Isolation and Web Services Transactions

David Paul, Frans Henskens, Michael Hannaford
School of Electrical Engineering and Computer Science, University of Newcastle, Australia
David.Paul@newcastle.edu.au, Frans.Henskens@newcastle.edu.au,
Michael.Hannaford@newcastle.edu.au

Abstract

The traditional ACID properties for transactions are typically ignored in the Web Services environment to ensure an acceptable level of service. However, this common lack of isolation can cause difficulties. We look at ways to reduce these problems but still maintain an acceptable level of service.

1. Introduction

Web Services transactions combine multiple services, possibly located on heterogeneous systems, into a single logical unit. They can thus be thought of as multidatabase transactions[1]. Typically, the traditional ACID (atomicity, consistency, isolation, and durability) properties of database transactions are relaxed in such systems, as the necessary locks and restrictions would severely reduce the usefulness of the system. Atomicity is often replaced with semantic atomicity[2], which simply requires that any transaction either succeeds fully or converts the system to a state such that the transaction may not have run (typically achieved through compensating transactions that logically undo any actions a failed transaction has already completed). Isolation, on the other hand, is typically ignored.

Isolation ensures that transactions do not get an inconsistent view of data because of concurrently running transactions. In the strictest sense, isolation is only guaranteed if concurrent transactions are serializable, meaning that there is some order in which they could be run serially that would achieve the same result as running them concurrently. Often, isolation is achieved by ensuring that transactions cannot access data being written by other transactions. This is typically realized by placing locks that stop other transactions from having access to particular data until the original transaction has finished with it. While this works well for single systems, for Web Services transactions only calls to the individual services are isolated. Thus, transactions that use multiple services may not be serializable. Further, using locks would not be acceptable in this environment, as Web Services transactions can run for long periods of time (perhaps taking weeks or months to complete), and blocking access to services until a transaction had finished would result in an unacceptable degradation of service.

However, there are some problems that arise because global isolation is not enforced. When a transaction has finished working with one resource, any later transactions can see the changes made. This is fine if the original transaction completes successfully, but if it later fails, and then undoes the changes made on that resource, the other transactions that used that service potentially have an inconsistent view of the state of that resource. Thus, transactions that should have been able to succeed may fail, or a transaction may unnecessarily follow a different path because of the inconsistent view of the data.

We look at ways that these problems can be overcome, or at least minimized, allowing an acceptable level of service while improving the handling of isolation of Web Services transactions.

2. Motivating Examples

Consider a situation where two people, A and B, wish to travel to a small town. The town is serviced by only one airline, and has only one hotel. Thus, both A and B need a flight from the airline and a room in the hotel. A also requires a rental car, though B does not. If A books the last seat on the flight, then, even if B can successfully book accommodation, B’s transaction would fail. If A is then unable to hire a car, A would cancel the booking, so B should be able to book instead. However, unless B explicitly resends the request, B would not be aware that the booking would now succeed.

A similar situation may occur if the town was serviced by two airlines, X and Y. If B prefers to travel

0-7695-3049-4/07 $25.00 © 2007 IEEE
DOI 10.1109/.29

with X, but is willing to travel with Y if necessary, and A gets the last seat on the flight from X, then B must book with Y. If A later cancels, then it should be possible for B to cancel the flight with Y and instead book with the preferred airline X. However, B's transaction has already succeeded, booking a flight with Y. Thus, B would be forced to travel with Y unnecessarily, even though a seat on X would be available.

3. Potential Solutions

To stop the first situation, where B cannot book a flight, it would be possible for B to resend the request at a later time. In fact, B could keep a hold on the initial successful request for accommodation and only resend the flight request. However, B must decide how long to wait between requests. If too short a time is chosen, then the repeated requests could put too much strain on the service provider. If, on the other hand, B does not request often enough, a third person may book the flight after A has cancelled their booking, but before B has resent their request.

Another approach is to use optimistic concurrency control to enforce isolation[3, 4]. While such work shows promise, transactions are still required to wait for other transactions to complete before they can commit themselves. This can cause unacceptable delay when transactions may run for very long periods of time.

The final approach considered here is to have all requests include a time limit, and, in the event of a failure, have the service provider add the request to a queue. Then, if the situation changes, the service provider could notify the client that their request would now succeed, and ask if the action should go ahead. Thus, B could request a flight with a time limit of, say, until one week before they want to travel. B would still have their request fail when first requesting the flight, but when A cancelled, B would be notified and could then accept and have the flight booked for them. If B had since changed their mind, however, they could reject the offer and the flight could then be offered to other clients. Similarly, in the second situation, B's request for a flight with airline X could also be queued, so that if A cancelled, B could cancel the flight with airline Y and instead accept the flight with airline X.

While this final approach does solve the problems from the motivating example, it does have problems of its own. The most obvious of these is that the client requesting the service must be available, so that the service provider can later contact them, after the transaction has finished. The transaction coordinator could perhaps be this contact point, but then the coordinator would need a way to determine whether to accept the offer or not, and how to proceed from that point.

Another problem with this method is that B may be holding on to the initial successful accommodation request simply hoping that they will later be able to get a flight. In this way, other clients who want accommodation may have their requests fail, only to have B later cancel if no flight became available. The service provider can, however, determine how long a client can obtain a hold, and any cancellation fees that apply. Further, if new clients use the same system and send a time limit with their request then, when B cancels, they will be notified and successfully make their booking.

Having a large number of clients waiting for these notifications and then changing their plans when the notifications arrive would result in more work being wasted than if clients had simply waited to resend their request. It would be a decision for individual clients, however, as to whether extra work should begin on receipt of a notification. And again, service providers can set limits as to how long a client has to cancel plans, and any compensation that the service provider would require for such cancellation.

4. Conclusion

While strict isolation in Web Services transactions would result in unacceptable levels of service, the lack of isolation in current protocols can cause problems. One way to avoid some of these problems is to allow Web Services to "call back" clients whose requests have failed but would now succeed. This method has problems of its own, however, some of which still need to be overcome.

5. References

[1] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres, "Ensuring relaxed atomicity for flexible transactions in multidatabase systems," in *ACM SIGMOD international conference on Management of data*, 1994, pp. 67-78.

[2] H. Garcia-Molina, "Using semantic knowledge for transaction processing in a distributed database," *ACM Trans. Database Syst.,* vol. 8, pp. 186-213, 1983.

[3] M. Alrifai, P. Dolog, and W. Nejdl, "Transactions Concurrency Control in Web Service Environment," in *European Conference on Web Services*, 2006, pp. 109-118.

[4] S. Choi, H. Jang, H. Kim, J. Kim, S. M. Kim, J. Song, and Y. Lee, "Maintaining consistency under isolation relaxation of web services transactions," in *Web information systems engineering*, 2005.