Research Journal of Recent Sciences _____ ISSN 2277-2502
Vol. **1 (ISC-2011),** 426-429 **(2012)** *Res.J.Recent Sci.*

Mini Review Paper

Concurrency Issues of Distributed Advance Transaction Process

Sheetlani Jitendra and Gupta V.K.
Department of Computer Science, NIMS University, Jaipur, Rajasthan, INDIA

Available online at: www.isca.in
(Received 11th October 2011, revised 27th February 2012, accepted 29th 2012)

Abstract

The transactional model for distributed system has been around for many years and it is considered a well-established and mature technology. The traditional transaction model, although suitable for conventional database applications such as banking and airline reservation systems, does not provide much flexibility and high performance when used for complex applications such as object oriented systems, long-lived transactions, or distributed systems. Nested transactions have been proposed to overcome the limitations of flat transaction model. Nested transactions extend the notion that transactions are flat entities by allowing a transaction to invoke atomic transactions as well as atomic operations. They provide safe concurrency within transaction, allow potential internal parallelism to be exploited and offer an appropriate control structure to support their execution. In this paper we describe distributed database system and their transaction process. In this paper we also describe advance-nested transactions where the transactions from one system interact with the transactions from another system. Such nested transactions can expect to become more important with the introduction of network operating systems and heterogeneous distributed database systems. Finally, we will study about concurrency issue of nested transaction with respect to distributed database.

Keyword: Distributed database, database, distributed processing, transaction, transaction manager, nested transaction, flat transaction, atomicity, consistency, isolation, durability, subtransaction.

Introduction

A distributed transaction[1] includes one or more statements that reference/modifies data on two or more distinct sites of distributed database. Having discussed distributed database and feature of distributed database, now we are ready to discuss distributed transaction and problem related to distributed transaction.

In this chapter we discuss distributed database transaction and concurrency related problem arise due to data distribution and replication. We also discuss transaction process model of distributed database and nature of transaction. Distributed database transaction: Transaction Management[2] deals with the problems of keeping the database in a consistent state even when concurrent accesses and failures occur.

A transaction consists of a series of operations performed on a database. The important issue in transaction management is that if a database was in a consistent state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed. This should be done irrespective of the fact that transactions were successfully executed simultaneously or there were failures during the execution. Thus, a transaction is a unit of consistency and reliability. The properties of transactions will be discussed later in the properties section. Each transaction has to terminate. The outcome of the termination depends on the success or failure of the transaction. When a transaction starts executing, it may terminate with one of two possibilities:

The transaction aborts if a failure occurred during its execution. The transaction commits if it was completed successfully.A distributed transaction[2] is an operations bundle, in which two or more network hosts are involved. Usually, hosts provide transactional resources, while the transaction manager is responsible for creating and managing a global transaction that encompasses all operations against such resources. Distributed transactions, as any other transactions, must have all four ACID properties, where atomicity guarantees all-or-nothing outcomes for the unit of work

A distributed transaction processing system is a collection of sites or nodes that are connected by communication networks.

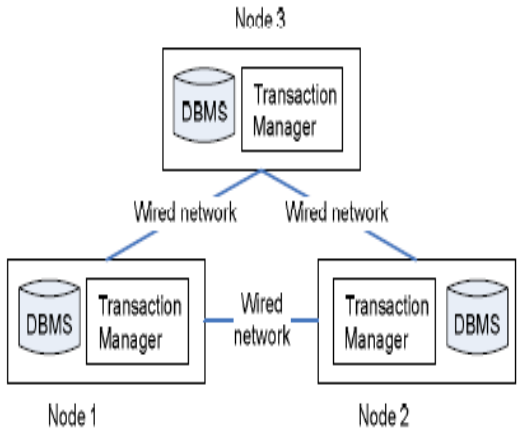

Figure-1
Distributed database

The communication networks are usually reliable and high speed wired networks, like LANs or WANs. At each node in a distributed system, there is a local database management system and a local transaction processing system (TPS) that operates semi-independently and semi-autonomously. An execution of a transaction in a distributed database system may have to spread to be processed at many sites. The transaction managers at different sites in a distributed transaction system cooperate for managing the transaction execution processes.

Distributed Transaction-Processing Model: We consider a distributed database management system with a data collection of sites interconnected by a network. Each site runs one or more of the following software modules:

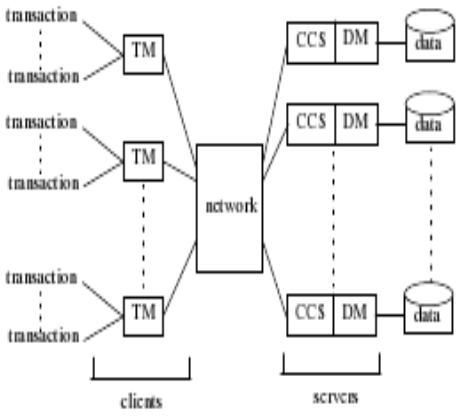

Figure-2
Transaction process

A client runs only the TM module, and a server runs only the DM and CCS modules. TMs supervise transaction interactions between users and the DDBMS, CCSs coordinate transactions, and DMs manage the actual database. The network is assumed to be perfectly reliable and point-to-point FIFO. Figure shows the system architecture. The database is a collection of data items or objects, and each object is managed by a single DM. Users interact with the DDBMS by executing transactions, which are on-line queries or application programs. Transactions communicate with TMs, Tms communicate with CCSs and DMs, and DMs manage data. In order to execute a transaction, a client issues read, predeclare, write, commit, lock-release and abort operations. A server responds with read-response and lock-set operations.

Transactions communicate[2] with TMs, TMs communicate with Dms, and DMs manage the data. TMs supervise transactions. A single TM, meaning that the transaction issues all of its database operations to that TM, supervises each transaction executed in the DDBMS. The TM manages any distributed computation that is needed to execute the transaction. Four operations are defined at the transaction-TM interface.

READ(X): returns the value of X (a logical data item) in the current logical database state. WRITE(X, new-value): creates a new logical database state in which X has then specified new value. BEGIN and END operations to bracket transaction executions. DMs manage the stored database, functioning as backend database processors. In response to commands from transactions, TMs issue commands to DMs specifying stored data items to be read or written.

Category of distributed transaction: Transactions in a distributed system can be categorized into two classes: Local transactions are submitted directly to local transaction managers. Local transactions only access data at one database system at one site, and are managed by the local transaction manager. On the other hand, global transactions are submitted via the global transaction manager. A global transaction can be decomposed into a set of sub-transactions;

ACID property of transaction: The concept of a database transaction (or atomic transaction) has evolved in order to enable both a well-understood database system behavior in a faulty environment where crashes can happen any time, and recovery from a crash to a well understood database state. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems. Each transaction[5] has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands). Every database transaction obeys the following rules (by support in the database system; i.e., a database system is designed to guarantee them for the transactions it runs):

Atomicity: Either the effects of all or none of its operations remain ("all or nothing" semantics) when a transaction is completed (committed or aborted respectively). In other words, to the outside world a committed transaction appears (by its effects on the database) to be indivisible, atomic, and an aborted transaction does not leave effects on the database at all, as if never existed.

Consistency: Every transaction must leave the database in a consistent (correct) state, i.e., maintain the predetermined integrity rules of the database (constraints upon and among the database's objects). A transaction must transform a database from one consistent state to another consistent state (however, it is the responsibility of the transaction's programmer to make sure that the transaction itself is correct, i.e., performs correctly what it intends to perform (from the application's point of view) while the predefined integrity rules are enforced by the DBMS). Thus since a database can be normally changed only by transactions, all the database's states are consistent. An aborted transaction does not change the database state it has started from, as if it never existed (atomicity above).

Isolation: Transactions cannot interfere with each other (as an end result of their executions). Moreover, usually (depending on concurrency control method) the effects of an incomplete transaction are not even visible to another transaction. Providing isolation is the main goal of concurrency control.

Durability: Effects of successful (committed) transactions must persist through crashes (typically by recording the transaction's effects and its commit event in a non-volatile memory).

The concept of atomic transaction has been extended during the years to what has become a Business transaction, which actually implement types of Workflow and are not atomic. However also such enhanced transactions typically utilize atomic transactions as components.

Type of distributed transaction: By structure, distributed transaction is dividing into two types. A flat transaction, FT, is an operation, performed on a database, which may consist of several simple actions. From the client's point of view the operation must be executed indivisibly. Main disadvantage with FTs: If one action fails the whole transaction must abort. A nested transaction[3] occurs when a new transaction is started by an instruction that is already inside an existing transaction. Issues related to distributed transaction: There are a number of issues or problems, which are peculiar to a distributed database and these, require novel solutions. These include the following:

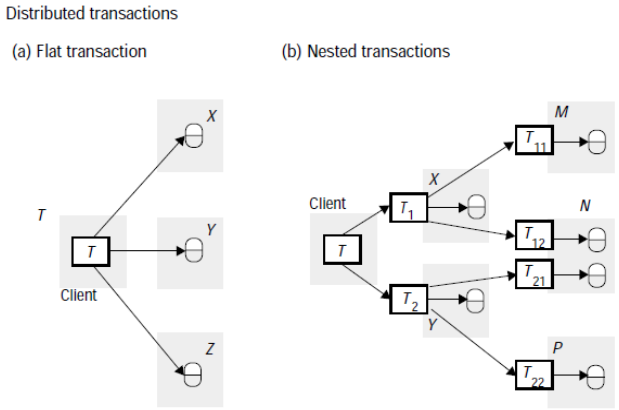

Figure-3
Type of transitions

Distributed query optimisation: In a distributed database the optimisation of queries by the DBMS itself is critical to the efficient performance of the overall system. Query optimisation must take into account the extra communication costs of moving data from site to site, but can use whatever replicated copies of data are closest, to execute a query. Thus it is a more complex operation than query optimisation in centralised databases.

Distributed update propagation: Update propagation in a distributed database is problematic because of the fact that there may be more than one copy of a piece of data because of replication, and data may be split up because of partitioning. Any updates to data performed by any user must be propagated to all copies throughout the database. The use of snapshots is one technique for implementing this.

Distributed catalog management: The distributed database catalog entries must specify site(s) at which data is being stored in addition to data in a system catalog in a centralised DBMS. Because of data partitioning and replication, this extra information is needed. There are a number of approaches to implementing a distributed database catalog. Centralised- Keep one master copy of the catalog, Fully replicated - Keep one copy of the catalog at each site, Partitioned - Partition and replicate the catalog as usage patterns demand, Centralised/partitioned- Combination of the above.

Distributed concurrency control: Concurrency Control[4] in distributed databases can be done in several ways. Locking and timestamping are two techniques, which can be used, but timestamping is generally preferred. The problems of concurrency control in a distributed DBMS are more severe than in a centralised DBMS because of the fact that data may be replicated and partitioned. If a user wants unique access to a piece of data, for example to perform an update or a read, the DBMS must be able to guarantee unique access to that

data, which is difficult if there are copies throughout the sites in the distributed database.

Transaction Concurrency: If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency[4] exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:

The lost update problem: A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.
The dirty read problem: Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.

The incorrect summary problem: While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Conclusion

Transaction management is an old concept in distributed data base management systems (DDBMS) research. In this paper, we have reviewed the basic concepts of advanced transaction management. We discuss the basic concept of nested transaction in distributed database systems, and also discussed the advantage, property and operations of nested transaction in distributed environments. It is really important for database to have the ACID properties to perform.
We are in the process of investigating schemes by which the performance of high security level transactions can be improved without compromising with the security. Further we are looking to secure real time distributed systems by which the performance of high security level transactions can be improved without compromising the security.

References

1. Tamer M. Ozsu and Patrick Valduriez. Principles of Distributed Database Systems, Second Edition. Prentice-Hall **(1999)**

2. Distributed Transaction Processing on an Ordering Network By Rashmi Srinivasa, Craig Williams, Paul F. Reynolds **(2002)**

3. Moss E.B., Nested transactions: An approach to reliable distributed computing, Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA **(1981)**

4. Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley **(1987)**

5. Bernstein P. and Goodman N., Concurrency Control in Distributed Database Systems, ACM Computing Surveys 13/2, **(1981)**

6. Kaur Manpreet, Transaction Processing in Distributed Databases. Amritsar College of Engg. and Tech,Amritsar Sheetlani Jitendra, Jangde Manoj, Concurrency Control in Distributed Transaction process., Prabandhan and Taqniki **04**, 271-274 **(2010)**

7. Sheetlani Jitendra, Jangde Manoj. Timely Computing base Transaction in DBMS., Shodh **04**, 5 **(2010)**

8. Sheetlani Jitendra, Jangde Manoj Concept and Technique of Transaction process of Distributed Database Management system., advancement in computational technique and application **01**, 190-194 **(2011)**

9. Sheetlani Jitendra and Jangde Manoj, Nested Transaction Management in distributed database, CGTTM, LNCT Indore, **(2011)**

10. Elmasri Navathe, Database Concepts By Pearson Education, **(2011)**

11. Colloly., Data base Concepts By Pearson Education, **(2011)**

12. Coronel Rob Introduction to Database Concepts, **(2011)**

13. Sheetlani Jitendra, Gupta Dhiraj, World of DBMS, **(2009)**