JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

CONCURRENCY INSIGHT: OPTIMISTIC PARADIGM ON LOST UPDATE TRANSACTIONS IN A DISTRIBUTED DATABASE

ONUNGWE, H. O[1] ., EKE, O. B [2]. and Alabi O.A [3].

Department of Computer Science

Faculty of Science

University of Port Harcourt

ABSTRACT

Concurrency is a situation where more than one processes are running at the same time. In such scenarios, there are tendencies that there will be conflict in reading and writing of data items Concurrency is associated with four major problems (P1 – P4): Unrepeated Read (P1), Inconsistent Analysis (P2), Lost Update (P3) and Phantom Read (P4). A Concurrency Control techniques need to be put in place to ensure that data items are serialize in a structured pattern to avoid an adverse compromise on data integrity and consistency. Basically, there are two major techniques of controlling Concurrency on Databases; Pessimistic and Optimistic Control techniques. This paper is aimed at handling the lost update problem (P3) using the Optimistic technique. We used Optimistic Locking System to insert locks on the data items. The locks were only released when the first transaction had reached it's commit stage in order to prevent data duplication. Our Optimistic Locking approach used the locks only at the beginning of the transaction. Optimistic locking was proven to be the best choice in a situation where if the odds of conflict are very low (many records and relatively few users, or very few updates and mostly "read" operations),

KEYWORDS: Optimistic, Lost Update, Concurrency, Concurrency Control, Serialisation, Database, Locks etc

1.0 INTRODUCTION

Multiple processes need to be serialize in a manner that there should not be conflict issues of data integrity. Concurrency is a property of a system representing the fact that multiple activities are executed at the same time. In conventional database systems, concurrency control ensures the correct executions for concurrent transactions T1, ..., Tn.

Concurrency Control in distributed databases can be done in several ways. Locking and time stamping are two techniques, which can be used. Again, 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

Logically, Concurrency Control is an illusion that processes, transactions etc are running the same time without any form of conflict. Concurrency is a pertinent tool in modern programming, especially with the rise of multi-core architectures and the increasing prevalence of distributed systems. And like any other tool, it is important to understand how and when to use it to have efficiency. Morgan (2013) asserts that in common thread, concurrency, applied correctly, can improve the performance of a program but the correct application may not be readily apparent. He opined that Concurrency is conceptually difficult, and requires a different approach than sequential programming. The purpose of his case study is to investigate the advantages to executing multiple tasks in parallel. To this end, the concurrency in his case study was introduced at a higher level of the program than in previous case studies. Instead of using concurrency to calculate the image filters over multiple pixels within an image simultaneously, concurrency was used to apply the sequence of image filters to multiple images at once. One limitation of his work is that he did not explore the effects of varying the number of available processors. He presented results for the non concurrent (i. e: single core) case, as well as multiple concurrent implementations that execute on four-cores. Extending the case studies to utilize larger numbers of cores would be an interesting avenue of investigation, though would require additional hardware.

1.2 Problems of Concurrency:

Some problems may occur in multi-user environment when concurrent access to database is allowed. These problems may cause data stored in the multi-user DBMS to be damaged or destroyed. Transactions running concurrently may interfere with each other, causing various problems. The

problems associated with concurrency are clearly depicted in Figure 1.1.

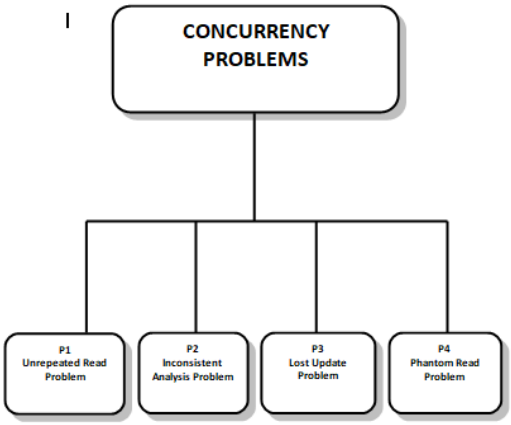

Figure 1.1: **Concurrency Problems in Transaction**

1.3 Security Concerns in Distributed Database

Security means protection of information and information system from unauthorized access, modification and misuse of information. The purpose of distributed database security is to deal with protecting data from people or, software having malicious intension. Distributed system has four main security components, security authentication, authorization, Encryption, and multi level access control.

Authentication: - usually authentication is initialised by password. A user must provide the correct password when establishing a connection to prevent unauthorized use of the database. Passwords are assigned when user are created newly.

Authorization: - The purpose of authorization is to supply one secured access point enabling the users to link up to the network once and allow them access to authorized resources. Encryption: - It is the technique of encoding data that only authorized users can understand it. A number of industry standard encryption algorithms are useful for the encryption and decryption of data on the server.

Multi-level access control: - In a multi-level access system, user are limited from having complete data access. Policies restricting user access to certain data parts may result from secrecy requirement or, they may result from loyalty to the principal of least privileged (a user only has access to relevant information). Access policies for multi-level system are typically to as either open or, closed. In an open system, all the data is considered unclassified unless access to a particular data element is expressly prohibited.

Mohamad (2013), worked on Security Aspects of Distributed Database. He listed Fragmentation, Allocation, and Replication as the three main issues of Distributed System Designs. He went

further to review all the security issues and design features of databases in a general form and distributed databases in particular. He equally indicated some related security and design issues including multilevel security in distributed database systems. The security aspects lead him to investigate the distributed data and centralized control, and some retrieval problems in distributed databases of accessing control and integrity. Moreover, he described the most common mechanisms of discretionary security and stated the emerging security used in distributed system tools.

1.4 Distributed Systems

A distributed system is a network that consists of autonomous computers that are connected using a distribution middleware (Mahnaz Nasseri et al. 2017). Such autonomous computers are what is described as capable of collaborating on a task, examples of distributed systems are: telephone network and cellular network, computer network such as Internet, ATM machine, mobile computers etc. Distributed Systems are loosely coupled because there is no share memory. A Distributed System is a collection of various sites, distributed over a computer network (Ladwig et al. 2011). A distributed system consists of hardware and software components located in a network of computers that communicate and coordinate their actions only by passing messages. . (Jussi 2008), defines a distributed system as a collection of independent computers that appears to its users as a single coherent system or as a single system. A Distributed Transaction permits a given transaction to reference several different sites, although each single request can reference only one remote site. A Distributed Request allows an individual to reference data from several remote sites, each request can access data from more than one site, whereas a transaction can access several sites. The ability to execute a distributed request provides fully distributed database processing capabilities because the database is partitioned into several fragments.

2.0 LITERATURES REVIEW

Vlad (2019), asserts that Lost Updates anomaly is a lesser known phenomena of consistency. In the real sense, it is not less because the challenges comes with lots of conflicts when multiple transactions try to update the same database rows since it is still needful that there should be consistency in the transaction. He opined that a

Lost Update occurs when a user overrides the current Database state without realizing that someone else changed it between the moment of data loading and the moment the update occurs. In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction. It occurs when two concurrent transactions try to read and update the same data. Lost update problem occurs when multiple transactions try to touch the same rows in the database. The lost update problem can occur if an application reads a value, modifies it and writes it back to the database. This can have serious implications depending on the type of application. We shall use the Optimistic Locking System to solve the problem of overriding in lost update problem.

Qilong et.al (2008)worked on "A Concurrency Control Algorithm Access to Temporal Data in Real-time Database Systems". Their proposed method was designed in such a way that checking algorithm is carried out to guarantee the use of validated data that fit with the transaction scheduling process. The checking process ensures that all temporal data in the read set of a transaction remain valid during all its execution time which will guarantee the temporal consistency of this transaction. Afterward, the key factor concurrency control algorithm is adjusting validation rules during validation phase, which schedules the priority transactions that are near to complete by asserting validation factor. The validation algorithm calculates the validation factor of the validating transaction, which is a variable calculated from the current time, the start time and the deadline time of the transaction, and calculates the temporal deferrable time of the transaction.

Toby Weston (2012) worked on Optimistic and Pessimistic Concurrency Control with Shared Memory Models DRAFT-14092012 A look at modern concurrency control mechanisms in Java. He opined that Concurrent programming has always been difficult, mostly because of the shared memory model and traditional approaches guarding it. His paper aims to explore the problems, describing characteristics of concurrency control in shared memory systems, comparing optimistic and pessimistic approaches using a real world example and comment on the current state and appropriateness of technology choices. Distributed models avoid contention as they don't actually share memory, each process

works on its own local heap. Techniques such as the actor model or distributed message passing effectively simulate a distributed model and are out of scope for his work. One major goal of his work is to present alternative implementations of a common concurrency problems; typical optimistic lock based synchronization solutions and optimistic, software transactional memory based solution.

Walid et.al (2012). Worked on "An Optimistic Concurrency Control Approach Applied to Temporal Data in Real-time Database Systems. They propose an optimistic lock concurrency control method based on Similarity, Importance of transaction and Dynamic Adjustment or Serialization Order called OCC-SIDASO. This method uses dynamic adjustment of serialization order, operation similarity and the transaction importance, for maintaining transaction timeliness level, minimizing transactions wasted restart, and guaranteeing temporal consistency of data and transactions. Our own work equally used Optimistic Locks approach to solve the Lost Update problem of Concurrency Control issues. Optimistic concurrency control (OCC), also known as optimistic locking, is a concurrency control method applied to transactional systems such as relational database management systems and software transactional memory. OCC assumes that multiple transactions can frequently complete without interfering with each other. Optimistic concurrency control works by ensuring that the record being updated or deleted has the same values as it did when the updating or deleting process started.

3.0 MATERIALS AND METHODS

When 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. We introduced Locks to each of the various transactions, read all the data, and then release the Locks in order to allow the first transaction to commit before reading the values of the second transaction.

3.1 Overcoming the Lost Update Problem

Possible Solutions to overcoming the Lost Update Problem:

1. Ignore It. The simplest technique is to ignore it, hoping it will never happen; or if it does happen, that there will not be a terrible outcome.
2. Locking. Another popular technique for preventing lost update problems is to use locking techniques.
3. Read Before Write.
4. Timestamping.

Our work is more concerned with Locking the database to overcome the lost update problem. We proposed a method for preventing lost updates by using optimistic concurrency control to perform what is called optimistic locking on the data. Locking (e.g., Two-phase locking - 2PL) - Controlling access to data by locks assigned to the data. Access of a transaction to a data item (database object) locked by another transaction may be blocked (depending on lock type and access operation type) until lock release. Whenever a data item is being accessed by a transaction, it must not be modified by any other transaction. In order to ensure this, a transaction needs to acquire a *lock* on the required data items. A **lock** is a variable associated with each data item that indicates whether a read or write operation can be applied to the data item. In addition, it synchronizes the concurrent access of the data item. Acquiring the lock by modifying its value is called **locking**. It controls the concurrent access and manipulation of the locked data item by other transactions and hence, maintains the consistency and integrity of the database. Database systems mainly use two modes of locking, namely, *exclusive* **locks and** *shared locks*. One of the main techniques used to control concurrency execution of transactions (that is, to provide serialisable execution of transactions) is based on the concept of locking data items. A lock is a variable associate with a data item in the database and describes the status of that data item with respect to possible operations that can be applied to the item. Generally speaking, there is one lock for each data item in the database. The overall purpose of locking is to obtain maximum concurrency and minimum delay in processing transactions.

Optimistic concurrency control typically uses three phases in order to help to ensure that data is

not lost. Figure 1.2 depicts the phases of Optimistic Concurrency Control measures.

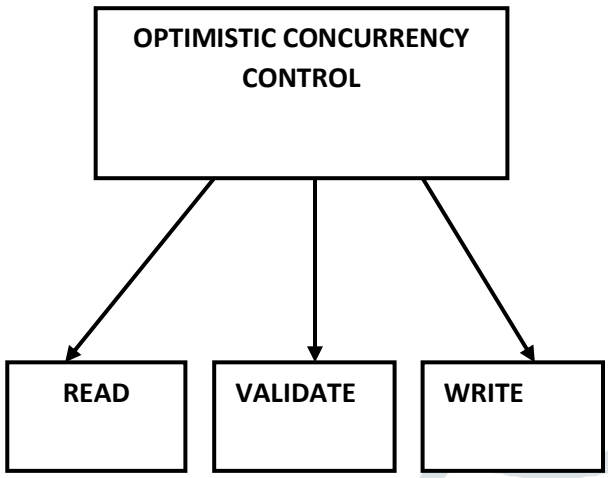

Figure 1.2 **Phases of Optimistic Concurrency Control**

Read Phase: Transaction can read values of committed data items from the database. However, updates are applied only to local copies of the data items kept in the transaction workspace. Validation Phase: Checking is performed to ensure serializability will not be violated if the transaction updates are applied to the database.

Write Phase: If the validation phase is successful, the transaction updates are applied to the database otherwise, the updates are discarded and the transaction is restarted.

When you use Optimistic Concurrency Control, you do not find out that there is a conflict until just before you write the updated data. In Pessimistic approach, you find out that there is a conflict as soon as you try to read the data. We marked the transaction's beginning, by reading the database values, and tentatively write changes. We further checked whether other transactions have modified data that this transaction has used (read or written). In that way, we were able to place a Lock on the transactions. The Locks are release upon when the initial transaction has reached it's Commit stage. The essence of the Lock is to ensure that data items are not duplicated in the same transaction.

4.0 CONCLUSION

With optimistic locking, we do not check for conflicts except at the time that we write updated data to disk. If user1 updates a record and user2 only wants to read it, then user2 simply reads whatever data is on the disk and then proceeds, without checking whether the data is locked. User2 might see slightly out-of-date information if user1 has read the data and updated it but has not yet "committed" the transaction. Data is not always available to users because locking is used; access is improved since deadlocks no longer apply. Again, however, users run the risk that their changes will be thrown out if those changes conflict with another user's changes. Thus, for example, user1 might put an exclusive lock on a record and update it. When the record is updated,

its version number changes. User2, who is using a read-only transaction, can read the previous version of the record even though the record has an exclusive lock on it. Pessimistic locking allows you an option that optimistic locking does not offer. Pessimistic locks fail ”immediately” – that is, if you try to get an exclusive lock on a record and another user already has a lock (shared or exclusive) on that record, then you will be told that you cannot get a lock. Wait mechanism applies only to pessimistic locking, not to optimistic concurrency control. There is no such thing as ”waiting for an optimistic lock”. If someone else changed the data since the time that you read it, no amount of waiting will prevent a conflict that has already occurred. Neither pessimistic nor optimistic concurrency control is ”right” or ”wrong”. When properly implemented, both approaches ensure that your data is properly updated. In most scenarios, optimistic concurrency control is more efficient and offers higher performance, but in some scenarios pessimistic locking is more appropriate. In situations where there are a lot of updates and relatively high chances of users trying to update data at the same time, you probably want to use pessimistic locking. If the odds of conflict are very low (many records and relatively few users, or very few updates and mostly ”read” operations), then optimistic concurrency control is usually the best choice. The decision will also be affected by how many transactions that are updated at a time. If two transactions attempt to modify the same record, the second transaction will wait for the first one to either commit or rollback. If the first transaction commits, then the second one must be aborted to prevent lost update. Our approach prevents users and applications from editing stale data. It notifies users of any locking violation immediately. Our proposed system used locks to control concurrent access to data, achieving two important database goals: Consistency and integrity.

REFERENCES

Moudani, W., Khoury, N. and Hussein, M. (2012) “An Optimistic Concurrency Control Approach Applied to Temporal Data in Real-time Database Systems. *WSEAS TRANSACTIONS on COMPUTERS*, (12).

Ladwig, G., & Tran, T. (2010, November). Linked data query processing strategies. In *International Semantic Web Conference* (pp. 453-469). Springer, Berlin, Heidelberg.

Joshi, M., & Srivastava, P. R. (2013). Query optimization: an intelligent hybrid approach using cuckoo and tabu search. *International Journal of Intelligent Information Technologies (IJIIT)*, *9*(1), 40-55.

H. Qilong, P. Haiwei, Y. Guisheng,(2008). A Concurrency Control Algorithm Access to Temporal Data in Real-time Database Systems, International Multi-symposiums on Computer and Computational Sciences,

Toby Weston (2012). Optimistic and Pessimistic Concurrency Control with Shared Memory Models DRAFT-14092012 A look at modern concurrency control mechanisms in Java