Management of Large Distributed Transactions

Extended Abstract submitted to HPTS Workshop 2001

Weihai Yu

Department of Computer Science
University of Tromsø, Norway
weihai@cs.uit.no

1 Introduction

Transaction processing includes mechanisms for concurrency control and recovery, as well as *transaction management* that maintains the necessary overall context information of transactions and conducts the concurrency control and recovery activities. However, transaction processing suffers from two scalability problems for large transactions.

- Concurrency control for transactions that access *large amount of data* and last for *long period*. This will cause large amount of data to be locked for long period and thus hinder concurrent access of them. Moreover, the more data each transaction accesses, the higher chances of deadlocks among transactions.
- Management of distributed transactions that involve *many server sites*. The most noticeable tasks for transaction management are transaction enlistment and commitment processing. These tasks may involve extra remote invocations and eventually also disk accesses, which are the main causes to poor performance.

Most applications therefore choose to adopt a less stringent approach such as TP-lite, low isolation levels or other limited scopes of distributed transactions.

In this paper we focus on scalability issues of distributed transaction management, i.e., transaction enlistment and commitment processing. There has not been much research on these issues though, mainly because it is widely believed that transactions should be kept small and should involve as few sites as possible. However we think it is now time to face these issues, given, for instance, that large-scale enterprise systems and application integration are expected to be one of the most important directions in IT industry, and that global computational grid for large-scale resource sharing is regarded as a promising research field. Furthermore, given the huge data space, transactions accessing many sites may not necessarily lead to high deadlock rates.

Note that the scalability issues discussed in this paper can be orthogonal to those due to concurrency control. As a matter of fact, compared to transactions involving human interactions (such as transactions for software engineering), those involving many sites may not necessarily last long (though many remote invocations *do* take longer time than a few local procedure calls). Furthermore, it is often the size of data *relative* to the overall data space that is the real problem for concurrency control.

We will discuss the current distributed transaction management approaches and propose an architecture that is particularly aimed at distributed transactions accessing many sites and combines the advantages of current approaches. We will show that this architecture does not require much reengineering efforts.

2 Requirements for Management of Large Distributed Transactions

2.1 Interoperability

In a large distributed environment, the many systems or sites involved are inevitably heterogeneous and autonomous with respect to various aspects including transaction management. Mechanisms must be provided to resolve heterogeneity and to some extend release autonomy. The overhead of this, however, should not increase significantly with the number of sites involved. Furthermore, the scope for administration and configuration should be kept as local as possible.

2.2 Security

When a distributed transaction proceeds, there are two kinds of interactions among sites happening at the same time:

- Remote method invocations, among application programs and/or resource managers,
- Management of transactions, among transaction management subsystems at each site.

Both these interactions have particular requirements on security. Regarding the latter, the coordinators and participants must be trusted principals. Otherwise a non-trusted coordinator may hold locked resources arbitrarily long or a non-trusted participant may repeatedly cause transactions to abort. Note that this kind of interactions is typically invisible to the application. Again, the security overhead should not grow significantly with the number of sites and much of the security checking efforts should be able to be reused beyond transactions' lifetimes.

2.3 Performance and optimizations

During the last three decades, enormous efforts have been made to enhance performance of transaction processing. Optimizations are critical aspects of TP products. When involved in larger environments, these somewhat localized optimizations should continue contributing to the overall performance of transactional applications.

2.4 Management of resources

One important resource in transactional applications is sessions between sites, such as remote database connections. A session may include, among other things, a network connection between the sites, options and attributes of various settings, application specific context or state like database cursor positions, etc. Sometimes sessions are also used for transaction management purposes. A session is typically not durable, but still managed within a transaction scope, such that associated resources are released upon transaction termination. Establishment of a session may involve for instance security checking as mentioned before and is thus expensive. Management mechanisms such as pooling can be used to avoid unnecessary overhead of session establishment.

2.5 Reengineering

Management of very large transactions is a new challenge. New mechanisms might be introduced or existing ones extended. One important requirement is that reengineering should be enforced under limited scope and should not affect considerably the existing mechanisms.

3 Current Transaction Management Alternatives

Basically, the sites involved in a transaction form a tree. The difference of the transaction management mechanisms lies in the shape of the trees.

3.1 Trees of transaction sites

This approach is used in almost all TP products. The tree of sites that a transaction accesses parallels the invocation patterns in the application. Usually, the root of the tree is the first site that the transaction accesses. There is an edge from a to b if b is first invoked from a on behalf of the transaction. Enlistment of a site to the transaction is done the first time that site is accessed by the transaction. Typically, the transaction manager (TM) of a site maintains information of TM of its parent site and the TMs of it immediate child sites.

Two-phase commitment is proceeded recursively from the root down to the leaves of the tree. Voting results are collected in the opposite direction.

There is usually a session maintained for neighbor sites (often in form of a database connection) that must be managed by the transaction. Remote invocations go through the sessions. Sessions can also be pooled and need not be released after termination of transactions.

Because transaction management shares the same tree structure as remote invocations, sessions can be used for both purposes. Interactions between the TMs may happen through the sessions with the help of communication managers at both sites. Or a direct connection between TMs is set up for transaction management purposes. Optimizations that piggyback TM messages and make use of synchronous nature of remote invocations like “implicit yes-vote” can be relatively easily enforced.

In a large distributed environment, it is generally not known in advance how invocations will occur. Therefore, heterogeneity and autonomy should be resolved at potentially every site pairs upon (first) invocations, so is security checking. This may imply reengineering everywhere.

When the tree gets deep, the daisy chain of communication for commitment processing becomes a severe source of latency. A useful optimization is flattening of the tree. After flattening, the coordinator site maintains an extra connection to all sites a transaction involves for commitment processing. Since potentially every site can be a coordinator or a participant, the number of extra connections can be considerably large. The dilemma here is, these connections, if not released, may not be reused often, while establishing one for every invocation is prohibitively expensive.

3.2 Flat collections of transaction sites

Unlike the tree-of-site approach in which a TM is enlisted at the TM of the immediate invoker, in this approach, every TM is enlisted at the single TM called the *superior coordinator*. OMG OTS, when no interposition is used to generate subordinate coordinators, is such an example. For every remote invocation, the global reference to the superior coordinator is included in the transaction context. Upon first invocation of the transaction on a site, the TM of the site enlists itself to the superior coordinator.

The superior coordinator can be either the TM where the transaction starts or configured as one of a few well-known sites. The former is similar to tree flattening mentioned before. In the latter case, interoperability and stringent security checking for transaction management purposes only happen between the superior coordinator and TMs at sites that transactions access, rather than between potentially every pair of sites. However, sessions for remote invocations are not used for

transaction management purposes. This will result in extra messages, since messages for transaction enlistment could otherwise be piggybacked in invocation messages.

4 The Proposed Architecture

We propose here an architecture in which a large distributed transaction environment consists of a number of transaction management domains (TM domains). A site belongs to either one TM domain or none. Distributed transaction management is supported within each TM domain using some domain-specific mechanism. Typically, a TP monitor or a distributed database system provides domain-specific transaction management.

Method invocations within a TM domain are manipulated by the transaction management facilities of that domain. This will typically form a tree of sites within that domain. Upon the first invocation across domain boundaries, an *external coordinator* will be selected. Selection of the external coordinator can be based on criteria like conformance to certain standard (such as OMG OTS or MS-DTC), location, availability, load balancing etc. Once an external coordinator is selected, the current coordinator within the TM domain will be registered to the new external coordinator and becomes a subordinate coordinator. In the new TM domain, the local coordinator will be registered to the external coordinator and plays the roll of a subordinate coordinator in that domain. Subsequent cross-domain invocations will be registered to the same external coordinator.

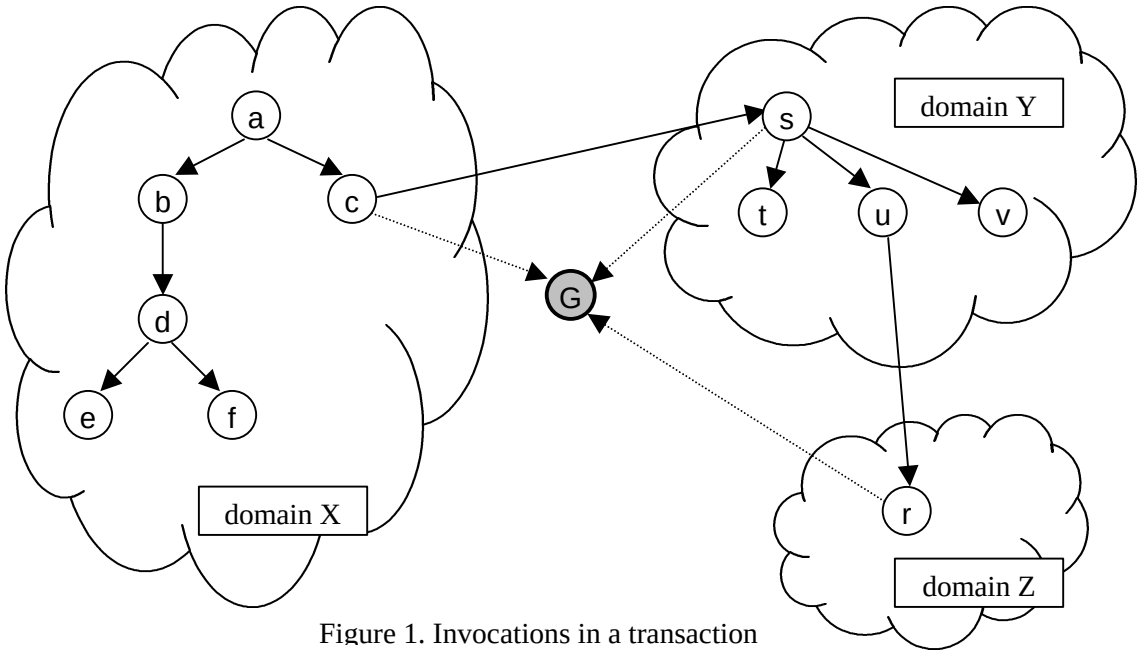

Figure 1. Invocations in a transaction

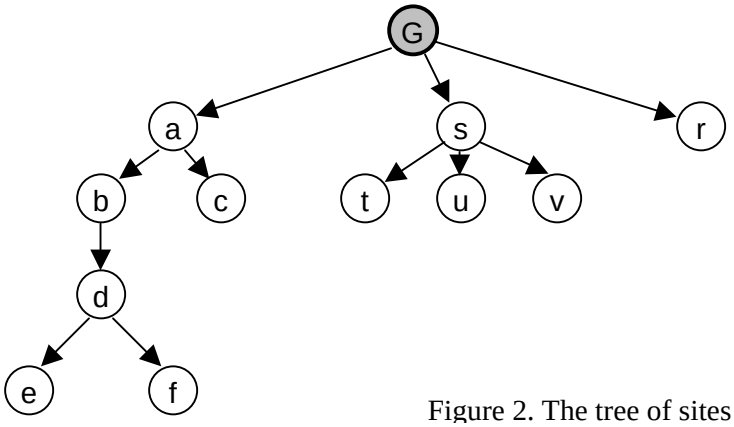

Figure 2. The tree of sites

Figure 1 shows three TM domains X, Y, and Z. A transaction is started at site a in domain X. Sites b, c, d, e and f within the same domain are then invoked. This will lead to a tree of sites rooted at a. The TM at site a is now the coordinator and TMs at other sites are subordinate coordinators. When an invocation is made from site c on site s in domain Y, site G, which for instance is a CORBA OTS server, is chosen as the external coordinator. Sites a and s are then registered to G as subordinate coordinators. r becomes a subordinate coordinator registered to G when it is invoked by u. Figure 2 shows the corresponding tree of sites.

5 Discussions

In the proposed architecture, the constructed tree of sites reflects the pattern of remote invocations within TM domains, while the roots of these trees are enlisted directly at the external coordinator.

Overhead for resolving heterogeneity, relaxing autonomy and stringent security checking is only necessary for cross-domain invocations. Due to the fact that sites in the same TM domain have higher degree of affinity than those in different domains, cross-domain interactions are relatively few. The overhead for interoperability and security will not contribute significantly to the overall performance of large transactions. Furthermore, the amount of TM domains and sites for external coordinators are relatively small with respect to the overall amount of sites, pooling “sessions” between domains and external coordinators will improve their utilization without consuming uncontrollable amount of resources.

Since intra-domain invocations are managed by domain-specific mechanisms, nearly all optimizations in these mechanisms still work. The only restriction is that optimizations will not propagate to the entire tree of sites once sites at foreign domains are invoked.

Sessions that need to be pooled and reused are those between sites within TM domains. They can still be pooled without any change. This is not unrealistic, since sessions with strong notion of states like database connections and those for load balancing and replication are only necessary within TM domains.

The use of external coordinators will also prevent the trees from getting arbitrarily deep. Optimizations like flattening of trees can still be used within TM domains. Note that such optimizations often require intervention with the remote invocation mechanisms, such as piggyback of participant addresses in return messages back to the coordinator.