The Need for Distributed Asynchronous Transactions

Lyman Do, Prabhu Ram, and Pamela Drew
Boeing Phantom Works, Mathematics and Computing Technologies
The Boeing Company
P.O. Box 3707, M/S 7L-70, Seattle, WA 98124-2207, USA
{lyman.s.do, prabhu.ram, pamela.a.drew}@boeing.com

ABSTRACT

The theme of the paper is to promote research on asynchronous transactions. We discuss our experience of executing synchronous transactions on a large distributed production system in The Boeing Company. Due to the poor performance of synchronous transactions in our environment, it motivated the exploration of asynchronous transactions as an alternate solution. This paper presents the requirements and benefits/limitations of asynchronous transactions. Open issues related to large scale deployments of asynchronous transactions are also discussed.

1. INTRODUCTION

Boeing has a system that maintains configuration control of airplanes which maintains information such as which parts are used in a particular instance of an airplane. This system supports 60,000 users and has a service level agreement (SLA) of 2 minutes. The SLA is a contract between the system users and implementers, and, defines the maximum transaction response time. The system is composed of eight distributed databases, six of which are on a LAN and the another two are on WAN. Part of the data is replicated and global transactions are implemented to guarantee data consistency. Two phase commit (2PC) [1] protocol has been used to guarantee global atomicity.

We have conducted experiments on our production environment to evaluate the performance and scalability of the system. The result shows that the SLA cannot be met when a global transaction accesses two or more databases through the WAN. For example, the transaction that accesses five databases, three of which are on the LAN and the others are on the WAN, takes nine minutes to complete. Further investigations [7] show that the 2PC component of the transaction response time is constant and insignificant but the blocking nature of the synchronous communication that implements 2PC contributes most to the poor performance.

Extended transaction models (ETMs) have been researched for well over a decade and generally attempt to exploit application semantics to increase concurrency and reduce synchronization latency. Several well known ETMs are summarized in [4]. Most of the efforts have focused on isolation relaxation to allow more concurrency in multidatabases and other environments, to support for long lived transactions, and to support for non-traditional database applications (such as CAD/CAM and collaborative work). It should be noted that none of these ETM implementations are commercially available (with the exception of nested transactions [5] in some transaction processing monitors). It is also interesting to note that overall system performance issues are rarely addressed in ETMs. Beyond functionality, system performance is the key measure of success in industrial deployments. From the commercial side, some DBMS, middleware, and collaborative tool vendors provide tools such as persistent queues, message oriented middleware, etc., but the traditional transaction issues of providing concurrency control between transactions and transaction atomicity are left to the application to manage.

Given these reasons, we have been investigating asynchronous transaction technology to improve transaction response time without compromising data consistency. We describe the requirements, benefits, and limitations of using asynchronous transactions in Section 2. In Section 3, we raise open issues that deserve additional research attention in order to make asynchronous transaction a practical alternative.

2. Asynchronous Transactions

An asynchronous transaction is a distributed composite transaction that is composed of sub-transactions. Similar to traditional synchronous transactions, these sub-transactions will reach the same termination decision. However, the termination of these sub-transactions is asynchronous, i.e., some of the sub-transactions may have been committed while others may be still executing or have not yet executed. One significant characteristic of asynchronous transactions is they are guaranteed to be propagated to the target systems *once and only once*. Once an update at the source is committed, it will commit and expect that the update will be propagated to the target appropriately. The "once" part guarantees this expectation, and the "only once" part guarantees that update propagation will not be duplicated. It should be noted that asynchronous transactions are not only applicable in traditional database application domains. It is an infrastructure component whose presence can benefit asynchronous information sharing environments in the collaborative work area and asymmetric replication environments such as data warehouses [2,6].

2.1 Requirements

The candidate distributed environments that are suitable to deploy asynchronous transactions must have two important characteristics:

1. *Temporarily out-of-sync information*: Using asynchronous transactions, there will be a time gap between the update at the source and the update at the target. Hence, the source and target databases are temporarily out-of-sync until the entire asynchronous transaction is completed. The system (application designer) has to be aware of this characteristic and be able to tolerate the temporarily out-of-sync information.

2. *Unidirectional update propagation*: Asynchronous transactions are particularly suitable to environments that require unidirectional update propagation, i.e., the update will be propagated from a source to one or more targets, but not vice versa. If the computing environment implements bi-directional concurrent update propagation, the asynchronous nature of transaction may result in an incompatible execution sequence of transactions in the global perspective and may require manual reconciliation. The transaction management technology must be capable of detecting and flagging such inconsistencies to assist in the reconciliation process.

2.2 Benefits and Implications

We refer readers to [3] for a detailed discussion on the benefits and implications of deploying asynchronous transactions. Briefly here are the benefits of deploying asynchronous transactions:

1. Asynchronous transactions improve response time and system throughput by eliminating synchronous interactions between its components. After submitting a request, the client is free to remain active or it can terminate successfully, without waiting for a collective consensus. The system throughput is also improved since extended resource locking is reduced to a minimum.

2. As asynchronous transactions guarantee once-and-only-once execution semantics, a failed sub-transaction can be re-tried once the problem is fixed, thereby facilitating forward recovery.

3. Since asynchronous transactions allow participating transactions to commit unilaterally, it eliminates the lock-and-wait scenario, i.e., the global deadlock, prevalent in strict synchronous transactions.

Beyond the two important environment characteristics discussed in Section 2.1, additional implications of deploying asynchronous transactions technology include:

1. Additional resources have to be committed to implement asynchronous transactions in order to guarantee the once-and-only-once execution semantic. A typical example of such a resource is a transactional queue which incurs additional I/O overhead.

2. Since sub-transactions of an asynchronous transaction can be committed unilaterally, some scenarios may require rolling back of a committed sub-transaction. The application may need to support compensation or equivalent logic in order to rollback unilaterally committed sub-transactions.

3. Open Issues and Conclusions

While synchronous transactions clearly have their place, we have discussed the need for asynchronous transactions to support distributed applications. In this section, we discuss several issues that need to be addressed if asynchronous transactions are to be widely deployed and used. In general, there needs to be a better theoretical foundation developed for asynchronous transactions (just as it has been done for synchronous technology) and has to be grounded by implementations that deliver on overall system performance.

1. There needs to be ways of managing the temporal inconsistency as discussed in Section 2.1. The methods used to manage asynchronous transactions must allow control and quantification of propagation delays so that applications can function with some temporal guarantees. Fundamental research into this area has been done in efforts such as Epsilon serializability [8]. However, ETMs need to address additional implementation and performance issues.

2. The environment we discussed in Section 1 is a sub-system that is part of an integrated system composed of other commercial off-the-shelf (COTS) software. In fact, a typical transaction indeed spans several COTS software, each of which has their own encapsulated databases. Integrated systems such as these need asynchronous transactions so that one application in a COTS software does not get "tied down" by synchronous interactions with another application in a different COTS software. If asynchronous transactions are to be used in such integrated environments, the following issues need to be addressed:

- It is very likely that in such systems, information will flow from multiple COTS sources to multiple COTS targets concurrently. Additionally, data in these COTS sources will have inter-relationships amongst them. Given these restrictions, how can redundant update propagations be consolidated before submitting them to the targets?
- Design issues such as how many asynchronous transactions mechanisms (persistent queues for example) are required to support large environments for load balancing and performance improvement reasons needs to be explored.

3. Most large scale systems will cause bi-directional conflicting updates. How are the concurrency and isolations issues handled in the presence of bi-directional conflicting updates propagated through multiple asynchronous transaction mechanisms?

4. How the forward recovery of failed update propagations are handled? Significant research effort has gone generally into failure handling in similar environments and we would like to see these research grounded with implementations. Issues including requirements on applications (for example, the interactions between applications and the asynchronous transaction mechanism may need to be idempotent), if and how the asynchronous transaction mechanism stores state changes relevant to the application, etc., need to be investigated further.

5. How the involvement of an additional component in the system, namely, the asynchronous transactions mechanism, improves and does not deteriorate overall system performance due to the mechanisms' own overheads?

To meet the needs of our own enterprise, we have been working on prototypes that extend vendor provided tools to address the issues discussed above.

4. References

[1] Bernstein, P., Hadzilacos, V., and Goodman, N., Concurrency Control and Recovery in Databases, Addison-Wesley, 1987.

[2] Do, L., *et.al.*, *Issues on Developing Very Large Data Warehouses*. In Proc. of the 24th VLDB, 1998.

[3] Do, L., and Ram, P., *State of the Art of Asynchronous Transaction Mgt.*, Boeing Tech. Report SSGTECH 98-016.

[4] Elmagarmid, A., Editor, Database Transaction Models for Advanced Applications, Morgan Kauffman, 1990.

[5] Moss, E., Nested Transactions: An Approach to Reliable Distributed Computing, MIT Press, 1985.

[6] Ram, P. and Do, L., *Delta Extraction for Incremental Data Warehouse Maintenance*. Boeing Phantom Works M&CT Tech. Report 99-003. Manuscript submitted for publication.

[7] Ram, P., Do, L., and Drew, P., Distributed Transactions in Practice. Submitted for publication.

[8] Ramamritham, K., and Pu, C., *A Formal Characterization of Epsilon Serializability*, IEEE TKDE, 7(6), 1995.