BULETINUL Universității Petrol – Gaze din Ploiești	Vol. LVIII No. 1/2006	37 - 42	Seria Matematică - Informatică - Fizică

Need for group transactions in multimedia and design databases

Monica Vlădoiu, Cătălina Negoiță

Universitatea Petrol-Gaze din Ploiești, Bd. București 39, Ploiești, Catedra de Informatică
e-mail: mvladoiu@upg-ploiesti.ro

Abstract

Classical concurrency control assumes that transactions are short. This is far too radical for usually long multimedia/design transactions (e.g the locking concurrency control has problems because a long transaction can block the access of others to its locks during its execution time; in optimistic control conflicts, they are detected only at the end, so rollback is unacceptable). An appropriate model for design/multimedia process, as an ensemble of sub-tasks, seems to be the group transaction. This allows the exchange of partial data and the restoration of not committed data by using as many intermediary database levels as needed between the private and the public database.

Keywords: *multimedia databases, concurrency control, group transaction*

Issues with management of transactions in multimedia and design databases

Databases, be them multimedia or not, are strongly shared resources that are concurrently accessed both by numerous application programs and many end-users. This behavior implies the need for a control component, which must control the access to the shared data, in order to avoid the undesired interferences between the concurrent processes. The core concept of this component is the transaction. Classical transactions lead to concurrency control and to error recovery, which addresses the limitation of unpleasant effect of various errors.

Transaction management is mandatory as a result of the need for data sharing consistently and concurrently. Consistency refers to respect the integrity constraints on the database. If transactions are executed serially, the consistency is naturally enforced. Serializability is the major criterion for the correctness of concurrent transactions' executions (i.e., transactions that have overlapping execution time intervals, and possibly access same shared resources), and a major goal for concurrency control. As such it is supported in all general-purpose database systems.

The rationale behind it is the following: If each transaction is correct, then a serial execution of these is correct. As a result, any execution that is equivalent (in its outcome) to such serial execution is correct. A transaction schedule is serializable if it is equivalent (in its outcome, the resulting database state) to a serial schedule (serial schedule: no overlap in two transactions' execution time intervals is allowed, i. e. consecutive transaction execution is provided).

The key transaction processing features of a Database Management System (DBMS) are the ACID properties. Without them, the integrity of the database cannot be guaranteed. In practice, these properties are often relaxed somewhat to provide better performance. An example of a transaction is a transfer of funds from one account to another, even though it might consist of multiple individual operations (such as debiting one account and crediting another). The ACID properties guarantee that such transactions are processed reliably. The ACID features are:

- *Atomicity* refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them are. The transfer of funds can be completed or it can fail for a multitude of reasons, but atomicity guarantees that one account will not be debited if the other is not credited as well;
- *Consistency* refers to the database being in a legal state when the transaction begins and when it ends. This means that a transaction cannot break the rules, or integrity constraints, of the database. If an integrity constraint states that all accounts must have a positive balance, then any transaction violating this rule will be aborted (in a nutshell all transactions must leave the database in a consistent state);
- *Isolation* refers to the ability of the application to make operations in a transaction appear isolated from all other operations. This means that no operation outside the transaction can ever see the data in an intermediate state; a bank manager can see the transferred funds on one account or the other, but never on both—even if she ran her query while the transfer was still being processed. More formally, isolation means the transaction history (or schedule) is serializable. For performance reasons, this ability is the most often relaxed constraint (transactions cannot interfere with each other);
- *Durability* refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and will not need to abort the transaction. Typically, all transactions are written into a log that can be played back to recreate the system to its state right before the failure. A transaction can only be deemed committed after it is safely in the log (successful transactions must persist through crashes.).

Implementing ACID correctly is not simple. Processing a transaction often requires a number of small changes to be made, including updating indices that are used by the system to speed up searches. This sequence of operations is subject to failure for a number of reasons (e. g. the system may have no room left on its disk drives, or it may have used up its allocated CPU time).

ACID suggests that the database be able to perform all of these operations at once. In fact this is difficult to arrange. There are two popular families of techniques: *Write ahead logging* and *Shadow paging*. In both cases, locks must be acquired on all information that is updated, and depending on the implementation, on all data that is being read. In write ahead logging, atomicity is guaranteed by ensuring that information about all changes is written to a log before it is written to the database. That allows the database to return to a consistent state in the event of a crash. In shadowing, updates are applied to a copy of the database, and the new copy is activated when the transaction commits. The copy refers to unchanged parts of the old version of the database, rather than being an entire duplicate.

Almost all databases used nothing but locking to ensure they were ACID until recently. This means that a lock must be acquired anytime before processing data in a database, even on read operations. Maintaining a large number of locks, however, results in substantial overhead as well as hurting concurrency. If user A is running a transaction that has read a row of data that user B wants to modify, for example, user B must wait until user A's transaction is finished.

An alternative to locking is to maintain separate copies of any data that is modified. This allows users to read data without acquiring any locks. Going back to the example when user A's transaction gets to data that user B has modified, the database is able to retrieve the exact version of that data that existed when user A started their transaction. This ensures that user A gets a consistent view of the database even if other users are changing data that user A needs to read. It is difficult to guarantee ACID properties in a network environment. Network connections might fail, or two users might want to use the same part of the database at the same time. Two-phase commit is typically applied in distributed transactions to ensure that each participant in the transaction agrees on whether the transaction should be committed or not.

Classical concurrency control involves short transactions (from milliseconds to seconds). This assumption is reflected, for instance, by the atomicity condition that enforce the complete rollback of a not committed transaction in case of an error occurrence. This requirement is far too radical for design or multimedia transactions, that both are long transactions (spanning from hours to days, or even weeks). Having said that, we can presume that design and multimedia databases long transaction are needed. This presumption has the following consequences:

- complete rollback for these long transactions must be avoided by introducing intermediary save points despite that the saved objects are not yet consistent;
- blocking long duration transactions have to be prevented to happen, so that the concurrency will be stimulated;
- inconsistent information exchange between design/MMDB users should be allowed;
- simultaneous and concurrent designing is desired – those could be done by allowing concurrent updates on the same object to take place, with special concern to insurance of inconsistency detection;
- to offer means for reflecting the organizational structure of the design-multimedia process;
- transactions must provide support for checkIn/checkOut from a client-server architecture;
- the user has to be empowered with concurrency control.

Despite all the above relaxations of classical concurrency mechanisms, in many cases serializability must be ensured all the same. Consequently, in those cases the new proposed transaction model should expand the classical transaction paradigm instead of replacing it.

The proposed solution

A suitable model for the organization of a design or multimedia process as an ensemble of sub-tasks is the group transaction. The premise for this is the assumption that each task is being executed by a group of designers or developers working together. The main idea behind this model consists in the introduction of an intermediary level between the public and private databases that corresponds to the partial project databases (known as well as semi-public or group databases). Within this organization the checkIn/checkOut operations can take place both between public and group database and involving private and group databases (see Figure 1).

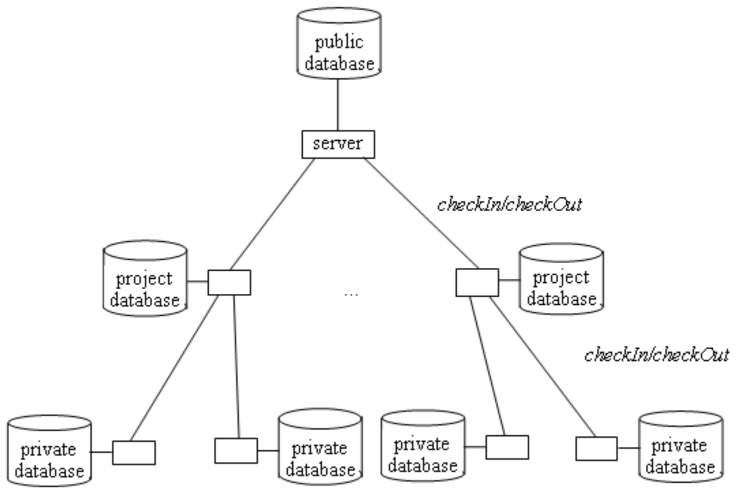

Fig 1. Group transaction model's hierarchy

This simple idea has led to a special transaction model, which accomplishes the requirement that incomplete parts of the developed project should be visible to other group member, without making them publicly available. Within this model there are two kinds of transactions: group, respectively user transactions. Each group transaction is associated with a group of designers who co-operate to develop a common task or project. When a group transaction is created, a correspondent group database will be created as well. The group transaction can checkIn/checkOut between the public database and its associated group database, as in the classical case. For group transaction, serializability is easily ensured due to the close collaboration between the members of each group.

For each group transaction, correspondent user transactions will be created as sub-transactions of it. So the user transactions will depend totally on the transaction that has created them. A private database will be started for each user transaction. CheckIn/checkOut operations between group and user database can be done afterwards. The main restriction within this model is that the user must not have direct access to the public database. Besides the checkIn/checkOut operations, the group transaction model allows operations as creation and modification of new group of users, starting and ending both group and user transactions. This model has only three database levels and two transaction levels. A generalization of this model, which permits an arbitrary number of imbrication levels, will allow nested transactions to take place. Finally, a multimedia database transaction involves all its sub-systems - these are at least three: hierarchical storage management, multimedia database system and information retrieval system.

Concurrency control

Concurrency control is a method used to ensure that database transactions are executed safely (i.e., without data loss). Concurrency control is especially applicable to DBMSs, which must ensure that transactions are executed safely and that they follow the ACID rules. The DBMS must be able to ensure that only serializable, recoverable schedules are allowed, and that no actions of committed transactions are lost while undoing aborted transactions. So, concurrency control must synchronize the actions of transactions that take place simultaneously, by preventing their interference. The best known concurrency control algorithms are: locking, multi-versions and optimistic algorithms (time stamps sorting, committing time certification):

- *locking* is the most used synchronization mechanism. The basic idea is that any persistent object has an associated lock. Before a transaction can access this object it has to request the access to this lock. If another transaction has already got access to this lock or to a lock which conflicts with the new request, then the current transaction has to wait till the lock is unlocked (examples of such locks are exclusive or share). The lock can be achieved to various database levels, from simple records to compound objects, or even to the whole database. The locking algorithms are pessimistic, in the sense that they reject a lock access request if there is only the possibility that something goes wrong;
- *multi-versions* offer the possibility to create a new version of a database object for each update transaction. This mechanism of version control is used in synchronization insurance, to improve the concurrency between the transactions that only "read" from the database and the ones that only "write". Two of the main requirements of the models of transactions that apply to the design and multimedia environments refer to the support for a high degree of concurrency and to the exchange of partial objects. The version management can be used to increase the concurrency and to facilitate the exchange of project objects, which are partially consistent, among the co-operating transactions. The configurations are object collections that are treated both as locking and as versioning units. At their level the consistency is to be ensured, regardless of the versions of the various components;
- *optimistic algorithms* allow all the transactions to start their execution before trying to achieve their own changes in the database. Then a validation procedure is launched to verify

if conflicts have occurred - if a conflict is detected the transaction is rolled back. The process that determines if a transaction can safely be committed is called certification. Within this context a transaction is divided in three phases: reading, validation and writing. Within the reading phase each object that is accessed by a transaction is copied in a separate working space, which is owned by each transaction. All transaction updates will be executed on these local copies. No locking of a transaction takes place. During the transaction execution in this phase, the concurrency control collect certain operations that will be later used in the validation phase to detect conflicts (such as the list of all objects that are read by the transactions, respectively the list all objects that are written by the transactions). After the reading phase when the transaction enters into the validation phase, a time stamp will be attached to it. This stamp is an element of a monotone increasingly sequence (as the time moment when the transactions ends the reading phase). During this phase of a transaction T three criteria are evaluated for each transaction T' whose time stamp precedes T's. If one of the criteria is accomplished, then serializability is guaranteed. The three criteria are: (1) T' has ended its writing phase before T has started its reading phase, (2) the set of elements that T' has written do not intersect with the set of elements that T reads and T' finishes its writing phase before T starts its writing phase, and (3) the set of elements that T' has written do not intersect with the set of elements that T reads and with the set of elements that T writes. If any of the above conditions is accomplished for each transaction T' that precedes T, according to the time stamps, then T may enter its writing phase. Otherwise T must be stopped and it should enter a rollback operation (which is trivial as all the updates have been executed on local copies). During the writing phase, all the objects that have been modified by the transaction are committed in the database. This is the moment when all the updates that this transaction has done become visible to the other transactions.

Error recovery

The condition of transaction atomicity implies that the multimedia DBMS must guarantee that the partial results of the transactions that fail must not be propagated in the persistent database. There are three types of errors that the recovery manager has to deal with: storage media, system or transaction errors. Errors related to the storage medium are solved usually by data replication, either in the database by mirroring, either in log databases. System errors, hardware or software, lead only to the destruction of the data from the volatile memory, the one from the secondary storage support remaining intact. A transaction can fail from various reasons: either it is involved in a deadlock situation and it has been chosen to be rolled back, or there are inconsistencies related to respecting the integrity constraints or the user aborts it.

Recovery error module restores the database in a consistent state if an error occurs by using a backup copy of the whole database, which is consistent because during its construction no update transaction has been allowed. This operation is time and material consuming, so it is not done often - there is an up limit to the frequency to which the backup is efficient. However, just doing backup copies is not enough because due to durability, the updates committed by the committed transactions must not be lost. To avoid this, one should consider that between the database states are only differences. The necessary information to error recovery is in the before-image, here the after image and in the logical logs files, which memorize the committed transactions that will be re-executed in case of an error occurrence.

Conclusions

Classical control of concurrency assumes that database transactions are short time operations. This is reflected by the atomicity that enforces the rollback of an un-committed transaction, in case of an error. This request is far too radical for multimedia or design transactions, as such operations can include the developer's intellectual effort within hours, days or even weeks. If one thinks of the optimistic control in which the conflicts where detected only at the end it becomes obvious why such an approach is unrealistic for multimedia/design transactions. The locking concurrency control can also have similar problems if we think that a long time transaction can block the access of other transactions to the locks it holds during its execution time. This is unacceptable. This reasoning is still true for the transaction of access and processing of large multimedia data amounts (e. g. movies). An appropriate model for the organization of a design or multimedia process as an ensemble of sub-tasks seems to be the group transaction. This model allows the exchange of partial data and the restoration of not committed data by using as many intermediary database levels as needed between the private and the public database.

References

1. Aizawa K., Nakamura Y. - *Advances in Multimedia Information Processing - PCM 2004: 5th Pacific Rim Conference on Multimedia, in Lecture Notes in Computer Science*, Springer; 2005
2. Candan K.S., Celentano A.- *Advances in Multimedia Information Systems: 11th International Workshop MIS 2005, in Lecture Notes in Computer Science series,* Springer 2005
3. Furht B.- *Multimedia Technologies and Applications for the 21st Century*, Kluwer Academic Publishers, 1998
4. Hirzalla N.B., Karmouch A.- A multimedia query specification language*, in Nwosu K., Thuraisingham B., Bruce Berra P., Multimedia Database Systems, Design and Implementation Strategies*, Kluwer Academic Publishers, 1996
5. Khoshafian, S.- *Multimedia and Imaging Databases*, Morgan Kaufmann, 1995
6. Lee K., Lee Y.K., Berra P.B.- Management of Multi-structured Hypermedia Documents: A Data Model, Query Language, and Indexing Scheme, *in Multimedia Database Management System - Research, Issues and Future Directions,* Kluwer Academic Publishers, 1997
7. Royo J. D., Hasegawa G.- *Management of Multimedia Networks and Services: 8th International Conference on Management of Multimedia Networks and Services, MMNS 2005, Barcelona, Lecture Notes in Computer Science Publisher:* Springer; 2005
8. Subrahmanian V.S.- *Principles of multimedia Database Systems*, Morgan Kaufmann Pub. Inc., San Francisco, CA, 1998
9. Yu C.T., Meng W.- *Principles of database query processing for advanced applications,* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998
10. * * * *Multimedia Computing and Networking 2004 - Proceedings of S P I E (International Society for Optical Engine)* California, 2004

Nevoia de tranzacții de grup în bazele de date multimedia și de proiectare

Rezumat

Controlul clasic al concurenței presupune că tranzacțiile au durată scurtă. Această cerință este mult prea radicală în cazul tranzacțiilor multimedia sau de proiectare, de vreme ce o astfel de tranzacție este de durată (de exemplu în cazul bazat zăvorîrii pot apărea probleme pentru că o tranzacție de lungă durată poate bloca accesul altor tranzacții la zăvoarele pe care ea le deține, pînă cînd se va termina; în controlul optimist conflictele sînt detectate abia la sfîrşit, deci rularea înapoi este inacceptabilă). Un model potrivit pentru organizarea unui proces de proiectare sau a unuia multimedia ca un ansamblu de sub-task-uri pare să fie modelul tranzacției de grup. Acesta permite schimbul de date parțiale şi restaurarea datelor "ne-comise" în baza de date prin folosirea atîtor nivele intermediare de baze de date câte sînt necesare între baza de date publică şi bazele de date private.