iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 7 **Issue:** III **Month of publication:** March 2019

DOI: http://doi.org/10.22214/ijraset.2019.3446

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue III, Mar 2019- Available at www.ijraset.com

Comparative Analysis of Distributed Concurrency Control Protocols

S. Kiran[1], I. Pavan Raju[2], Poonam Sharma[3]
[1, 2]PG student, [3]Assistant professor, Department of computer science, Amity University-Haryana, 122413, INDIA

Abstract: ***Concurrency control algorithms have traditionally been based on locking and timestamp ordering mechanisms. Recently optimistic schemes like distributed, multi-version, optimistic concurrency control scheme have been propose which is particularly advantageous in a query-dominant environment. In this paper we do a comparative analysis of the various concurrency control protocols used for distributed databases with their advantages and disadvantages.***
Keywords: ***Database systems Concurrency Optimistic protocols Distributed algorithms***

I. INTRODUCTION

A database in which storage devices are attached to a different processors is known as distributed database. It can be stored in multiple computers that are either located in same location or scattered over network of interconnected computers.
System administrators can distribute collections of data across multiple physical locations. A distributed database can reside on organized network servers or decentralized independent computers on the Internet, on corporate intranets or extranets, or on other organization networks. Besides distributed database storage replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies implementations can and do depend on the needs of the business and the sensitivity/confidentiality of the data stored in the database, and the price the business is willing to spend on ensuring data security, consistency and integrity.[1]

A. Motivation Behind Distributed Databases

According to Claudia Leopold, there motivations for implementing a distributed system instead of simply utilizing the computing resources of a standard computer are

1) The Price to Performance ratio for the system is more favourable for a distributed system
2) Some applications are solved most easily using the means of distributed computing
3) Distributed computing allows the sharing of resources - both hardware and software
4) Distributed Computing allows the system to grow incrementally as computers are added one by one.
5) By integrating the computers into a distributed system, the excess computing power can be made available to other users or applications." [2]

B. Transactions In Distributed Data Bases

In a distributed transaction two or more network hosts are involved. Usually, we have coordinator and transaction manager. The coordinator is used to provide transactional resources, while the transaction manager creates and manages a global transaction that encompasses all operations against such resources. Distributed transactions have four main properties as any other transactions, these are known to be ACID properties which are atomicity, consistency, isolation, durability. Database are common transactional resources and, often, transactions span a couple of such databases.
In this paper, we will see different transactional protocols for the distributed systems and have a comparative analysis among them.

C. Architecture of Distributed Data Bases

Architecture of the data bases involves a single tier, 2-tier and a 3 tier where in a single tier, user directly programs and changes the databases. In a 2 tier, the database uses an application through which the databases are accessed by the user.
Based on the complexity of the users, the 3 tier architecture can be separated in to different tiers in order to use the present data in the database.

1) *Tier1:* It is the database or the data tier where the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue III, Mar 2019- Available at www.ijraset.com

2) *Tier 2:* It is the application or middle tier which reside the application server and the programs that access the database. This application tier presents an abstracted view of the database. Users are unaware of the database which exists beyond the application.
3) *Tier 3:* This tier is known to be user or the presentation tier where users operations are done here. In this tier, multiple views of the database can be provided by the application. Views that are generated by applications that reside in the application tier.[3]

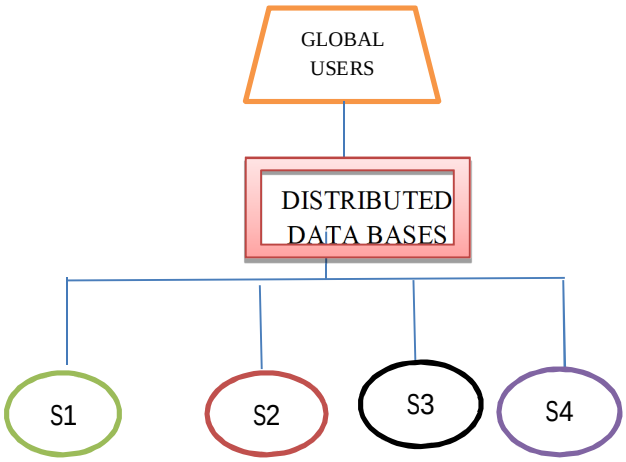


Fig: 1: schematic diagram for distributed data bases

Failures in the distributed transactions occurs due to three main reasons that is due to failure of site, loss of messages and failure of communication link. In order to recover the data we used 2 phase commit protocols where in the first step obtaining decision is made and the in the second step recording of the decision is made and once all the child sites complete the transaction then parent site will commit the transaction further.

To avoid the problems and infinity waiting time period in 2 phase commit protocol, Shared-nothing architecture has been widely used in distributed databases to achieve good scalability. While it offers superior performance for local transactions, the overhead of processing distributed transactions can degrade the system performance significantly. Here, each node is independent and self-sufficient and doesn't share any memory location. It has self-healing capabilities and eliminates single point failure and aims towards non-disruptive up gradations.

It is important to control the concurrency of the transactions in multiple programming environment where different multiple transactions can be executed simultaneously. There are different concurrency control protocols to ensure the atomicity, isolation, and serializability of the concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

a) Lock based protocols
b) Time stamp based protocols

II. CONCURRENCY CONTROL PROTOCOLS IN DISTRIBUTED DATA BASES

A. Timestamp Ordering Protocol

The majorly used concurrency protocols is the timestamp based protocol. The protocol can use either system time or logical counter.

At the time of execution lock based protocols manage the order between the conflicting pairs among transactions, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the the age of the transaction determines its ordering. A transaction X initiated at 0005 clock time and would be older than all other transactions that come after it. Another transaction Y entering the system at 0009 is four seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp.[4]

1) The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations.
a) The timestamp of transaction T is denoted as TS (T).
b) Read time-stamp of data-item X is denoted by RTS(X).
c) Write time-stamp of data-item X is denoted by WTS(X).

2) Timestamp ordering protocol works as follows

a) *If a transaction Ti issues a read(X) Operation:* Operations will be rejected if the time stamp of transaction T is less than the write time stamp of X and will be executed otherwise. All data-item timestamps updated.

b) *If a transaction Ti issues a write(X) Operation:* Operations will be rejected if time stamp of transaction T is less than either the read time stamp of X or write time stamp of X. Otherwise, operation executed.

In a system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

To prevent any deadlock situation in the system, there are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation. They are wait-die scheme and wound-wait scheme.

A. *Multi Version Of Time Stamp Ordering Protocol*

The data item Q has sequence of versions Q_1, Q2, Q3…Q_N. Each version Q_k has three fields. First one is content which stores the values, other is the WTS which stores the time stamp for write versions of Q, and the last is the RTS which stores the read time stamp for read versions of Q. when transaction T creates a new version of Q then WQ and RQ are initialized to TS(T).[5]

B. *Time Stamp Based Optimistic Concurrency Control Protocol (OCCP)*

To overcome the problems of lock based protocols we use occp which has three phases

1) Read: reads from database and writes them onto private space

2) Validation: transaction is about to commit and here occp checks whether there are any conflicts are not

3) Write: if there are no conflicts then the private space is copied into data base avoiding the problems of lock based protocols.

During execution, each transaction will either read or write, so the steps are:

a) Is validating transactions in serial problem with conflicting transactions.

b) Is validating transactions write-write.

c) Is validating transactions read-write.

Two transactions T1 & T2

If IT (T1) <IT (T2) IS TRUE then transactions is in serial problem with conflicting transactions

If write set (T2) and write set (T1) doesn't have anything in common then transactions have write –write operation

If read set(T2) and write set (T1) have common element 'e' then each element of 'e' will have less time instance to read or write than the time instance for the whole transaction.

C. *Optimistic Concurrency Control Protocol with Locking Systems (OCCL)*

To overcome the problems of optimistic concurrency control protocols we have locking systems attached to them where each transaction T has both read, write phase and also has read and validation lock.

In the first condition it ensures the locking mechanism and in the second condition either serial or parallel validation takes places

1) *In serial Validation*

a) Read phase: T1 works on own local copy and sets R- lock in its RS (T1)

b) Validation phase: T2 checks its WS (T2) against RS (T1) by setting the v locks

2) In parallel validation:

a) Read phase: both RS and WS are needed to set R lock

b) Validation phase: if R lock exists and T2 is not the only one lock that exists the validation will take place. [6]

D. *Sundial Concurrency Protocol*

Distributed transactions in OCCL has two major limiting factors. First one is the distributed transactions suffer from latency.

Second, due to the increase in latency the likelihood of the contention among distributed transactions, leads to lower performances and high abort rates.

Sundial, which is an optimistic concurrency control protocol that addresses these two limitations.

First, transaction abort rate can be reduced.

Second, to reduce the overhead of remote data accesses, Sundial allows the database to cache remote data in a server's local main memory and maintains cache coherence.

Sundial outperforms the next-best protocol by up to 57% under high contention.[7]

III. COMPARITIVE ANALYSIS

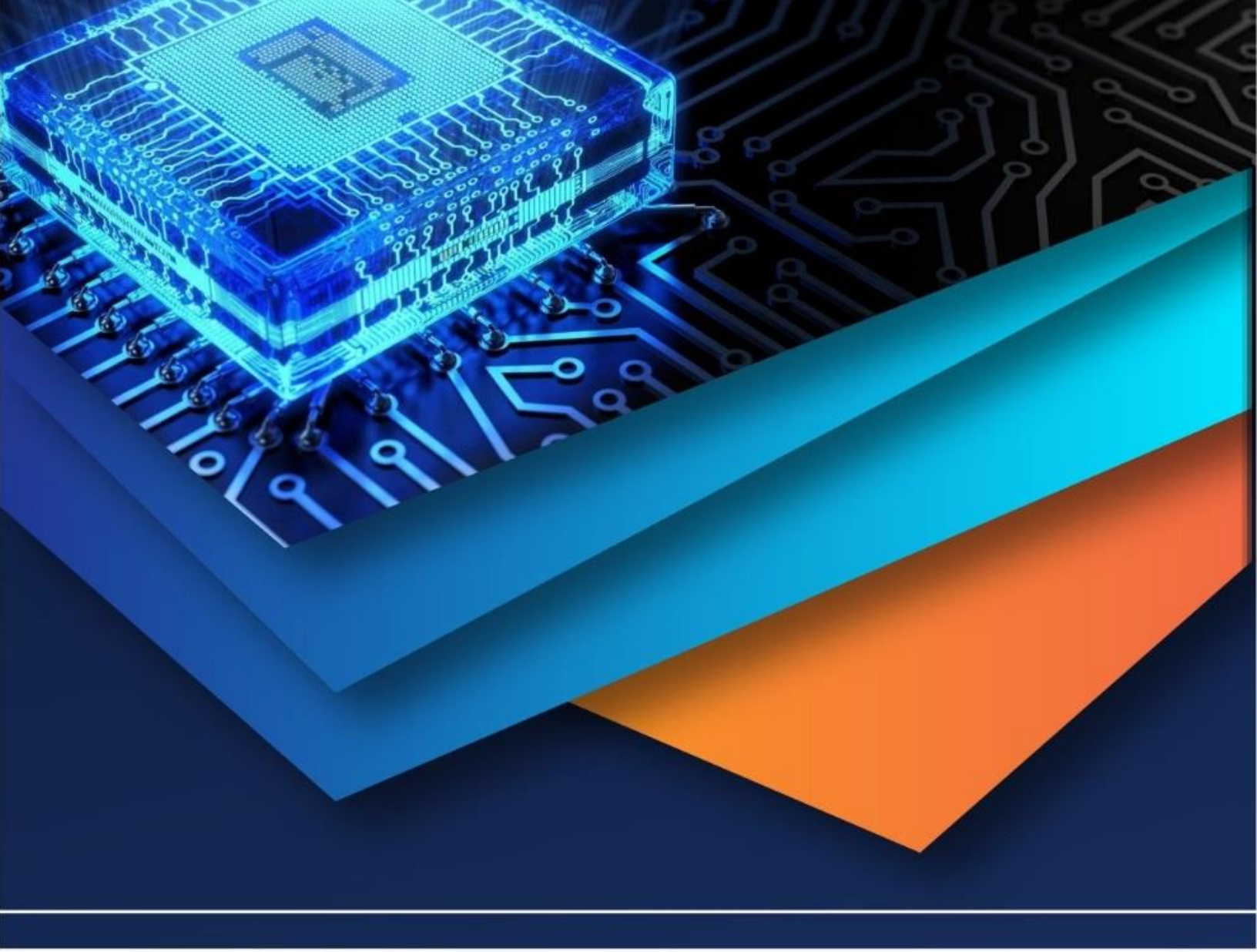
Problem	Type of protocol used	Solution	Advantages	Disadvantages
To achieve a non 2 phase protocol by have independent memory for each node[8]	Shared nothing architecture	Leap is introduced where the processing is done across multiple nodes where distributed transactions can be converted into local transactions	-based on leap, oltp system is developed. -self healing capabilities -eliminates single point failures -non disruptive upgrade	Takes much time to respond
To ensure serializability among transactions.[9]	Time stamp based protocols	Transaction is designed a way that all write operations are done at end All write operations are performed from atomic action No transactions are executed while writing Transactions which are aborted are restarted	The timestamp based protocol ensures freedom from deadlock, since no transaction ever waits	Scheduling is not recoverable in transactions and leads to series of roll backs
In above protocols locks are acquired but not released and if locks are released the new locks can't be acquired.[10]	Optimistic concurrency control protocol	Read validation and works are done in order to avoid conflicts.	Provides scalability Less data base connections are used No client and server calls since no locks are applied. No chain blocking. No delay.	Longer transactions Longer time if restarts Synchronization is done repeatedly which is called starvation
Once the conflicting transactions are aborted using occp, then time to be waited for restarting is unknown[11]	Occp is introduced with locking systems i.e. Oocl	Has both locking mechanism as well as validation of transactions is done. In validation of transactions both serial as well as parallel validation is done.	Useful for long running transactions.	Concurrency within the transactions. Implementation overhead.
High latency High likelihood of contention among the distributed systems Implementation overhead[7]	Sundial protocol is used	Determines the logical order among transactions at runtime, based on their data access patterns. Sundial allows the database to cache remote data in a server's local main memory and maintains cache coherence	Improving concurrency Having light weight overhead for implementation.	Needs extra storage to maintain logical leases of the tuples Transactions are difficult for partionable loads.

IV. CONCLUSIONS

First, Sundial requires extra storage to maintain the logical leases for each tuple. Although this storage overhead is negligible for large tuples, it can be significant for small tuples. One way to reduce this overhead is for the DBMS to maintain the tuples logical leases in a separate lease table. The DBMS maintains leases in this table only for tuples that are actively accessed. The second issue is that Sundial may not deliver the best performance for partitionable workloads. Sundial does not assume that the workload can be partitioned and thus does not have special optimizations for partitioning. Systems like H-Store perform better in this setting. Our experiments show that if each transaction only accesses its local partition, Sundial performs 3.8× worse than a protocol optimized for partitioning. But our protocol handles distributed (i.e., multi-partition) transactions better than the H-Store approaches. Finally, the caching mechanism in Sundial is not as effective if the remote data read by transactions is frequently updated. This means the cached data is often stale and transactions that read cached data may incur extra aborts.

REFERENCES

[1] Andrew S. Tanenbaum and Maarten Van Steen–DISTRIBUTED SYSTEMS-121rtac IEEETAYA diets, Second Edition
[2] Claudia Leopold-Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches
[3] M. Tamer Özsu, Patrick Valduriez-Principles of Distributed Database Systems
[4] Sunil Dhondu Mone, "Timestamp-Ordering Protocol for Concurrent Transactions - A Performance Study, National Conference on Emerging Trends in Computer Technology (NCETCT-2015)
[5] Paul Kahoro,Charity Wanjiru,Newton K,.1 July 22, 2015
[6] Jiandong Huang, John A. Stankovic, Krithi Ramamritham, Donald F. Towsley-"Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes"
[7] Xiangyao Yu, Yu Xia, Andrew Pavlo, Daniel Sanchez Larry Rudolph, Srinivas Devadas Journal Proceedings of the VLDB Endowment VLDB Endowment Homepage archive Volume 11 Issue 10, June 2018
[8] Michael Stonebreaker-The Case for Shared Nothing University of California, Berkeley,U.S. Air Force Office of Scientific Research Grant 83-0254 and the Naval Electronics Systems Command Contract N39-82-C-0235.
[9] Concurrency control in distributed database systems Philip a. Bernstein and Nathan Goodman Computer Corporation of America, Cambridge, Massachusetts 02139
[10] A. Chiu; Ben Kao; Kam-yiu Lam-"Comparing two-phase locking and optimistic concurrency control protocols in multiprocessor real-time databases"-10.1109/WPDRTS.1997.637965
[11] Quazi Ehsanul Kabir Mamun ,Hidenori Nakazato-"timestamp Based Optimistic Concurrency Control",IEEE Region 10 Conference.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY