Transaction Management Using Serial Validation Approach for Relational Database

Han Ni Kyaw : Marlar Win Khin
University of Computer Studies, Yangon
hnk.honeybee@gmail.com

Abstract

Nowadays, accessing the same database from the different applications is natural when one company wants to sell their products not only by itself but also by other partner companies or agencies. The accesses to the database can be transactions from many client applications. It needs to monitor, examine, and control the concurrent accesses so that the overall correctness of the database is maintained. In our case study, we implements data accessed components which uses the serial validation with optimistic concurrency scheme to ensure the serial equivalence of the overlapping client transactions. The use of a multi-tiered architecture to develop the overall system architecture will meet the best architecture design decision to distributed deployment of these components as a separate tire on the web. These components can be connected via their interfaces from any application client and concurrency control is achieved through them.

1. Introduction

In order to preserve data consistency, concurrency control is used in Database Management System (DBMS) [Gray and Reuter, 1994]. The objective is to ensure transactions isolation whenever there are concurrent operations requesting access to the same object; these must be coordinated in order to prevent inconsistencies [Härder and Rothermel, 1993; Lu and Satyanarayanan, 1994], in such a way that each user perceives he/she is the only one with access to the objects (data) [Gama and Alvarado, 2002]. This way, the DBMS is required to guarantee the isolation property [ANSI, 1992; Adya et al., 2000; Lu and Satyanarayanan, 1994], by transaction serialization. Concurrency control schemes that had been suggested to achieve the effect can be broadly classified into two categories –

Pessimistic concurrency control scheme - any access to an object must be preceded by a request and ***Optimistic concurrency control scheme*** - work done as part of a transaction is synchronized or validated at the end of the transaction.

In our case study, airline reservation system, we use multi-tiered component software architecture based on the J2EE technology. The optimistic concurrency control mechanism with serial validation algorithm is implemented on the business tire component so that it can be consumed by remote and distributed application for future purpose. The business tire uses the data access tire in which we use the technique called object relation mapping (ORM) which maps database table to objects and table columns to objects' field and it doesn't need to worry about complex sql works and cascade database operations. Our concurrency method allows the concurrent access of a single table when the individual columns of the table are accessed by separate users or applications. Transactions are validated before being allowed to commit. This reduces the delay of waiting for a table to be free for access and decreases the delay of rolling back transactions that are concurrently accessing a table. The reduction of these delays increases the overall data processing efficiency for the system. For system design purpose, we use MVC design pattern to meet the airline reservation systems nature such as frequent changing of user interface, business rules and policies.

2. Related Work

In 1981, H.T Kung and John T. Robinson proposed non-locking optimistic concurrency control methods. They apply their approach in concurrent insertion to B-trees. After this, their method had been applied in systems such as banking, airline, and e-shop. In our system, we use another technique called object-relational-mapping to reduce delay when performing transactions operations and to ensure automatic cascade operations.

3. Theoretical Background

This section introduce the background theories needed for the propose application.

3.1. Optimistic Concurrency Control

'Optimistic' because it is based on the observation that, in most applications, the likelihood of two clients transactions accessing the same object is low. Transactions are allowed to proceed as though there are no possibility of conflict with other transactions until the client complete its task and commit. When a conflict arises, some transaction is aborted and will need to restart by the client. Each transaction has the following three phases:

Read: Here, all writes are to private storage.
Validation : Make sure no conflicts have occurred.
Write : If Validation was successful, make writes public. (If not, abort!)

3.1.1. Read and write phase

Read phase: In this phase each transaction has a copy of the most recently committed version of each of the objects that it updates. The use of tentative versions is to allow transactions to abort with no effect on the objects. In addition, two records are kept of the objects accessed with a transaction: a read set containing the objects read by the transaction and a write set containing the objects write by the transaction.

Write phase: If the transaction succeeds in validation, it is considered committed in the system and changes are made to the database. Otherwise the transaction is rolled back or restarted at the read phase.

3.1.2. Validation Phase

Validation is made to ensure that the scheduling of a particular transaction is serially equivalent with respect to all overlapping transactions-i.e, any transactions that had not yet committed at the time this transaction started but committed at the time this transaction enters validation phase.

Serial Validation : The validation of transaction T_v checks whether its read set overlaps with any write set of earlier transaction Ti. If there is any overlap, the validation fails. The following program describes the algorithm for the validation of T_v.

```
Boolean valid = true;
for (int Ti = startTn+1; Ti <= finishTn; Ti++)
{  if (read set of Tv interests write set of Ti)
     valid=false;
}
```

Let startTn be the biggest transaction number at the time when transaction T_v and finishTn be the biggest transaction number at the time T_v enters validation phase.

Parallel validation : The only different with the serial validation is that now must check using *active*, the set of transactions that have finished their read phase but have not yet completed their write phases. With parallel validation a transaction may cause another transaction to abort and then abort itself and it has more overhead as compare to serial validation.

3.2. Model View Controller Pattern

The main idea behind the MVC architecture is to separate the business model functionality from the data presentation and control logic. The **'Model'** is how the underlying data is structured. The **'View'** is what is presented to the user or consumer. The **'Controller'** is the element that performs the processing. Separating these three elements makes it easier to achieve loose coupling, because it makes it possible for the controller to work with multiple different Model and View components.

3.3. Multi-tier architecture

Multi-tier just refers to the physical structure of an implementation. i.e. the presentation layer may on some web servers, then that talks to backend application servers over the network for business logic, then that talks to a database server. MVC abstracts away the details of how the architecture of an application is implemented. Nowadays MVC design is often implemented using a multi-tier architecture. This design pattern make easier maintenance process since changing in business logic cannot affect in presentation layer, and vice versa.

4. Propose system

In this system, user can reserve the tickets, view the flight schedules and send the feedback. The application is implemented in separate tiers including presentation tier, business tier, data access tier and persistence tier.

4.1. Serializability

Serializability is checked before transactions write their updates to the database. Serial equivalence is the widely used criterion for correctness of concurrent execution of transactions.

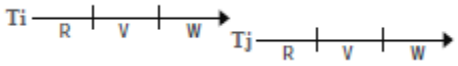
Let transactions T1,T2,... ,Tn, be executed concurrently. If the initial data structure is $d_{initial}$ and the final data structure is d_{final}, the concurrent execution of transactions is correct if some permutation π of {1,2, . . . , n} exists such that

$d_{final} = T_{\pi(n)} o T_{\pi(n-1)} o..... T_{\pi(2)} o T_{\pi(1)} d_{initial}$ (1)

where "o" is the usual notation for functional composition.

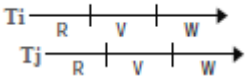
The use of validation of serial equivalence as a concurrency control is a direct application of eq. (1) above. To verify (l), a permutation π must be found. This is handled by explicitly assigning each transaction T, a number t(i) during the course of its execution. The meaning of transaction numbers in validation is the following: there must exist a serially equivalent schedule in which transaction Ti comes before transaction Tj whenever t(i) < t(j). This can be guaranteed by the following validation condition: For each transaction Tj with transaction number t(j), and for all Ti with t(i) < t(j); one of the following three conditions must hold:

(1) Ti completes its write phase before Tj starts its read phase.



(2) The write set of Ti does not intersect the read set of T;, and Ti completes its write phase before T; starts its write phase.

(3) The write set of Ti does not intersect the read set or the write set of Tj, and Ti completes its read phase before Tj completes its read phase.



4.2. Serial Validation

Serial validation guaranties the validation condition 1 and 2 and write phase must be serial. To implement this, place the assignment of a transaction number, validation, and the subsequent write phase all in a critical section. In the following, we bracket the critical section by "<" and ">". Let tnc be the global transaction number.

```
tbegin = (
    create set := empty;
    read set := empty;
    write set := empty;
    delete set := empty;
    start tn := tnc)
tend = (
    <finish tn := tnc;
    valid := true;
    for t from start tn + 1 to finish tn do
    if (write set of transaction with transaction
       number t intersects read set) then
        valid := false;

    if valid then
     ((write phase); tnc := tnc + 1; tn := tnc)>;
    if valid then ( cleanup )
    else (backup)).
```

Figure 1. Serial validation algorithm

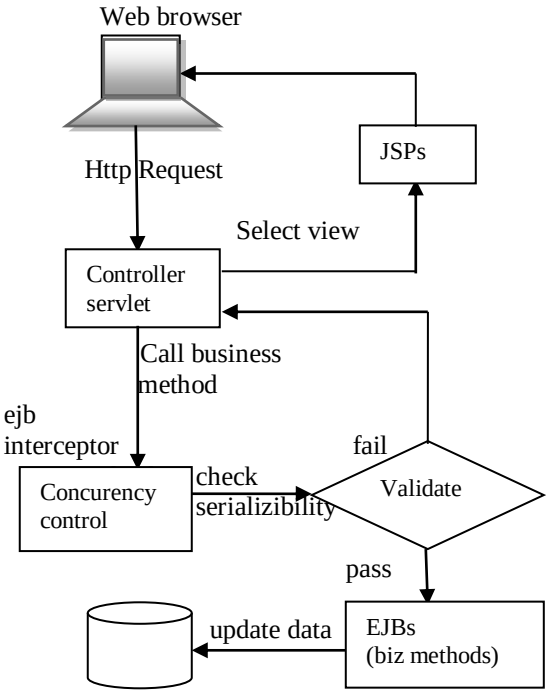


Figure 2. Overview system design

4.3 Client transaction management

The system records which class of the flight and how many tickets user want to reserve and put them in associate read sets and write sets for each transaction. After that the system performs two or three phases: a read phase, a validation phase and a possible write phase. During the read phase, assign the startTn as the largest committed transaction number at the time and all writes take place on local copies of the nodes to be modified.

After the read phase, the validation phase enters [see figure.1] and sets the finishTn as the largest committed transaction number at the time. During the validation phase, current write sets are validated against with earlier committed transaction in the range of startTn+1 to finishTn. If the

validation fails, the current transaction is rollback. It the validation succeeds, the write phase enters and changes are made to the database.

4.4 Advantages of the system

The advantages of the propose system are as follows. Since concurrency control algorithm is implemented in the ejb interceptor class, only the transaction that passes the validation can access the enterprise java beans (meant to perform server-side operations; highly available (24x7), fault-tolerant, multi-user, secure; ejb client can even be an application on different server) methods. The nature of the airline reservation system is that its presentation design and business are always changing. Since the application is implemented in separate tiers (presentation, business, data access, and persistence), changing in one tier cannot affect the rest of the tiers. In data access tier, we use object relational mapping. It has two main advantages in our system: Firstly, it provides database heterogeneity. Second, it increases the performance of the algorithm. We do not need to write complex sql join nor not need to worry about cascade update to the database.

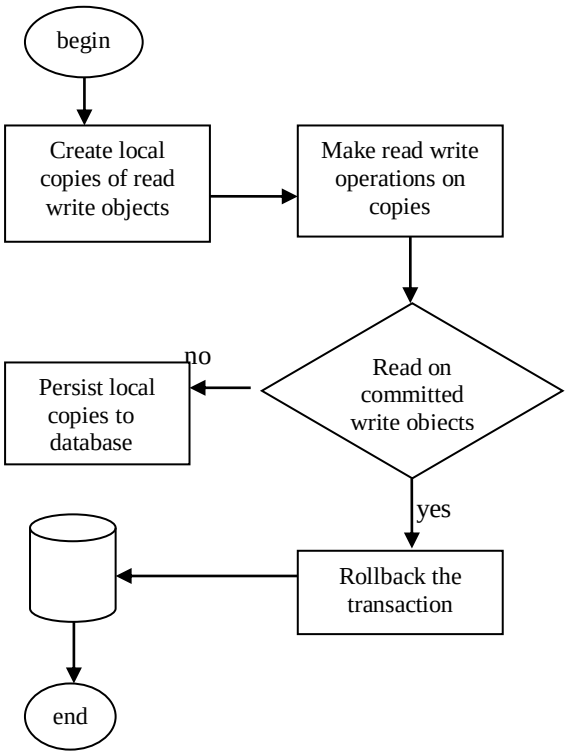


Figure 3. Optimistic concurrency control with serial validation algorithm flow.

A DETAILED EXAMPLE. This example illustrates ideas of optimistic and its advantages. We assume that a history is presented to a scheduler.

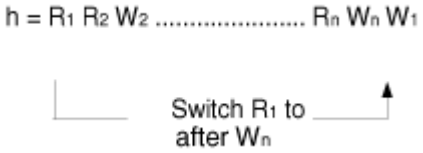
In this example, we use *Ri* and *Wi* to represent the read action (read set) and the write action (write set) of a transaction *Ti*. Let *h* be an input history of *n* transactions to the scheduler as follows:

$h = R1R2W2R3W3 \ldots. RnWnW1$

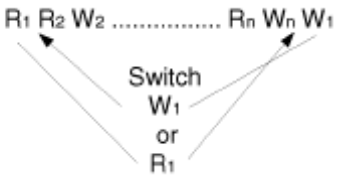
Here transaction *T*1 executes the read actions, followed by the read/write action of *T*2, *T*3,.., *Tn*, followed by the write actions of *T*1. Suppose *R*1 and *W*2 conflict as represented by an edge as follows:

The history *h* is not allowed in the locking protocols because *W*2 is blocked by *R*1. If *T*1 is a long transaction and *T*2 is a small transaction, the response time for *T*2 will suffer. In general *T*1 can block *T*2, *T*2 can block *T*3 (if *T*2 and *T*3 have a conflict) and so on. Let us consider several cases in optimistic approach.

Case 1: For the history *h*, in the optimistic approach of Kung and Robinson, *Ti* (*i* = 2,…, *n*) can commit. Write sets (*Wi*s) of committed transactions are saved to validate against the read set of *T*1. Basically the conflict preserving exchange (switch) as follows is attempted so that *R*1 can be brought next to *W*1.



Case 2: An extension of this idea is to try the either exchange (switch) as follows:



The resulting histories can be either

$R1W1R2W2 \ldots.. RnWn$

or

$R2W2 \ldots.. RnWnR1W1.$

For switching *W*1, we would need to save not only the write sets of committed transactions, but also the read sets of committed transactions. This

will allow more histories to be acceptable to the scheduler.

5. Future Work

The system's database is accessed through enterprise java beans via concurrency control. If other systems such as airline agencies want to use these beans services, that application must be written in java language. Usable to applications written in other languages require converting these business beans to web services.

6. Conclusion

Today many dynamic web applications uses database system and maintaining the integrity and consistency of the data is very important since invalid data state can cause great impact to the mission critical system. It needs to use appropriate concurrency mechanism to protect the data. The goal of the concurrency control algorithm is to allow several transactions to be executing simultaneously such that collection of manipulated data item is left in a consistent state and final result should be same as if each transaction ran sequentially. The main advantage of optimistic concurrency control method is deadlock free and allow for maximum parallelism as compare with other concurrency control mechanism.

7. Referenes

[1] Rima Patel Sirghesh, Gerald Brose, Micah Silverman "Mastering enterprise java beans", 2006.

[2] Bharat Bhargava, Fellow, IEEE, "Concurrency control in database system", 1999.

[3] Atul Adya, "Transaction management for mobile object using optimistic concurrency control" Massachusetts Institute of Technology, 1994.

[4] H.T. Kung and John T. Robinson, "An optimistic method for concurrency control", Carnegie-Mellon University, 1981.

[5] Michael J. Carey and Michael R. Stonebraker "The performance of concurrency control algorithms for database management system" university of Wisconsin and university of California.

[6] Andrae Muys "A Concurrency Control Protocol for Multiversion RDF Datastores" university of Queensland, 2000.

[7] Bernstein, P. et al., "Concurrency Control and Recovery in DatabaseSystems", Addison Wesley, 1987, Chapter 2: Serializability Theory.

[8] Coulouris, Dollimore and Tim Kindberg, "distributed systems concepts and design",Addison Wesley 2001.

[9] Amit P. Sheth and Ming T. Liu. Integrating locking and optimistic concurrency control in distributed database systems. In Proceedings of the 6th International Conference on Distributed Computing Systems, May 1986.

[10] Skeen D. "Nonblocking Commit Protocols" proceedings of the ACM SIGMOD International Conference on Management of Data, pages 133{142, May 1981}.