

Singapore Management University Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

12-1993


Asynchronous Transaction Commitment in Federated Database Systems

San-Yih HWANG
University of Minnesota

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Jaideep SRIVASTAVA
University of Minnesota

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Citation

HWANG, San-Yih; LIM, Ee Peng; and SRIVASTAVA, Jaideep. Asynchronous Transaction Commitment in Federated Database Systems. (1993). *ICPADS '93: International Conference on Parallel and Distributed Systems, December 15-17, 1993, Taipei: Proceedings*. 440-444. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/909

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Asynchronous Transaction Commitment in Federated Database Systems

San-Yih Hwang Ee-Peng Lim Jaideep Srivastava
Department of Computer Science, University of Minnesota
Minneapolis, MN 55455

Abstract

We propose a new (and restricted) model for global transactions which allows asynchronous commitment of subtransactions. Our model requires each global transaction to have a fixed structure with update to the data in at most one database. Based on this transaction model, we present two concurrency control algorithms, namely Asynchronous Site Graph and Asynchronous VirtGlobalSG, which employ asynchronous commitment and achieve global serializability. Compared to other proposed algorithms, our algorithms employ asynchronous commitment so as to increase transaction performance. Furthermore, our algorithms do not put restrictions on transaction data access or local histories.

1 Introduction

A federated database system (FDBS) integrates and provides a uniform access to a set of pre-existing local databases, each of which is managed by its own DBMS. A key feature of FDBSs is to reduce the interference to the local DBMSs and existing local database applications. Ideally, each local DBMS and application must continue to operate without any modification in the integrated environment. Each local DBMS must be able to decide how to execute a command without any coordination from FDBS, and determine what information to reveal to the FDBS. These features are referred to as *autonomy*.

To preserve autonomy of component DBMSs, an FDBS supports two types of transactions, namely local transactions and global transactions. Local transactions are generated by the existing local applications and each local transaction accesses data from a local database. Furthermore, local transactions are submitted directly to their respective local DBMSs with the FDBS having no knowledge of them. Global transactions are submitted to the FDBS and may access data across multiple local databases.

To achieve atomic execution of both global and local transactions, a number of FDBS concurrency control algorithms, which work in conjunction with 2 phase commit (2PC) protocol, have been proposed.

In general, this requires that the local DBMSs provide a visible *prepared-to-commit* state to the FDBS concurrency control. Since providing such control information violates autonomy, various researchers have proposed that the FDBS simulates a prepared to commit state. However, unlike a real prepared to commit state, this simulated prepared-to-commit state cannot tolerate failure, i.e. a failure at the local DBMS will undo the effect of a *simulated* prepared-to-commit transaction. This causes a problem since the FDBS transaction management may have decided to commit such a transaction. A number of mechanisms have been proposed to solve this problem (e.g., see [HSL93a]). However, each of these proposed solutions imposes restrictions on the type of local site concurrency control mechanisms and the data that local transactions can access. These restrictions affect the autonomy of local databases.

Furthermore, even if the prepared-to-commit state is supported by the local DBMSs, the execution cost of the 2PC protocol can be high. Thus, some mechanisms have recently been proposed to allow asynchronous commitment and to use *compensating transactions* to achieve *semantic atomicity* (e.g., see [RELL90]). In these protocols, a subtransaction can commit without synchronizing with other subtransactions. If the subtransaction is aborted, the entire global transaction is considered aborted, and a compensating subtransaction is executed for each subtransaction that has committed earlier. Although this approach looks appealing, in an FDBS environment it is difficult, both for the systems and the users, to specify a compensating transaction for each global subtransaction.

In [HS90], a unilateral commit paradigm is proposed for distributed transaction management to commit each subtransaction unilaterally. When a subtransaction is aborted, it is *retried* until it is committed. A distributed transaction is modeled as a hierarchy of subtransactions, and a subtransaction can not be started until its parent commits. This restricted transaction model is valid only for certain applications. Besides, serializability cannot be achieved in an FDBS environment where local transactions are beyond the control the FDBS.

In [EJK91], asynchronous commitment is analyzed

from a theoretical viewpoint, and sufficient conditions are proposed to achieve global serializability in the presence of failures. The transaction model presented requires that the dependency between subtransactions of a global transaction be acyclic. The proposed sufficiency conditions require each local DBMS to produce *rigorous* histories (i.e., each local DBMS must use strict two phase locking (2PL) as its concurrency control mechanism).

To achieve asynchronous commitment of a global transaction, it is required that no two update subtransactions of the global transaction depend on each other. To ensure this, we have proposed restrictions on the structure of global transactions. Specifically, a transaction in our model consists of a number of multi-site retrieval queries and at most one single site update. This transaction model serves an interesting class of applications in which a federated application retrieves information from a number of local databases, makes its decision, and then updates at most one database. While standard 2PC has to be used for full-fledged transactions, asynchronous commitment is sufficient for the restricted transactions. No restrictions are placed on local transactions. In this paper we identify the problems encountered in asynchronous commitment. We also present two algorithms for asynchronous commitment and analyze their behavior.

The rest of this paper is organized as follows. In section 2, we describe the transaction and system models. In section 3, we identify the problems for asynchronous commit protocol and define the set of executions that is allowed for asynchronous commitment. Section 4 presents two concurrency control algorithms that employ asynchronous commitment. Finally, in Section 5 we conclude this paper.

2 Transaction Model

In this section, we present the new model for global transactions and describe their execution when subtransactions can unilaterally commit.

2.1 Transaction and Query Model

A federated database system provides a global schema, which is derived from the local schemas of component local databases. Global users issue queries to the FDBS, which operate on the global schema. A global query is decomposed into a set of subqueries, each of which is executed at a local site, based on the query execution plan produced by the FDBS query optimizer [LS93]. The execution plan for a global query, represented as a partial order, determines the dependencies among subqueries.

To allow asynchronous commit of global transactions under an FDBS environment, we restrict our-

selves to the following transaction model.

Definition 1 (*ac-transaction*) *An asynchronous commit global transaction, called ac-transaction, is a transaction that is generated from a fixed structured program¹. Each ac-transaction is a partial order of a set of global queries², among which at most one global query is an update. Furthermore, this update query can only access a single site.*

To achieve asynchronous commitment, no two write subtransactions of a global transaction depend on each other. Otherwise these two subtransactions must be committed synchronously. By definition of ac-transaction, it is clear that such a situation is precluded since there is at most one subtransaction containing update operations. When a read only subtransaction G_i mutually depends on another (read or write) subtransaction G_j , G_i can commit without the synchronous commit of G_j . In case G_j gets aborted later, G_i 's results can be simply discarded since the read operations of G_i do not cause any permanent effect on the local database. Besides, since ac-transactions require the transaction programs be fixed-structured, when some subtransactions of a global transaction are committed and others are aborted, the results of the committed subtransactions can be used when the global transaction is restarted. We will describe how to re-execute a partially committed ac-transaction in the next section.

2.2 Execution of ac-transactions

As described earlier, a global query can be represented as a partial order of a set of subqueries. Several of the subqueries, however, may access the same database. By grouping subqueries accessing the same site together, we obtain the *site dependency graph* of the global query.

A global transaction is a partial ordering of its constituent global queries, between which dependency may exist. That is, a global query can be formulated only after the values of some of its previous queries are obtained. A subtransaction is a set of subqueries that access data stored in the same local database. Considering both intra- and inter-query dependency, we define the subtransaction dependency graph of a global transaction.

Definition 2 (*Subtransaction Dependency Graph*) *Let G be an ac-transaction that has a partial order \prec_G of n global queries, $\{Q_1, Q_2, \dots, Q_n\}$, and (V_i, E_i) be the site dependency graph of Q_i , $1 \leq i \leq n$. The subtransaction dependency graph of G is a directed graph*

¹A transaction program is said to be fixed-structured if the execution of the program from any database state results in the same sequence of read and write operations [MRKS91].

²The term 'query' used in this paper has a broader meaning. It indicates either retrieval operation (e.g., 'Select' statement in SQL) or update operation (e.g., 'Update' statement in SQL).

(V, E) , where V is the set of sites the queries access, and for sites s_i and s_j in V , $(s_i, s_j) \in E$ if either

1. there exists a global query Q_k such that $(s_i, s_j) \in E_k$, or
2. there exists a pair of global queries, Q_k and Q_l , such that $s_i \in V_k$, $s_j \in V_l$, and $Q_k \prec_G Q_l$.

The subtransaction dependency graph of an ac-transaction describes the dependency relationship among its subtransactions. If a subtransaction G_i does not depend on another subtransaction G_j , the abortion of G_j does not lead to the abortion of G_i . However, when a subtransaction T_2 depends on another subtransaction T_1 , T_1 may read a data item from one site, and the value of that data item is used by some subquery of T_2 at another site. Therefore, when a subtransaction (T_1) is aborted, all of its dependent subtransactions (including T_2) must be aborted.

Once some subtransactions of an ac-transaction G abort, G must be re-executed. However, since other subtransactions of G may have committed, their results are still valid and should be used in the re-execution. The following describes the re-execution procedure, RESTART-PROC, which is invoked when a subtransaction G_i of an ac-transaction G is aborted.

1. All subtransactions of G that depend on G_i are aborted. Each executed subquery in the query execution plans of the global queries in G that does not belong to the aborted subtransactions is marked as “EXECUTED”.
2. Re-execute the global transaction G . If a subquery is marked as “EXECUTED”, skip it and use the previous result, and continue executing the next subquery.

3 Issues in Asynchronous Commitment

To achieve consistency, a global subtransaction cannot unilaterally commit without any control. In this section, we examine the impact of asynchronous commitment on global serializability and identify the property that an asynchronous commitment algorithm must satisfy in order to achieve consistent execution.

With asynchronous commitment, an anomaly may occur where some subtransactions of an ac-transaction have committed and other subtransactions cannot commit without violating global serializability. In this case, the partially committed ac-transaction can never fully commit, and is said to be *starved*.

We now formally define a desirable property of the set of transaction executions, namely *starvation free serializability*, which must be ensured by any FDBS concurrency control algorithm employing asynchronous commitment.

Definition 3 (Pending Work) A subtransaction G_i of an ac-transaction G is said to be in the pending work of G in a global history H if either

1. G_i is not committed in H , or
2. G_i is read-only and committed in H , and G_i either has no dependent subtransactions or all its dependent subtransactions are in the pending work of G .

The pending work of an ac-transaction is the maximum amount of work (subtransactions) that can be discarded when it is restarted.

Definition 4 (Starvation-Free Serializability) Let $C(h_i)$ be the committed projection of a local history h_i . A global history $H = \{h_1, h_2, \dots, h_n\}$, where h_i 's are local histories, is said to be starvation-free serializable if, for each non-fully committed ac-transaction³ G , the following modified global history is serializable:

$$\{C(h_1) - G_1^p | G_1^p, C(h_2) - G_2^p | G_2^p, \dots, C(h_n) - G_n^p | G_n^p\},$$

where $\{G_1^p, G_2^p, \dots, G_n^p\}$ ⁴ is the pending work of G , and $C(h_i) - G_i^p | G_i^p$ means the operations of G_i^p are removed from $C(h_i)$ and appended to its end with a commit operation.

A starvation-free serializable history is such that, for each non-fully committed ac-transaction, by discarding the pending work of the ac-transaction and immediately re-executing the pending work, the resultant global history is serializable. Thus, each non-fully committed ac-transaction in a starvation-free serializable history still has the chance to fully commit.

4 Concurrency Control Algorithms with Asynchronous Commitment

In this section, we propose two FDBS concurrency control algorithms that employ asynchronous commitment.

4.1 Asynchronous Site Graph

Site graph algorithm was proposed by Breitbart and Silberschatz in [BS88]. It is a pessimistic approach that assumes potential conflicts exist between every pair of global transactions executed concurrently at the same site. Site Graph algorithm maintains an acyclic bipartite graph called Site Graph. The two

³An ac-transaction is said to be non-fully committed in a history if there exists at least one subtransaction whose commit is not in the history.

⁴ G_i^p is ϕ if either G does not execute at site i or the subtransaction of G at site i is not in G 's pending work.

```

repeat
  wait(event)
  case event of
  An ac-transaction  $G$  is newly started:
    Add an edge between  $G$  and each site node to which a
    subtransaction of  $G$  is sent;
    if (the induced edges of  $G$  cause cycles in Site Graph)
    then
      Delete all the induced edges of  $G$ ;
       $G$  is blocked;
    else
      Execute  $G$  according to the query execution plans;
    endif
  Global subtransaction  $G_s$  of  $G$  at site  $s$  is aborted by the
  local DBMS:
    Re-execute  $G$  according to (RESTART-PROC);
  All operations of global subtransaction  $G_s$  of  $G$  are finished:
  if ( $G_s$  is read only)
  then
    Commit  $G_s$ ;
  else
    Wait until all subtransactions of  $G$  on which  $G_s$ 
    depends commit and then commit  $G_s$ ;
  endif
  if ( $G_s$  is the last committed subtransaction of  $G$ )
  then
    Inform the user about the commit of  $G$ ;
  endif
until FALSE;

```

Figure 1: **Site Graph with asynchronous commit**

vertex sets are formed by the set of global transactions and sites, respectively, while edges connect a transaction with all sites where its subtransactions execute. Before a global transaction can be executed, a set of edges, which connect the transaction node to all site nodes the global transaction is going to access, is added to the Site Graph. A global transaction is allowed to execute only when the induced edges do not cause cycles in the Site Graph.

Figure 1 shows the Asynchronous Site Graph algorithm where asynchronous commit protocol is employed. Theorem 1 demonstrates the correctness of this algorithm.

Theorem 1 *The Site Graph with asynchronous commitment always generates starvation-free serializable histories. [HSL93b]*

4.2 Asynchronous Virtual Global Serialization Graph

The Virtual Global Serialization Graph (VirtGlobalSG) was used in DAGSO (i.e., Dynamic Adjustment of Global Serialization Order) to achieve global serializability[HHS93]. In this section, we further revise the VirtGlobalSG based approach to make it work with the asynchronous commit protocol.

The VirtGlobalSG is a graph that maintains the serialization orders of global transactions on various local executions. Specifically, VirtGlobalSG is a di-

rected graph (V, E) with edge labels, where V is the set of global transactions, and an arc $(G_i, G_j, s) \in E$ if the subtransaction of G_i is serialized before the subtransaction of G_j at site s , and no other subtransaction is serialized between G_i and G_j at site s .

A global subtransaction is validated before it is started and after it is serialized. When a global subtransaction of G starts at site s , an arc (LastSerialized(s), G, s) is added to VirtGlobalSG, where LastSerialized(s) is the most recently serialized subtransaction at site s (since it can be determined that G at site s must be serialized after LastSerialized(s)). When a global subtransaction of G at site s is serialized, a set of arcs $\{(G, T, s) \mid T \text{ is an ac-transaction that has been executed but not serialized at site } s\}$ are added to VirtGlobalSG (since it can be inferred that all non-serialized subtransactions executed at site s must be serialized after G at site s). If the induced edges of a subtransaction at a validation point cause a cycle in the VirtGlobalSG, the global subtransaction is aborted.

To prevent the anomalies caused by intra- and inter-transaction dependencies, we need to determine when a global subtransaction can unilaterally commit. In summary, a subtransaction G_s of G at site s can commit only when all of the following conditions (referred to as COMMIT-COND) hold:

1. All operations of G_s are finished.
2. If G_s contains an update query, then
 - (a) all subtransactions of G that G_s depends on in the subtransaction dependency graph have committed, and
 - (b) all subtransactions of G have been started.
3. All subtransactions serialized before G_s at site s have committed.

The algorithm for using VirtGlobalSG with the asynchronous commitment protocol is presented in Figure 2.

Theorem 2 *The VirtGlobalSG with asynchronous commitment always generates a starvation-free serializable history. [HSL93b]*

5 Conclusions

We discussed the issues related to asynchronous commit of global transactions in a federated database system. We have proposed a global transaction model for asynchronous commitment. We require each global transaction to be fixed structured and have at most one update operation. We also presented two concurrency control algorithms, namely Site Graph and

```

repeat
  wait(event)
  case event of
  Global subtransaction of G at site s is newly started:
    if (the induced arcs cause cycles in VirtGlobalSG)
      then
        abort subtransactions of G to resolve the cycles;
      endif
  Global subtransaction of G at site s is re-started:
    if (the induced arcs cause cycles in VirtGlobalSG)
      then
        abort other subtransactions at site s to
        resolve the cycles;
      endif
  Global subtransaction of G at site s is serialized:
    if (the induced arcs cause cycles in VirtGlobalSG)
      then
        abort subtransaction of G at site s;
      endif
  Global subtransaction of G at site s is aborted by the
  local DBMS:
    abort subtransaction of G at site s;
  All operations of global subtransaction of G at site s are
  finished:
    wait until (COMMIT-COND)
    commit the global subtransaction of G at site s;
until FALSE;

```

Figure 2: **Asynchronous VirtGlobalSG**

VirtGlobalSG, which employ asynchronous commitment and achieve global serializability.

Asynchronous commitment algorithms have the following advantages:

- Simplify the recovery procedures.
- Preserve autonomy.
- Decrease the chance of deadlocks.
- Increase the performance of both global and local transaction execution.

However, these advantages come at the cost of a stricter transaction model, i.e., ac-transactions. It is impossible to have asynchronous commitment on a general transaction model, while at the same time achieving global serializability. There is a tradeoff among *performance*, *transaction model*, and *correctness criterion*. Our work imposes reasonable restrictions on global transaction model and obtains performance gains, while maintaining the desirable correctness criterion of global serializability.

Acknowledgements

The idea of asynchronous commitment emerges from our discussion with Dr. Jiangdong Huang at Honeywell Technology Center. We would also like to thank him for his useful comments on this paper.

References

- [BS88] Y. Breitbart and A. Silberschatz. Multi-database update issues. In *Proc. of ACM SIGMOD Int'l. Conf. on Management of Data*, 1988.
- [EJK91] A. K. Elmargarmid, Jin Jing, and Won Kim. Global commitment in multi-database systems. Technical Report 91-017, Dept. Computer Sci., Purdue Univ., IN, 1991.
- [HHS93] S.-Y. Hwang, J. Huang, and J. Srivastava. Concurrency control in federated databases: A dynamic approach. In *Proc. of the 2nd Int'l Conf. on Information and Knowledge Management*, 1993.
- [HS90] M. Hsu and A. Silberschatz. Unilateral commit: A new paradigm for reliable distributed transaction processing. In *Proc. of the 7th Int'l Conf. on Data Engineering*, 1990.
- [HSL93a] S.-Y. Hwang, J. Srivastava, and J. Li. Transaction recovery in federated autonomous databases. *Distributed and Parallel Databases, An Interational Journal, to appear*, 1993.
- [HSL93b] S.-Y. Hwang, J. Srivastava, and E.-P. Lim. Asynchronous transaction commitment in federated database systems. Technical Report 93-54, Dept. Computer Sci., U. of Minnesota, MN, 1993.
- [LS93] E.-P. Lim and J. Srivastava. Query optimization/processing in federated database systems. In *Proc. of the 2nd Int'l Conf. on Information and Knowledge Management*, 1993.
- [MRKS91] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. Non-serializable executions in heterogeneous distributed database systems. In *Proc. of the 2nd Int'l Symposium on Databases in Parallel and Distributed Systems*, 1991.
- [RELL90] M. Rusinkiewicz, A. Elmargarmid, Y. Leu, and W. Litwin. Extending the transaction model to capture more meaning. *SIGMOD Record*, 19(1), 1990.