Integrating Snapshot Isolation Into Transactional Federations

Ralf Schenkel and Gerhard Weikum
University of the Saarland
P.O.-Box 15 11 50
D-66041 Saarbrücken
Fax +49 681 302 4014
email {schenkel,weikum}@cs.uni-sb.de

Abstract. This paper reconsiders the problem of transactional federations, more specifically the concurrency control issue, with particular consideration of component systems that provide only snapshot isolation, which is the default setting in Oracle and widely used in practice. The paper derives criteria and practical protocols for guaranteeing global serializability at the federation level. The paper generalizes the well-known ticket method and develops novel federation-level graph testing methods to incorporate sub-serializability component systems like Oracle. These contributions are embedded in a practical project that built a CORBA-based federated database architecture suitable for modern Internet- or Intranet-based applications such as electronic commerce. This prototype system, which includes a federated transaction manager coined Trafic (Transactional Federation of Information Systems Based on CORBA), has been fully implemented with support for Oracle and O2 as component systems and using Orbix as federation middleware. The paper presents performance measurements that demonstrate the viability of the developed concurrency control methods.

1 Introduction
1.1 Objectives of a Transactional Federation

With the ever-increasing demand for information integration both within and across enterprises, on one hand, and the proliferation of gateway and other middleware technologies such as ODBC, DCOM, JEB, or CORBA, on the other hand, there is renewed interest in providing seamless access to multiple, independently developed and largely autonomously operated databases [Ston98,HSC99,MK+99]. Such a setting is known as a database federation or heterogeneous multidatabase system. More specifically, the approach of building an additional integration software layer on top of the underlying component systems is referred to as a federated database system [SL90,ÖV98]. Among the challenges posed by such a system architecture is the problem of enforcing the consistency of the data across the boundaries of the individual component systems. Transactions that access and modify data in more than one component system are referred to as federated or global transactions [BGS92]; for example, an electronic commerce application could require a transaction to update data in a merchant’s database as well as the databases of a credit card company and a service broker that pointed the customer to the merchant and requests a provisioning fee for each sale. Providing the usual ACID properties for such federated transactions is inherently harder than in a homogeneous, centrally administered distributed database system, one reason being that the underlying component systems of a federation may employ different protocols for their local transaction management.

Problems of ensuring the global serializability and atomicity of federated transactions have been intensively studied in the past. The proposed solutions range from imposing additional constraints on the transaction protocols of the component systems to building an additional transaction manager in the federated software layer [Weihl89, BGRS91,BS92,Raz92,GRS94]. However, none of the proposed approaches can by any means be considered as the universally “best” strategy for federated concurrency control. Rather the choice of the most appropriate strategy depends on the transaction protocols of the component systems, the operational characteristics of the applications, and other factors. Therefore a federated transaction manager should support a suite of different strategies and allow the application builder (or administrator staff) to select...
the most suitable protocols for the federation. In addition, it should be extensible to incorporate new transaction management strategies, specifically tailored to the needs of the application at hand. Furthermore, the federation must be able to cope with different isolation levels [BBGM+95] in the underlying component systems, for example, the "snapshot isolation" provided by Oracle [Orac99]. Such options for relaxed (local) serializability are widely popular in practice. They have only recently attracted also the research community with emphasis on formal properties [ALO00,ABJ97,BLL00, FLO+99,SW+99], but have still been more or less disregarded in the literature on federated transaction management.

1.2 Problems With Component Systems Providing Snapshot Isolation

This paper specifically addresses the problem of federated concurrency control for transactional federations based on component systems some of which provide only snapshot isolation. Note that Oracle falls into that category; although it can provide also full serializability using table locking, snapshot isolation is the best setting that can be achieved without extra coding in the application programs and widely used in practice. Further note that already running global transactions against multiple, autonomous Oracle instances all of which use the same snapshot-isolation protocol poses severe correctness problems as the resulting global execution cannot even be guaranteed to be globally snapshot-isolated. When different Oracle instances employ different isolation levels or the federation includes also database systems that provide full serializability, it becomes even more unclear how to guarantee global data consistency. As for global transaction Atomicity and durability, on the other hand, a transactional federation can rely on the logging and recovery capabilities of the underlying component systems in conjunction with a standardized distributed commit protocol like XA or its CORBA counterpart OTS, as supported by virtually all commercial database systems. Therefore, this paper focuses on the concurrency control dimension of transactional federations.

For (local) transactions run under the snapshot isolation level, or SI for short, all operations read the most recent versions as of the time when the transaction began, thus ensuring a consistent view of the data through the transaction. A particularly beneficial special case is that all read-only transactions are perfectly isolated in the sense of the multiversion serializability theory [BHG87]. For read-write transactions, on the other hand, the sketched protocol cannot ensure (multiversion) serializability. In addition, Oracle performs the following check upon acquiring a write lock: if the data object to be locked has been written by another, already committed transaction that ran concurrently to the considered one (i.e., committed after the considered transaction began), then the current transaction is aborted and rolled back. This check aims to provide an additional level of sanity. Nonetheless, the protocol cannot ensure full (multiversion) serializability, with the following schedule as a counterexample (operation subscripts are transaction identifiers, \( x_i \) denotes the version of \( x \) generated by transaction \( t_i \), and \( t_0 \) is a fictitious initializing transaction):

\[
\begin{align*}
&\ r_1(x_0) \ r_1(y_0) \quad r_2(x_0) \ r_2(y_0) \quad w_1(x_1) \ c_1 \quad w_2(y_2) \ c_2
\end{align*}
\]

The example may lead to inconsistent data, for example, violating a constraint such as \( x + y < 100 \) although both transactions alone would enforce the constraint. Given that such anomalies appear to be infrequent in practice, the protocol is widely used in Oracle applications. However, although application architects accept the risk of inconsistencies, they are not happy with his state of affairs, especially as applications become more complex, span organizational boundaries, and become even more mission-critical for business success.

1.3 Contribution and Outline of the Paper

The paper’s contributions are threefold:

- We develop a formal model that allows us to reason about local versus global snapshot isolation and serializability in the context of federated transactions.
- Based on this model, we develop novel algorithms for federated concurrency control to ensure global serializability. Our algorithms leverage prior proposals, specifically the ticket method [GRS94] and federation-level graph testing with edges derived from SQL-