Concurrency Control and Recovery of Multidatabase Work Flows in Telecommunication Applications

W. Woody Jin
Marek Rusinkiewicz

Dept of Computer Science
University of Houston
Houston, TX 77204

Linda Ness
Amit Sheth

Bellcore
445 South St.
Morristown, NJ 07962

Abstract

In a research and technology application project at Bellcore, we used multidatabase transactions to model multi-system work flows of telecommunication applications. During the project a prototype scheduler for executing multidatabase transactions was developed. Two of the issues addressed in this project were concurrent execution of multidatabase transactions and their failure recovery. This paper discusses our use of properties of the application and the telecommunication systems to develop simple and efficient solutions to the concurrency control and recovery problems.

1 Introduction

Service Provisioning is the activity of setting up a telecommunication service based on a customer's requests. A provisioning activity involves a coordinated access to multiple heterogeneous telecommunication application systems, called Operation Support Systems (OSSs). Each OSS typically manages one function such as billing, or inventory control of a class of circuit facilities. We believe that such work flows can be effectively modeled as multidatabase transactions.

A multidatabase transaction model called *Flexible Transactions* were introduced in [7, 10] to model global transactions in heterogeneous multidatabase systems. The flexible transaction model separates the specification of the protocol for coordinating the component transactions of the global transactions from the specification of the component transactions. The coordination protocol governs both the control flow and data flow among the component transactions.

The applicability of the Flexible Transaction model to Bellcore Service Provisioning applications was evaluated by specifying two such applications using it and by developing a prototype system for executing such flexible transaction [2]. The prototype implementation has shown that many work flow applications can be declaratively specified as flexible transactions and provided the first evidence that it is possible to automatically synthesize a practical executable implementation from the specification.

Although the preliminary results were promising, the scope of the original implementation was quite limited. The earlier scheduler did not address the issues of concurrent execution of flexible transactions or the problems that may be caused by the failure of the scheduler. It is quite clear that both of these problems would have to be solved before it is established that the flexible transactions technology can be used to successfully support Service Provisioning Applications in an industrial environment. In this paper, we discuss the implementation of concurrency control and failure recovery for applications based on the flexible transaction model. We discuss the use of modified altruistic locking [9] as the proposed concurrency control mechanism. We also discuss the failure recovery of the Concurrency Controller and schedulers of individual flexible transactions. Where possible, we exploit our knowledge about the application and about the system environment to design simpler and more efficient solutions.

2 Flexible Transaction Model

A flexible transaction (FT) [7, 10] is specified by providing: the precondition of the global transaction, a set of subtransactions, the externally visible states of each subtransaction and the possible transitions among these externally visible states, preconditions and postconditions for the possible transitions of each subtransaction, and the postcondition of the global transaction.

Thus, a flexible transaction includes a protocol for coordinating the control and data flow among the component transactions. This protocol is specified by a state machine. The precondition for the global transaction specifies its legal initial states. The postcondition of the global transaction

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0456...$1.50

specifies the set of conditions defining its legal (i.e. *acceptable*) final states; some of these legal final states may be viewed as successful completion of the global transaction; others may be viewed as "successful (clean) failure" of the global transaction. The set of externally visible states of a subtransaction usually includes: *initial, executing, committed, prepared to commit, aborted.* The preconditions and postconditions may be stated in terms of the state variables for the subtransactions and additional variables. The preconditions may express dependencies based on the execution state of other subtransactions, as well as data dependencies or temporal dependencies. Non-trivial postconditions for transitions may be necessary to express correctness criteria on output messages containing parameters necessary for initialization of the subtransactions. Preconditions for transitions may express correctness criterion on input messages.

The traditional transactions are usually characterized by the atomicity, consistency, isolation and durability requirements, called the ACID properties of transactions. To better support performance and autonomy requirements, the Flexible Transaction Model relaxed the isolation and atomicity properties.

The full atomicity is replaced by the guarantee that a flexible transaction may terminate only in one of the acceptable states. Some of these acceptable states may involve compensation of some transactions. The consistency (correctness) requirement is replaced by the correctness criterion for flexible transactions, which is the existence of a scheduler that guarantees that a flexible transaction can reach one of the acceptable states, for any initial state of transaction and the multidatabase system. The isolation granularity of a flexible transaction is a subtransaction. Finally, the durability is guaranteed by assuming that each local system provide the durability of its committed subtransactions and by assuming that the state of the scheduler persists subsequent to system malfunctions.

The flexible transaction model allows both *compensable* [8] and *noncompensable* subtransactions to coexist within a single FT. If compensating transactions are defined, the compensation is considered correct, if both the original transaction and its compensating transactions commit, and the compensating transaction is serialized after the transaction it compensates. If the original transaction is not executed or aborted, the compensating transaction must not be committed.

3 Properties of Telecommunications Environment

An important part of our prototype implementation is a scheduler that controls the execution of a single flexible transaction and determines when various subtransactions can be scheduled [3]. Once scheduled for execution, a subtransaction is given to the execution agent that communicates with an OSSs to execute the subtransaction and passes back the results to the scheduler. The remainder of the paper outlines the concurrency control and failure recovery aspects of the scheduler.

To design concurrency control and recovery mechanisms that are suitable for our class of applications, we carefully analyzed the process of Service Provisioning. We have identified a number of properties of the Operation Support Systems involved in the provisioning than may affect the choice of a suitable method. These properties are listed below:

- *Order Preservation*: In every database system involved in the process, the submission order of transactions determines their execution order, as long as the transactions belong to the same class. There are only two priority classes - *hot* (express) and normal.

- *Relaxed Isolation*: The granularity of isolation is a (sub)transaction. The results of a committed subtransaction become visible to other concurrently executing transactions immediately. If the provisioning activity (flexible transaction) fails, these results are undone by executing compensating transactions.

- *Limited commutativity* For some systems involved in the process of provisioning, the subtransactions submitted to these systems are commutative, i.e., they can be executed in any order. However, one can not assume that all subtransactions are commutative. For example, if an allocate-transaction and its corresponding deallocate-transaction run concurrently, the result is unpredictable and the multidatabase system can be left in an inconsistent state. This is because if special precautions are not taken, a transaction performing a de-allocation of some data item may be executed before the corresponding allocate transaction is performed. Therefore, a data item may be left in the inventory marked as "allocated", even though its correct status should be "free".

- *Idempotency*: The systems involved in the process of provisioning have the "at least once" semantics i.e., the execution of a transaction can be repeated without causing any inconsistency. Usually, when the same request is issued twice, once the first is done, the second is rejected.

- *Monotonicity*: Once the preconditions for the execution of a (sub)transaction become true they do not become false later. In particular, the (sub)transaction can be undone by executing a compensating transaction if the scheduling preconditions for the compensating transaction are satisfied.

The above properties are used in the design of practical concurrency control and recovery mechanisms for the flexible transactions used to model Service Provisioning workflows. Order preservation together with the assumed rigorousness [4] of the local concurrency controllers allows the

early release of locks. Relaxed isolation requirements eliminate the need for global commitment protocols. Limited commutativity means that exclusive locks are required only for some classes of subtransactions. As a result the concurrency control mechanism we designed allows a higher degree of concurrency, than the one possible without this knowledge of the applications and system enironment.

The remaining properties significantly simplify the recovery mechanisms. Since the systems are idempotent in the case of a failure of the scheduler it can safely resubmit all transactions that were scheduled and for which no acknowledgement was received at the time of failure. This and the monotonicity property allow the roll-forward recovery of transactions that were in progress at the time of a failure from the re-do logs.

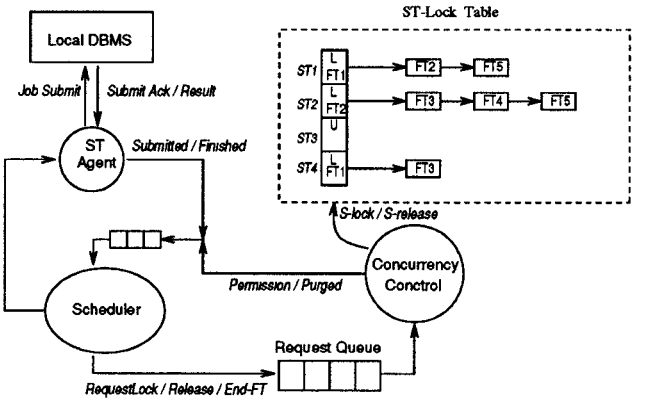

Figure 1: Execution Environment for Flexible transactions

4 Scheduling of Flexible Transactions

The process of scheduling of a flexible transaction is presented in Figure 1. The scheduler first checks for satisfaction of the preconditions for execution of each subtransaction, i.e., determines whether a subtransaction is *schedulable*. A schedulable subtransaction may be submitted for execution to the transaction agent. Once the agent notifies the scheduler that the subtransaction is completed, the scheduler checks whether an acceptable state has been reached. If this is the case, the flexible transaction is completed, otherwise the scheduling preconditions for subtransactions are checked again and new subtransactions may become schedulable. The process continues until the flexible transaction completes successfully or until it reaches a compensated state. In a multi-user environment, the scheduler must consult with the concurrency controller to decide whether a subtransaction may be submitted for execution. It also needs to log the state information in the secure storage, so that the scheduling can be resumed in case of a failure. These issues will be discussed in detail in the following sections.

4.1 Concurrency Control

Multidatabase transactions and the related concurrency control techniques are reviewed in [5]. In particular, a number of methods can be used to assure *global serializability* which constitutes a suitable correctness criterion for concurrent execution of multidatabase transactions, in the absence of additional information about their semantics. In the discussion below, we will attempt to take advantage of the properties of the Service Provisioning described above, to relax the requirements of global serializability and allow a higher degree of concurrency. With this objective in mind, we will introduce the concept of *FT-serializability*, which we believe to be a suitable correctness criterion for our environment.

The objective of concurrency control is to assure that the (relative) serialization order of multidatabase transactions should be the same, at all sites they execute. [6] have shown that the above condition is sufficient to assure global serializability. However, in the provisioning environment this requirement can be relaxed to require that the (relative) serialization order of Flexible Transactions should be the same *only at those sites where they conflict*. This would lead to a weaker notion of serializability (called FT-serializability), which will serve as our correctness criterion for concurrent execution of Flexible Transactions.

We first must define *conflicts* among flexible transactions. Commutativity is a property of a particular class of transactions. For example, two *allocate* transactions may be commutative, if they allocate different items, for even though different items may be allocated if the transactions are run in a different order, the allocated items are "satisfactory". Two (sub)transactions *conflict* iff they execute at the same (local) system, and they are not commutative.

The conflict relation is transitive, and hence determines a set of equivalence classes, which we will call *conflict classes*. Conflict classes can be used to determine the granularity of locking. To define flexible transaction serializability (*FT-serializability*), let us consider two flexible transactions FT_x and FT_y, and conflict classes, i and j.

A global schedule is *FT-serializable* if for any subtransactions ST_i^x and $ST_j^x \in FT_x$, and ST_i^y and $ST_j^y \in FT_y$ such that conflict(ST_i^x, ST_i^y) and conflict(ST_j^x, ST_j^y), $ST_i^x \prec ST_i^y \Rightarrow ST_j^x \prec ST_j^y$, at all sites they conflict. The $\prec$ relationship is defined in terms of local serializability. FT-serializability establishes a (partial) order among all flexible transactions.

To ensure FT-Serializability, we assume that each scheduling automaton of a flexible transaction sends a lock request to the global Concurrency Controller to determine whether a given subtransaction can be scheduled. Once the lock is granted the subtransaction may proceed further, otherwise it must wait.

Because of the assumption stated earlier, the submission order at each system, can be used to determine the execution and, eventually, the serialization order at each site. We rely on the concurrency control mechanism of the local system to assure that the transactions that are submitted to the

local system will be executed correctly with respect to the local concurrency control. Therefore, the lock held by a subtransaction can be released as soon as the subtransaction completes its submission phase. Hence, we may have several transactions that are executing concurrently at each site.

The proposed algorithm is similar to the altruistic locking introduced in [1]. We have taken advantage of the properties the systems involved in the provisioning to reduce the duration of the locks from the time needed to complete a transactions to the time needed to submit it. Although we use the same locking granularity as altruistic locking, our algorithm allows a higher degree of concurrency.

4.2 Failure Recovery

The objective of recovery mechanisms in transaction management is to enforce the failure atomicity of global transactions. In the case of flexible transactions, the recovery procedures must make sure that a failed flexible transaction eventually reaches an acceptable state – possibly the compensated one. We assume that the local systems involved in Service Order Provisioning use their local recovery mechanisms to handle their local failures. Therefore, in our discussion, we will concentrate on handling the failures of the execution controllers for flexible transactions, which include schedulers, concurrency controller and the local agents.

To recover the execution environment context, we need to restore the state informations that concurrency controller and schedulers had at the time of failure. The state information of the concurrency controller is status of its lock table. The status of the scheduler includes the information about the execution states of each subtransaction and the information about the scheduling permissions received from the concurrency controller. This information must be kept in the local logs maintained separately by the concurrency controller and the schedulers.

One of the major advantages of our failure recovery scheme is its *modularity*. Each scheduler and the concurrency controller may recover from a failure using only the information from its own log. Since the transaction agents are responsible for the execution of a single (sub)transaction there is no need to recover their state. Once a failure of an agent is detected (for example, by setting up a timeout) the agent process can be simply restarted. This cannot cause any harm, since we assume that the local systems are idempotent.

5 Summary

In this paper, we showed how current technologies for heterogeneous multidatabase systems, (flexible transactions combined with altruistic locking), can be applied to improve a large class of telecommunication industry applications, Service Provisioning. We first, analyzed the characteristics of the Service Provisioning, and the existing systems that participate in processing the applications, and then developed suitable transaction management techniques. By taking advantage of the specific properties of the local systems involved in the Service Order Provisioning, we were able to design simple and efficient concurrency control and recovery mechanisms. These mechanisms are less general than the solutions proposed in the literature for multidatabase transactions, however they allow a higher performance in the specific *real world* environment in which they are applied.

References

[1] R. Alonso and H. Garcia-Molina and K. Salem, *Concurrency Control and Recovery for Global Procedures in Federated Database Systems*, A quarterly bulletin of the Computer Society of the IEEE technical committee on Data Engineering, Vol. 10, No. 3, September 1987.

[2] Mansoor Ansari, Marek Rusinkiewicz, Linda Ness and Amit Sheth, *Executing Multidatabase Transactions*, Technical Memorandum TM-TSV-019450, July 1991.

[3] Mansoor Ansari, Linda Ness, Marek Rusinkiewicz, and Amit Sheth, *Using Flexible Transactions to Support Multi-system Telecommunication Applications*, Proceedings of the 18th VLDB Conference, Vancouver, British Columbia, Canada 1992.

[4] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silberschatz. *On Rigorous Transaction Scheduling*, IEEE Transactions on Software Engineering, 17, September 1991.

[5] Yuri Breitbart, Hector Garcia-Molina and Avi Silberschatz, *Overview of Multidatabase Transaction Management*, Tech Report No. STAN-CS-92-1432, Stanford University, 1992.

[6] Y. Breitbart and A. Silberschatz, *Multidatabase Update Issues*, Proceedings of ACM SIGMOD International Conference on Management of Data, June, 1988.

[7] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz *A Multidatabase Transaction Model for Interbase*, Proceedings of the 16th VLDB, August 1990.

[8] J.N. Gray, *The Transaction Concept Virtues and Limitations*, Proceedings of 7th International Conference on VLDB, September 1981.

[9] K. Salem and H. Garcia-Molina, *Altruistic Locking: A strategy for coping with long-lived transactions*, Proceedings of the 2nd International Workshop on High Performance Transaction Systems (September 1991), Amdahl Corporation, IBM Corporation.

[10] M. Rusinkiewicz, A. Elmagarmid, Y. Leu, and W. Litwin, *Extending the Transaction Model to Capture More Meaning*, SIGMOD Record, March 1990.