A Cost-aware Scheduling for Reservation-Based Long Running Transactions

Lin Qing, Pham Phuoc Hung, Jeong Yong Byun
e-mail : qinglin9@gmail.com, hung205a2@gmail.com, byunjy@dongguk.ac.kr
Dept. of Computer Engineering, Dongguk University

예약기반 장기수행 변동처리를위한 비용인지 시간계획

림청, 팜폭헝, 변정용
동국대학교 전자계산학

Abstract

Web Service technologies make the automation of business activities that are distributed across multiple enterprises possible. Existing extended transaction protocols typically resort to compensation actions to regain atomicity and consistency. A reservation-based transaction protocol is proposed to reduce high compensation risk. However, for a serial long running transaction processing, the resource that is reserved in the early stage may be released due to resource holding time expires. Therefore, our analysis theoretically illustrates a scheduling scheme that tries to prevent the loss of resource holding as well as gain an optimized execution plan with minimum compensation cost. In order to estimate cost of different schedules, we set up a costing model and cost metric to quantize compensation risk.

1. Introduction

The automation of business activities that are distributed across multiple enterprises becomes possible with the advent of the new generation of Internet-based technology, in particular, Web Services. Business activities typically involve related tasks that are loosely coupled and carried out over a long period of time. The automation of business activities, with direct computer-to-computer interactions and without human involvement, can provide substantial speed improvements and cost reductions for distributed enterprise computing. [1]

However, it is not pragmatic to apply conservative two phase locking protocol in distributed organizations. Instead, each Web Service is allowed to commit independently. If the whole transaction fails for some reasons, compensation [2] operations are invoked for the completed Web Services to eliminate the effects that have been done. In case of this situation, WS-BusinessActivity[3][4], one of WS-Transaction components, is launched to handle long-duration, ACID-relaxed transactions among loosely-coupled systems and asynchronous communication. However, it is difficult and expensive to resolve such inconsistencies, particularly when the databases are spread across multiple enterprises, as is the intention of Web Services. Some researchers have proposed a reservation-based transaction protocol [4] to coordinate business activities. In the reservation-based protocol, an application has full control over the reservation activity, as well as over how long the resource should be reserved.

However, those transaction protocols seldom consider serial long running transaction processing. Due to complexity of business activities, workflow is required to specify execution order of distributed services. For example, with an order processed in advance, can the vendor knows how many products to be reserved for client, and then payment, logistics could be arranged. Problem comes when logistics arrangement takes a long time that vendor's reservation period expires, transactions are prone to be in an inconsistence situation again, although a reservation-based protocol is applied, for the reason that every enterprise participating this activity is self-centered and selfish. A direct solution for this problem is to complete immediately as long as reservation time is out. But it is inadvisable when compensation cost is likely to be generated. The criteria of scheduling include reservation time constraints, a date flow graph that represents resources consuming sequence, and a quantitative compensation cost estimation. Our experiments approve that execution of transactions following the scheduling, enhances the performance of reservation-based transaction management.

The rest of paper is organized as following: in the 2nd chapter we will discuss related studies. Then we will focus on motivating scenario and discuss proposed method in the 3th chapter. System model, algorithm will be illustrated in the 4th and 5th chapter. Paper will be concluded with conclusion.

2. Related Work

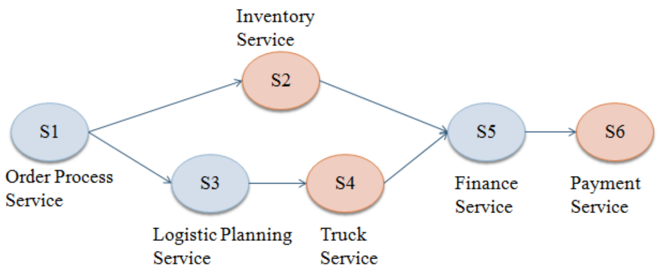
[5][6] Compared with paper [5], in paper [6] the reservation-based protocol but with an imposed time constraint is proposed, which helps eliminate those side effects. Also, concurrent conflicts detection is developed as well to enhance performance. Moreover, the simulation integrates this resource-centric approach into Service-Oriented Architecture (SOA).
[7] In this paper, they try to explore issues with emphasis on compensation-cost analysis based on two-phase locking commit pattern. Their main contribution is providing a

formal for pricing the compensation cost of aborted transactions and metrics and a framework for probabilistic analysis. And in our paper, first we inherit this mathematical method, and apply it to more complicated environment, which is specifically designed for processing order predetermined long running transactions. Second, we argue the price-based compensation model and extend concept consideration for compensation.

[8] This paper devises the algorithm for scheduling, which maintains atomicity with minimum compensation cost and satisfies temporal constraints as well. In contrast, we concentrate on reservation deadline constraint for the sake of reservation-based protocol, which is already proved to show better performance than pure WS-BusinessActivity protocol. Moreover, our purpose is to solve conflict between resource holding time and minimum compensation cost.

3. Motivating Scenario

We begin by using a simple example to show the motivation scenario. Fig.1 illustrates a data flow graph (DFG) of a distributed purchase and supply planning system, which needs to consume services and gain resources from multiple enterprises. In the graph, DFG is an essential element within the high-level synthesis flow and represents a chain of operations with data dependencies. For example, in our scenario, Order Processing Service is performed earlier than others. According to its results, Inventory Service updates quantity in the database and Logistics Planning Service maps out a suitable delivery plan through the knowledge of goods type and destination location. Subsequently truck service is called. And then Finance Service is to calculate the total fee that customer should pay for and payment request is sent to Bank Service immediately. This complex activity needs transactional support to guarantee its atomicity. Specifically, for products, payment and truck resource acquisition (red color in Fig.1), either all of them complete successfully or none of them do. Under the circumstance of reservation-based protocol, only when three positive reservation responses arrive will the coordinator issue “complete” command to all the providers.



(Fig.1) DFG of a distributed purchase and supply planning system

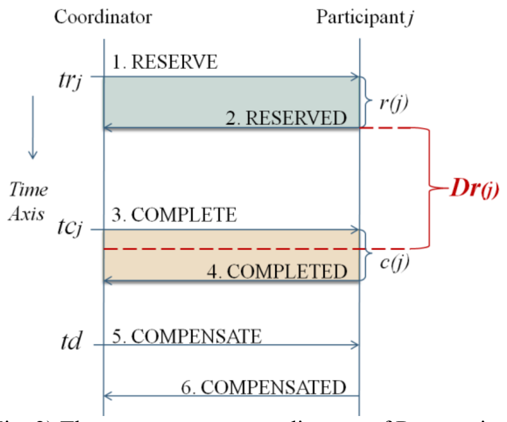
What’s more, it is necessary for every service provider imposes a time constraint for reservation according to paper [6]. But this time constraint brings problem. Take our purchase and supply system for example, items that have already been reserved from inventory may be released prior to the overall transaction reservation phase is able to end. Suppose Logistics Planning Service and Truck Service take longer time than usual so that Truck Service can not finish reservation but Inventory Service’s period for resource holding expires. As a sequence, transaction is forced into an inconsistent position again. This disadvantage is inevitable if we want to popularize reservation-based protocol for transaction management. Certainly, a direct solution is to complete as soon as time is out. But what if supplier requires penalty for cancelling a completed purchase, meanwhile succeeding tasks are actually prone to fail? Eventually, customer probably needs to pay extra money for a failed purchase. Let us suppose that the expected delivery day happens to be on busy season, e.g. Charismas Eve, and failure rate is relatively high. Under this circumstance, delaying item reservation from Inventory Service until Truck Service is successfully reserved could be a better choice.

From this example, we can see that for the serial long running transactions, simple reservation may force transaction to inconsistency or considerable compensation cost. Therefore, scheduling to seek the right time of distributed services’ reservation and completion is our main purpose in this paper.

4. System Model

In our reservation protocol, except for read or computation only tasks, each task within a business activity is executed as two subtasks. The first subtask involves an explicit reservation of resources according to the business logic. The second subtask involves the confirmation or a cancellation of the reservation. Compared with other transaction models, our novelty is that execution dependency only imposes serial restrictions on services’ reservations, whereas completions, which are also seen as commitments from distributed databases, can be rescheduled to pursue an optimized execution plan. In addition, some services. Next is the genetic modeling for the background and the problem we are going to solve.

4.1 Conversation Model of Reservation-based Protocol



(Fig. 2) The message sequence diagram of Reservation-based Protocol

In this model, message transfer delay is neglected to simplify computation.

1. For $j=1$ to n, at the time tr_j Coordinator invokes the Participant j (Service j) with a RESERVE message, which also demands that the reply from Participant j shall arrive at Coordinator before time $tr_j+r(j)$, and $r(j)$ is the relative timeout by which Coordinator waits for the reply.

2. On receipt of the RESERVE request, participant *j* should respond with a RESERVED after a successful computation and reservation of request. Otherwise, it answers FAIL.
3. Over some period of time, a COMPLETE command is issued to Participant *j* at the time of tc_j. And reply from Participant *j* should arrive after the period of *c(j)*. However, we should mention that starting to convert participant *j* 'state from reservation to completion should be in the condition that resource is still in reservation, which means $tc_j \leq tr_j + r(j) + Dr(j)$. Here, *Dr(j)* is the time constraint for reservation that is declared by Participant *j*.
4. In case of failure from other participants, a COMPENSATE message is issued to participant *j*.

4.2 Compensation Mechanism

In SOA, the compensating operation is usually another service (e.g. a debiting service) that cancels the effect of the already completed service (e.g. a crediting service). It is worth noting that the compensation logic cannot be automatically generated and therefore relies upon application-specific business logic.

In some cases, an operation cannot be easily compensated for, such as manufacturing an item, and compensating operation can charge the customer a cancellation fee and offer to sell the item to other parties. [2] However, many researches contribute to the compensation development in SOA, few of them study about costs resulted from cancellation. This cost could be real and precise price that is quoted by vendors, or it could represent compensation complexity for individual services. In this paper, we will not explain how to price compensation cost, for the reason that it must be discussed in a living example. But we do provide a cost model based on the common knowledge about compensation penalty, by which can we develop our cost-aware optimization scheduling.

To make our work more general, we assume that the compensation operation of a service can be invoked even if the service is still being executed. Compensation cost model is given as follows:

$$Cost\,(S_j, td, tc) = \begin{cases} 0 & td < tcj \\ \alpha_j\,(td - tcj) & tcj \leq td \leq tcj + c(j) \\ \beta_j\,(td - tcj - c(j)) & tcj + c(j) < td \end{cases}$$

Here, td is the time Coordinator decides to compensate; tc_j is the time that sending Participant *j* COMPLETE request, and *c(j)* is the period between request and response from Participant *j*. α_j and β_j are non-negative penalty coefficients for Participant *j*'s completing and completed state respectively. The reason we use here two penalty coefficients is that the cost to undo the effects after a service has completed is generally greater than that when the service is still being executed.

4.3 Transactional QoS Definition

In this paper, we are interested in properly scheduling reservation and completion of component services in terms of QoS criteria and constraints to obtain a minimized compensation cost. We consider four generic quality criteria for elementary services.

Execution Duration. Execution duration measures the expected delay in seconds or minutes even hours between the moment when a request is sent and the moment when the result are received. In our paper, both reservation duration *r(j)* and completion duration *c(j)* are required for service *j* in the environment of Reservation-based protocol.

Success Execution rate. It is the probability that Participant j responds correctly to the Coordinator. In our paper, both reservation success rate $q_r(j)$ and completion success rate $q_c(j)$ are required for service j because of Reservation-based protocol. $q_r(j)$ denotes the resource acquire rate and $q_c(j)$ is the probability that the service functions correctly and consistently provide the same service quality. That's reliability of a service.

Execution and compensation price. From texts above, we already define a cost function for service *j*. It is a linear equation with two Coefficients. α_j is the coefficient for modeling execution cost, and β_j is used for compensation cost after service *j* successfully completes.

Reservation Time Constraint. Participant *j* imposes time constraint *Dr(j)* for resource reservation, which is one of the main factors we will consider for our scheduling.

4.4 Compensation Cost Mathematical Evaluation

The cost is produced when a failure occurs. So basically we measure cost for each schedule by accumulating potential cost from every individual moment during execution. Thus firstly, calculate cost at time *t*. $P_c(t)$ is the success probability before time *t*, and $P_f(t)$ is the failure probability at time *t*. $cost(S_i)$ is the cost for a successful service *i* to compensate, and *n* is the number of completed services. Then cost at time *t* is calculated according to compensation model defined in 4.2.

$$C\,(t) = P_c(t) \times P_f(t) \times \sum_{i=1}^{n} cost(S_i) \qquad (1)$$

Final total cost is the integral of *C (t)* from starting point to the termination of transaction with respect to *t*. This is represented in the formula below, and *Ts*, *Tf* denote start point and termination point of transaction.

$$\int_{Ts}^{Tf} C(t)dt \qquad (2)$$

4.5 Simple Example

In this part, QoS parameters that are discussed above are assigned to the services in motivating scenario, which are listed in Table 1, and through this vivid example, it is easier to understand how effectively and necessarily cost-aware scheduling contributes reducing compensation cost.

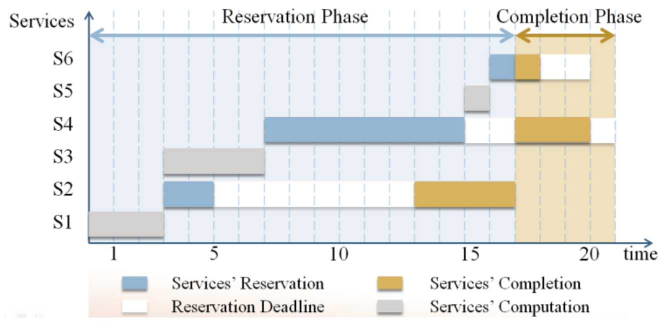
<Table 1> QoS Example for Motivating Scenario

Service Name	Order *j*	*qr(j)*	*r(j)*	*Dr(j)*	*qc(j)*	*c(j)*	*α / β*
Inventory	S2	99%	2	8	99.8%	4	0 / 0.1
Truck	S4	80%	8	8	99%	3	0/ 0.01
Payment	S6	99%	1	2	99.9%	1	0.8 / 0.9

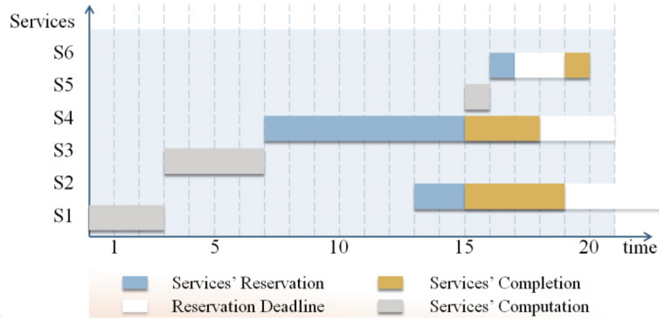
We should notice only database associated services are scheduled, for the reason that only resource consuming operations require cost for compensation. So Table 1 does not include other computational services. Via intuitive analysis of Table 1, we can see Truck service has lower success rate and lower cost coefficients compared with Inventory service. It is reasonable because truck is a kind of scarce resource which is different from products that can be increased effortless if market demands expand suddenly. Moreover, Payment service has highest success rate while most expensive compensation cost for the sake of bank system's reliability and complexity.

Let's see a natural schedule in Fig.3. The overall transaction experiences two fixed phases that we can see from Fig.3. The blue bar denotes reservation execution, and the orange one denotes completion execution. The white background represents deadline for reservation.

In case of aforementioned reservation time constraint problem, we assume transaction coordinator completes Inventory Service in advance, at time 13, with an aim to prevent the resource release. Fig.4 shows a cost aware optimized example. Apparently a rescheduled transaction gets rid of overall Reservation Phase and Completion Phase, which means executions of each individual service are pipelined to pursue cost optimization. Looking inside of the second schedule, Inventory Service is postponed behind Truck Service, which is prone to fail. Meanwhile, Bank Payment Service is scheduled at the rear of transaction since it is a high compensation cost service. Besides, mathematical evaluation of these two services is conducted as well. The result undoubtedly demonstrates the second schedule helps save cost more than 90% compared with nature scheduling according to formula that is defined above.



(Fig. 3) Natural Schedule Example



(Fig. 4) A Cost-aware Optimized Example

5. Scheduling Algorithm

Backtracking search [9] is our algorithm to solve this problem, which is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. The basic idea is that, in the first round, assign reservation start time for one service at a time. Second round is the assignment of completion start time for each service, until all the services know when they should start to reserve and complete.

And then, a schedule is found, the cost function is applied to calculate the expected compensation cost and this value will be recorded. After all the cases are searched and tried, eventually the application is able to get an optimized schedule via comparison among those estimated results.

6. Conclusion and Future Work

The main content of this paper is devoted to studying compensation cost optimization by proper scheduling at the same time without violating reservation time constraint when reservation-based protocol is applied. A quantitative model of cost evaluation and a backtracking algorithm is raised in this paper to solve this problem.

However, plain backtracking is not effective for large problem. For example, we increase service number in one transaction, or cut time by a smaller time unit. This will definitely enlarge the scale of problem. So domain-specific heuristic functions derived from our knowledge of the problem is able to help reduce search space greatly. This is our aim for future work.

Reference

[1]Sandeep Chatterjee, James Webber, Developing Enterprise Web Services: An Architect's Guide. Prentice Hall PTR, 2003, pp. 272.

[2] Biswas, Debmalya, "Compensation in the world of Web Service Composition", Semantic Web Services and Web Process Composition Volume 3387, Springer Berlin Heidelberg, 2005.

[3]OASIS Web Service Business Activity (WS-BusinessActivity), found at: http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec.pdf

[4]OASIS Web Service Coordination (WS-Coordination) Version1.2, http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os.pdf, 2009.

[5]Wenbing Zhao, Louise E. Moser P.M.Melliar-Smith, "A Reservation-Based Extended Transaction Protocol", IEEE Transactions on Parallel and Distributed Systems, 2008.

[6]LinQing, Aziz Nasridinov, Pham Phuoc Hung , Jeong Yong Byun , "An Improved Reservation-based Protocol for Timely Web Service Transactions", Journal of KIISE: Computing Practices and Letters, Volumn 17, 2011.

[7] Haitao Yang, Zhenghua Wang, Qinghua Deng, Scheduling optimization in coupling independent services as a Grid transaction, Journal of Parallel and Distributed Computing, Volume 68, Issue 6, June 2008

[8] Liu, An, Liu, Hai, Li, Qing, Huang, Liu-Sheng, Xiao Ming-Jun, "Constraints-Aware Scheduling for Transactional Services Composition", Journal of Computer Science and Technology, Volume 24, Issue 4, Springer Boston, 2009.

[9]Stuart Jonathan Russell, Peter Norvig, "Artificial intelligence: a modern approach", Prentice Hall, 2010, pp.137-155.