledgement

The paper is fully supported by a grant from the Fundamental Research Funds for the Central Universities (Project No. 13CX06009A and No. 14CX06007A). This work is a partial result of Jiuyun's visit to the University of Leicester supported by China Scholarship Council.

References

[1] Y. J. Wei Xiao-Juan, "Skyline query processing," *The Chinese Journal of Software*, vol. 19, no. 6, pp. 1386–1400, 2008.
[2] Y. Gan, M. Chechik, S. Nejati, J. Bennett, B. O'Farrell, and J. Waterhouse, "Runtime monitoring of web service conversations," in *Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '07. Riverton, NJ, USA: IBM Corp., 2007, pp. 42–57.
[3] G. He-Qing, "A survey on web service transaction," *The Chinese Journal of Computer Science*, vol. 32, no. 5, pp. 13–16, 2008.
[4] P. Dolog, M. Schafer, and W. Nejdl, "Design and management of web service transactions with forward recovery," in *Advanced Web Services*, A. Bouguettaya, Q. Z. Sheng, and F. Daniel, Eds. Springer New York, 2014, pp. 3–27.
[5] J. El Hadad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and qos-aware selection algorithm for automatic web service composition," *Services Computing, IEEE Transactions on*, vol. 3, no. 1, pp. 73–85, Jan 2010.
[6] M. S. Ali and S. Reiff-Marganiec, "Autonomous failure-handling mechanism for WF long running transactions," in *2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, June 24-29, 2012*, 2012, pp. 562–569.
[7] V. Grassi, "Architecture-based reliability prediction for service-oriented computing," in *Architecting Dependable Systems III*, R. Lemos, C. Gacek, and A. Romanovsky, Eds. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 279–299.
[8] W. Fang-Chun and Y. Jiang-Xia, "Qos prediction of web service composition with transaction mechanism," *Journal of Electronics & Information Technology*, vol. 3, p. 047, 2008.
[9] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '05. New York, NY, USA: ACM, 2005, pp. 1069–1075.
[10] F. Casati, S. Ilnicki, L.-j. Jin, V. Krishnamoorthy, and M.-C. Shan, "Adaptive and dynamic service composition in eflow," in *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, ser. CAiSE '00. London, UK, UK: Springer-Verlag, 2000, pp. 13–31.
[11] H. Q. Yu and S. Reiff-Marganiec, "A method for automated web service selection," in *Proceedings of the 2008 IEEE Congress on Services - Part I*, ser. SERVICES '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 513–520.