On Incremental Global Update Support in Cooperative Database Systems

Chengfei Liu  
School of Computer and Information Science, Uni. of South Australia, Adelaide,SA 5095 Australia  
Phone: +61 8 8302 3287, Fax: +61 8 8302 3381  
E-mail: chengfei.liu@unisa.edu.au  
AND  
Xiaofang Zhou  
Dept. of Computer Sci. & Electrical Engineering, Uni. of Queensland, Brisbane QLD4072 Australia  
Phone: +61 7 3365 3248, Fax: +61 7 3365 4999  
E-mail: zxf@csee.uq.edu.au  
AND  
Jinli Cao  
Dept. of Mathematics & Computing, Uni. of Southern Queensland, Toowoomba QLD4350 Australia  
Phone: +61 7 4631 1619, Fax: +61 7 4631 1775  
E-mail: cao@usq.edu.au  
AND  
Xuemin Lin  
School of Computer Sci. & Engineering, Uni. of New South Wales, Sydney NSW2052 Australia  
Phone: +61 2 9385 6493, Fax: +61 2 9385 5995  
E-mail: lxue@cse.unsw.edu.au

Keywords: Multidatabases, Global Transaction Management, Transaction Classification

Edited by: Yanchun Zhang, Vladimir Fomichov, and Anton P. Železnikar
Received: August 15, 1999 Revised: November 10, 1999 Accepted: November 15, 1999

OzGateway is a cooperative database system designed for integrating heterogeneous existing information systems into an interoperable environment. It also aims to provide a gateway for legacy information system migration. This paper summarises the problems and results of multidatabase transaction management research. In supporting global updates in OzGateway in an evolutionary way, we introduce a classification of multidatabase transactions and discuss the problems in each category. The architecture of OzGateway and the design of the global transaction manager and servers are presented.

1 Introduction

Data management technology has been evolving rapidly. While modern database management systems (DBMSs) and client-server based distributed computing environments are available now, most large organisations are still using out of fashion technologies (e.g., COBOL, pre-relational DBMSs) and mainframe computers to manage their data. It is currently a high priority task to find methodology of integrating existing information systems to meet the application requirements in an interoperable environment, and the ways for making legacy information systems (ISs) reusable by converting them into target ISs which use new database technologies [Brodie and Stonebraker, 1995].

A multidatabase system (MDBS) is a software interface to provide users uniform access to multiple, heterogeneous database systems. The component systems of an MDBS may include DBMSs of different designs and models, and possibly some file systems. In spite of different and possibly redundant and conflicting representations of objects using different models at different locations, a user can access information as if he/she is using a single centralised DBMS at the multidatabase level. Major issues in current multidatabase research include schema integration and global transaction management [Hurson et al., 1994].

OzGateway is a cooperative database system which aims to provide solutions for integrating heterogeneous existing information systems into an interoperable environment. It also aims to be a gateway system for legacy information system migration. In this paper, we discuss global update support in OzGateway. We know that supporting global update in multidatabases is very difficult. However, in many multidatabase applications there are no global constraints at all, since each site was developed independently and may wish to remain independent. Because of this, we introduce a classification of multidatabase transactions and discuss the problems for each category of transactions.

The rest of the paper is organised as follows: In section 2, problems and results in current multidatabase transaction management are summarised. The classification of multidatabase transactions is introduced in Section 3. In section
4, the architecture of OzGateway and a preliminary design of the GTM and servers are presented. Section 5 concludes the paper.

2 Multidatabase Transaction Management

In an MDBS, global transactions are executed under the control of the MDBS, and at the same time local transactions are executed under the control of the local DBMSs. Each local DBMS may employ a different transaction management scheme (or even no transaction management at all). In addition, each local DBMS has complete control over all transactions (both global and local) executing at that site, including the ability to abort any transaction at any point at its site. Typically, no design or internal DBMS structure changes are allowed in order to accommodate the MDBS. Furthermore, the local DBMSs may not be aware of each other and, as a consequence, cannot coordinate their actions. These issues make the transaction management in MDBS very difficult.

A local DBMS offers a set of operations, which can be classified into two classes: one deals with transaction operations (such as transaction_begin, transaction_end, abort, commit, prepare_to_commit), another deals with transaction status information (such as get_wait_for_graph, get_serialization_order and get_transaction_status). In general, a local DBMS does not export its wait_for_graph or serialization order when participating the MDBS. If a local DBMS only allows a database user (the MDBS transaction manager is nothing more than a local DBMS user to the LDBMS) to submit a transaction as a whole (i.e., from transaction_begin to commit), the MDBS has no control as to when it is executed.

In multidatabase transaction management three problems remain to guarantee [Breitbart et al., 1992]: global serialisability, atomicity of global transactions, and deadlock-free executions of global transactions. The presence of local transactions could make a serial execution of global transactions not satisfy global serialisability as some possible invisible relationship among global transactions could be introduced by local transactions. Global serialisibility can be achieved by forcing otherwise invisible conflicts by letting transaction $T_1$ write some objects on site $s$ and letting $T_2$ read these objects if $T_1$ proceeds $T_2$ (denoted as $T_1 \rightarrow T_2$) in the global serialisibility graph and site $s$ is involved in the execution of both transactions [Georgakopoulos et al., 1991]. Many other methods have also been proposed for global serialisibility: strongly serialisable scheduling [Alonso et al., 1987] given all local systems use the basic timestamping order, serialisation-point [Pu, 1988] (e.g., the first operation in the timestamping scheme, the first lock release in 2PL, the last operation commit in strongly recoverable scheduling [Breitbart et al., 1991]), scheduling based on rigorous local DBMSs [Breitbart et al., 1991], $\epsilon$ — serialisability, two level serialisability [Mehrotra et al., 1991], etc.

Atomicity of global transactions is difficult to support as a local DBMS does not have any obligation to the global transaction execution coordinated by the GTM, i.e., it does not usually export the prepare_to_commit operation, it can abort its local branch of a global transaction unilaterally at any time before commit, therefore the 2PC protocol can not be used. A server, when the local DBMS does not support the prepare_to_commit operation, can be used to participate in the global 2PC protocol on behalf of the local system. When a server votes to commit a global transaction but the LDBMS aborts the global subtransaction, the server has to redo or retry the global subtransaction. Another approach is the Compensate approach, which allows the GTM to semantically undo any committed global subtransactions.

Similarly, deadlock-free executions of global transactions are hard as the GTM cannot access the wait_for_graph for local transactions. Based on communication ordering and time-out, an approximation global wait_for_graph is used by the GTM to detect all deadlocks but also some false deadlocks.

All the above methods may result in poor performance or bring in some restrictions in use. It is very difficult, if not impossible, to give satisfactory solutions to these global transaction management problems without sacrificing local autonomy.

3 A Classification of Transactions

There are two types of transactions in an MDBS: global and local transactions. The fundamental distinction is by the data they access. A data item is global if it can be seen at the global level (i.e., that data item has been exported by the local system and been imported by the global system); otherwise it is local. A transaction is local to an LDBMS if it uses local data only, or uses local data and global data exported from the local system. A transaction is global if it uses only global data. Note that it is not a valid multidatabase transaction if it access a mixture of local data and global data from other local systems. The MDBS is not aware of any local transactions, but a local transaction can use global data which is exported from the local system. This is where most problems of multidatabase transaction management arise from.

From our discussion in the above section, one can see that it is very hard for the GTM to support global updates if local sites do not provide at their interface level sufficient transaction control commands or internal transaction status information. Most global transaction management solutions are based some assumptions about local systems. If any of the local sites cannot meet the requirement, the global transaction management has to sacrifice performance to use a less efficient method to compromise with that local system. Given that a multidatabase system
consists of a large number component systems, which may join or withdraw from the multidatabase system on their own merits, an MDBMS may have to always be based on the weakest assumption about local systems. This leads to poor performance.

Another problem of global update comes from schema integration. A multidatabase relation is often integrated from several local relations. To maintain local autonomy, such an integration is usually a view integration (as opposed to database integration which physically merges relations into a single global relation). In other words, such a global relation is a view defined from joining several relations; thus, it cannot always be updated. Another example of not being able to update a global relation comes from the common practice in multidatabase integration of applying some transformation rules on the local data items. For example, in order to integrate some component systems which store a piece of land using its geometry data, and some other component systems which store land areas, the global data can use areas. A simple transformation rule can be applied to those systems using geometry data. It is clear that the update of area in the global relation is impossible as such an update cannot be done in those component systems using geometry data.

We have seen that global update in an MDBS is difficult, and is not always possible. Now we discuss from a positive perspective: is this necessary? Given that the GTM can become very simple and much more efficient if global update is not to be supported or partly supported, it should be carefully decided whether it is worthwhile to support global update. We introduce a classification of transactions here. Which transactions an MDBMS should support, and how they are supported will be discussed later.

Transactions in an MDBS can be classified into the following categories:

1. **SWSR (Single-Write, Single-Read) transactions**: They read and update data from the same local system. Such transactions can be either global or local.

2. **SWMR (Single-Write, Multiple-Read) transactions**: They read data from multiple sites, but only update data from one site. They are global transactions.

3. **MWSR (Multiple-Write, Single-Read) transactions**: They read data from one single site, but update multiple local systems. They are global transactions.

4. **MMWR (Multiple-Write, Multiple-Read) transactions**: They read as well as update data from multiple sites. They are global transactions.

From the transaction management point of view, there is no difference between MWSR transactions and MWMR transactions. Thus we only discuss the MWMR transactions.

Many applications only need to share information among sites with the agreement that data can only be updated by the owner. Recent research on providing an integrated view of a variety of legacy data also falls into this category [Roth and Schwarz, 1997, Haas et al., 1997, Levy et al., 1996]. There are two ways to facilitate updating of data by its owner. The simplest way is to withdraw data from MDBS for update [Ahmed et al., 1991]. Therefore, this data cannot be seen at the global level during the update, and the update transaction is a local SWSR transaction. The updated data can join the multidatabase system later. By assuming that each local DBMS can maintain local consistency and handle local deadlock, it is obvious that there will be no need to consider these issues at the global level. However, besides the problem of data availability during the update, a major problem of this approach is that it can be very costly, particularly when the data to be updated are in a very large relation and the local and global data formats are different.

If a local transaction updates local data as well as its exported data, it is still a local SWSR transaction. This is most often the case as old local applications should be able to run without change even if some of its data have been exported to the global system. The GTM may not know the existence of these local SWSR transactions. Therefore, possible effects caused by the presence of such transactions should be considered in the design of GTM. Two global serialisability problems will still occur even without other types of update transactions:

1. (1) a global retrieval transaction reads dirty data from an aborted local update transaction;
2. (2) inconsistent reads between two global retrieval transactions due to local update transactions.

The first problem can be solved if each local system employs strict 2PL. The second problem can be compromised if global consistency is not considered. Fortunately, there are no global deadlock and atomicity problems.

While an SWSR transaction can always be regarded as a local transaction, it is also possible to consider an SWSR transaction as global if it accesses only exported the data of a local system. All SWSR transactions can be “globalised” by temporally “globalising” the local data they use. This does not violate local autonomy, because only an interface shell needs to be added to the local system.

Now consider a common case where an application needs to read data from multiple sites to make a decision about how to update its own data. The user may wish to keep the data to be updated in the global system to make their application program simpler. Therefore, an MDBMS should support the global update transactions which read data from multiple sites, but update only the data on their own local systems. From the transaction management point of view, it has no much difference whether the update is on local site or not, as long as only one site is involved for each transaction to update. So this kind of transactions are SWMR transactions. The Oracle Procedural Gateway only supports SWMR transactions [Sandroffini, 1994]. To support this category of transaction, there is no need to implement the 2PC protocol; and atomicity is easy to maintain. However, global deadlock is possible.

The hard applications are those MWMR transactions,
which read and update data managed by multiple DBMSs. All three problems remain in this category of transactions. We do not consider this at the moment as we think this is not often, and global consistency is not important. For example, a travel agent to prepare a travel for a client may need to book air-tickets from multiple airlines, book accommodation from different hotels and book cars from some companies. These bookings are related to each other as some or all bookings may have to be cancelled if other bookings cannot be made. This application needs to update the databases of several airlines, hotels and car rental companies. However, there is no global integrity constraint, as long as each local system is consistent.

4 Design of OzGateway

OzGateway is a cooperative database system which aims to provide solutions for integrating heterogeneous existing information systems into an interoperable environment. It also aims to be a gateway system for legacy information system migration. Figure 1 is the general architecture of OzGateway. A global user issues a global query using a global query language against a global schema. The Global Query Processor decomposes the global query into a set of single-site subqueries, which are organised as a query graph according to data dependency and query processing cost. OzGateway supports applications which may consist of part of legacy systems and part of target systems. One of the subqueries is to be executed at the OzGateway internal DBMS to merge the intermediate results from other subqueries.

The GTM dispatches these subqueries to the servers of the corresponding local systems. Global concurrency control may be considered depending on the category of transactions being supported. A server translates the subquery received into local query language and then passes the translated query to the local system as an ordinary local query. It reports execution status to the GTM, and passes the subquery results, if the execution succeeds, to the internal DBMS after translating the subquery results into the OzGateway common data format. The GTM reports execution failure to the user and aborts all other subqueries on receiving a failure report from any server, or delivers the global query results to the user when the internal DBMS finishes data merging if none of the subqueries fail. The architectures of the GTM and a server are shown in Figure 2 and Figure 3, respectively.

OzGateway supports global updates along an evolutionary path. At the first stage, we are keen to set up a multidatabase environment to enable the user to share information which can otherwise not be shared. Therefore, global update is not allowed (all updates are through local SWSR transactions). At the second stage, we will support SWMR transactions, which are sufficient to support legacy system migration. In time, MWMR transactions may need to be supported; however, this is not considered in our current design. To enable OzGateway to evolve from stage 1 to stage 2 smoothly, we should avoid a total restart when stage 2 begins. This is realised by using the open GTM architecture. We believe that the next stage only needs to substitute a limited number of components.

Sitting between the GTM and a local DBMS, a server
passes the subtransaction and data to the local DBMS, and reports the execution status of the subtransaction and returns result back to the GTM after necessary translation.

To support legacy ISs, a server serves as a gateway which translates requests at the global level to legacy ISs. Functions which cannot be performed by legacy ISs at the local level are either performed by the OzGateway internal DBMS or by the server. In the later case, a server also serves as a wrapper which supports a set of functions required at the global level in terms of local systems, especially for legacy ISs. For instance, it is possible for the server to enhance the local system such that a better GTM can be expected. This is particularly important when the local system is a file system, or a legacy IS. In supporting 2PC, a server can also simulate a prepare_to_commit status by redoing or resubmitting the subtransactions aborted by the local DBMS after the server votes to commit.

5 Conclusion

OzGateway is designed as a vehicle to facilitate research in multidatabase systems and legacy information system migration. In this paper, we re-examined the problems in a MDBS environment. To minimize the tasks of the GTM, a classification of multidatabase transactions was introduced and problems in each category were discussed. The general architecture, the GTM, and the servers of OzGateway were presented. Currently, we are investigating various approaches to problems in each transaction category.

References


