International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-8, Issue-3, May 2020
https://doi.org/10.21276/ijircst.2020.8.3.8
www.ijircst.org

Version Locking Mechanism in Database

Swati

ABSTRACT-The distributed database provides a resource sharing environment, where multiple transactions at different sites coexist in order to access the resources. In this paper we investigate multi version locking protocol in distributed environment. Multi-version database has the potential to significantly increase the amount of concurrency in transaction processing as they can avoid read-write conflict by completing the read requests with older versions of data while the write operation is in progress. These algorithms are particularly effective for long queries, which otherwise cannot finish due to the high probability of conflict with other transactions. We have given an improvement on the algorithm proposed by Jie shao. We have not imposed any restriction on the involvement of all the existing sites to obtain a global version which thereby reduces the burden on each individual site. Our scheme involves the collection of local transaction identifier only from the participating site whether it can be one or many. We have proposed our new algorithm to obtain a correct version for a particular data item during read and write operation.

KEYWORDS- Local transaction identifier; Global consistent snapshot version; Serializability.

I. INTRODUCTION

A distributed database management system is a collection of inter related sites which are connected by a computer network. The data base management system includes a concurrency control protocols that coordinate the execution of multiple transactions [14, 8] which is an essential part of the database system. It is used to handle the concurrent execution of operations by different transactions on the same data item while maintaining the serializability in the schedules and managing the atomicity, consistency, isolation and durability properties of the transactions [4].

The database system must choose either of the two alternatives for handling the updates: (1) either to overwrite the old data with the new data ("update-in-place systems") or (2) to write a new copy of the record with the new data, append it with the old data ("multi-versioned systems") [9,16].

The primary advantage of multi versioned systems is that the transactions can write to a particular record while the read operation on the same record proceed in parallel .As a result read transaction do not block write transaction this is possible because any read operation can read the older versions until the write transaction has committed. [9,2].We can consider history H1 to illustrate the following concept.

H1:$r_1(p,0),w_2(p,4),w_2(q,6),c_2,r_1(q,0),c_1$.

In the given history H1 T1 and T2 are the two transactions (r1, w2) is the read and write operation for transaction T1 and T2 respectively, c1 and c2 is the commit action for the two transactions. The mode of operation for the transaction are T1[$r_1(p,0),r_1(q,0)$ and T2[$w_2(p,4),w_2(q,6)$].The read on q by T1 gives the value 0 but will not give the value of latest write of 6 carried out by transaction T2 which commit before T1.This is possible by storing multiple version of the data item q. If no multiple versions would have being maintained than the transactions T1 would abort. A global consistent snapshot is required to eliminate the problem of data inconsistency in the distributed system so that update can be propagated to all the relevant sites. In this paper we have proposed our distributed multi-version concurrency control (DMVCC), protocol that ensures that the global snapshot version is obtained by consulting only the participating sites, we primarily stress on reducing the burden on all the existing sites in the distributed database. In this paper we have given an improvement on the algorithm proposed by [3].In order to obtain global version of the data we have proposed our algorithm which is refinement over the work carried out so far in distributed environment.

II. ARCHITECTURE OF DISTRIBUTED MULTI-VERSION CONCURRENCY CONTROL

In a distributed database there are three main components: the data records, distributed transaction manager (DTM) and the consistency coordinator [3] as shown in "Fig.1"

Manuscript Received May 2, 2020

Swati, Department of Computer Science, Amity University, Haryana Gurgaon, India (email: sgupta@ggn.amity.edu)

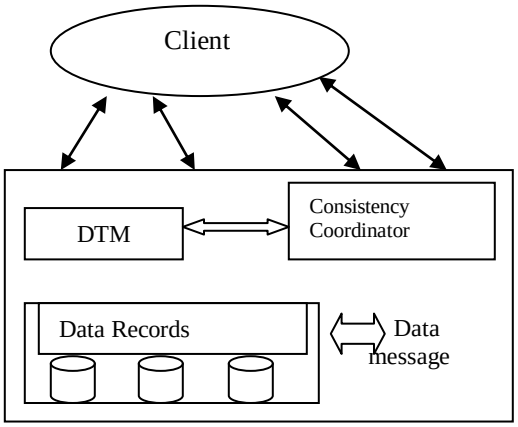

Fig 1: Distributed Database Architecture [3]

Details of each component are as follows:

A. Data records

The data records are the local databases which are being stored at individual local sites and are independent of one another. Whenever a request is made by the client the DTM subdivides the request and assign them to individual local sites. On the verge of commit operation by the transaction the data records generate a snapshot version which is uniquely identified by the local transaction identifier (LTID) and is sent to the consistency coordinator which uses the LTIDs [3] to determine a global consistent snapshot version.

B. Distributed transaction manager

The client initiates the request and contact the DTM which split the transaction into sub-transactions and send them to the corresponding data records present at individual local site. The global copy of the data item is obtained when DTM contact the consistency coordinator [3].The consistent version is a set of local transaction identifier that are collected from the data records which present the version of a particular data item at each site which is being given by the notation

Global consistent snapshot=LTID1 ∪ LTID2 ∪….LTIDn (where LTID1, LTID2…LTIDn are the local transaction identifier generated from the associated sites)

C. Consistency coordinator

The consistency coordinator is used to calculate the global snapshot version of a particular data item. The DTM contact the consistency coordinator in two cases:

Case 1: When a request is made from client side to read a particular data the DTM contact the consistency coordinator

Case 2: When the transaction completes its task at individual site the LTID from the sites are sent to consistency coordinator through DTM in order to obtain global version of the data record

II. LITERATURE REVIEW

A number of algorithms have been proposed in the area of multi version locking. Some important work in this area is as follows:

JieShao et al[3]explains that in order to achieve read consistency the system support snapshot read which do not block write operation[3]. They proposed an architecture which comprises of three main components: partition, DTM, consistency coordinator. The concept of Local transaction identifier (LTID) has been introduced to the number of snapshot for each version number. The author proposed an algorithm to calculate the global snapshot version which is being created if LTID generated by each partition appeared in a serial manner. The snapshot related to maximum LTID values of each partition form a global consistent state [3].

Mohammad Sadoghi et al [10] proposed an KV indirection index maintenance technique that is useful to lessen the input/output burden by maintaining a single table which contain both up to date and historical data known as version enabled table. In order to prevent the change of unaffected attribute the proposed technique [10] decouple the physical and logical representation of record in order to differentiate between physical row identifier and logical record identifier

Kaloian Manassiev et al[5] proposed a method in which update transaction create a new version which is being broadcast to all other replicas. It evaluates a novel page level distributed concurrency control algorithm. The paper proposed two approach specifically update anywhere with no scheduler support and master update with scheduler support[5].The conflict is minimized in the master update by transferring all update transaction on a master replica and distributing read only transaction across a set of slave replicas in a version aware manner.

David Lomet et.al[2] proposed a new form of conflict manager known as timestamp conflict manager (TCM) which associate any committed transaction with its transaction timestamp which is made dependable with transaction isolation order. The TCM maintain a series of timestamp for a transaction.The paper has formulated certain principles that control the access of read-write operation on the particular data item. The timestamp range is adjusted whenever any conflict occurs so that read is ahead of write. This reduces transaction abort so that it there is no blocking of the operation in the case of write-write or read-write conflict.

Jose M Falerio et al[9] proposed a bohm, a new concurrency protocol for main memory multi version database system. It guarantee serializable execution while ensuring that read never blocks writes by separating concurrency control and version management from transaction execution. The bohm method does not require additional bookkeeping and coordination to achieve serializability.

Per Ake Larson et.al[7] discussed concurrency control mechanism optimized for main memory database system. It

has designed and implemented two multi version concurrency control (MVCC) methods, one is optimistic using validation phase and the other one is pessimistic which uses locking. The paper has implemented a prototype main memory storage engine which begins with a) high level overview of how data is stored b)how it is read and how updates are handled c) how to determine which version to read.

Hoda M.Oet.al[6] et al proposed a new approach for determining the optimal number of versions to be maintained by determining the minimum timestamp from the linked list and store it in an integer variable min_t . The authors presented an efficient and new scalable lock-free commit algorithm that allows write transactions to progress in parallel with garbage collection of unused versions.

Veluchandhar et.al[17] proposed an algorithm that uses the concept of backup mechanism to improve upon the performance of concurrent transactions. It uses multilevel security for distributed databases. The sub query analyzer monitors the data access and waiting time for the transaction. The paper discusses the different levels of security which are being effectively used for the enhancement of performance for concurrent transactions. The different security levels proposed in the model are view level, secret level and top secret level [17]. The proposed security model allows a transaction to issue read-down, read-equal and write-equal operations which is sufficient to prove that along with data access security is not violated.

Yang Zhan et al [4] paper introduces a method to convert pointer-based data structures into efficient lock-free implementations known as versioned programming. The technique allows an existence of different version of node to exist at the same time through an arbitrary composition of pointer such that each thread can pick the most suitable version and has a consistent view of the whole data structure[4].In this paper versioned tree implementations has been done [4]. With a fewer number of writers, versioned programming surpassed read-log-update which lock nodes.

Robert Gottstein et al[16] present the multi-version index(MV-IDX) approach which introduce the concept of index only visibility checks which is being significantly used to diminish the amount of input-output storage accesses thus thereby reducing the maintenance of index overhead. The technique achieve significantly lower response times and higher transactional throughput on OLTP workloads [16].It is an optimization to traditional indexing that partly addresses the issue of indexing .The index is augmented by index snippets which are comprised of a bitmap and a timestamp, correlated to the database pages.[16] For each page, a transactional timestamp is stored along with a single bit for each tuple version in the corresponding page visibility checks and the delay of index updates.

IV. PROPOSED APPROACH

The purpose of our algorithm is to obtain a global consistent version of the individual data item in the distributed environment. The scheme involve the collection of local transaction identifier only from the participating site whether it can be one or many and prohibits the involvement of all individual sites to obtain the global version as discussed in [3].This reduces the burden on each individual sites to participate in the formation of global version.

We have taken into account only the participating sites because of the following reasons:

1) Each version cannot be generated and present at all the sites unlike in [3] which emphasize the involvement of every individual site to obtain a global version. The highest version can be obtained from a single site and can be made as global version without prior existence of the same highest version at all the sites.

2) Distributed database system itself paves the way for data fragmentation as data is fragmented so only the necessary data can be present at the individual site. We have analyzed the read-write set at different sites in distributed environment which would primarily stress on how the version maintenance is done globally so that each site can access the correct and consistent version of the data.

Details of the proposed algorithm are given below

A. Proposed Algorithm

For a given transaction T

Subdivide the transaction T into sub transaction t1, t2….tn and add them to transaction queue (T_Queue) which contain the list of requesting transaction

while (T_Queue!=Empty)

Call for procedure Read_Set(I)

Call for procedure Write_Set(I)

End of T_Queue equals empty

Procedure Read_Set(I):A transaction T read the data item I

For all T in read mode

DTM contact the consistency coordinator to obtain global version

Global LTID(I)=LTID(s1) ∪..LTID(sm)

/*s1 and sm are the participating site which were involved to produce the global version*/

Return successful execution and commit.

Procedure Write_Set(I):A transaction T want to write on the data item I

while(Request_dataitem!=NULL)

{

DTM split the requesting T into sub transaction such that

T=t1, t2…tn

If (requesting site=single individual site)

{

Global LTID=Max_LTID(s1)

/* Maximum Version obtained from single site and notified through DTM to Consistency coordinator*/

}

Else

If (requesting_site! =single individual site)

{

Global LTID=LTID(s1)) ∪..LTID(sm)

/*The same highest version is generated at site s1 and sm to obtain a global snapshot version*/

}}

Wait until all running transaction commit

B. Explanation of the Algorithm

For a given transaction T, the clients contact the DTM which insert the transaction into transaction queue. If the requesting transaction is read only than we call the procedure read_set, the DTM contact the consistency coordinator to obtain a global version. If the requesting transaction is update than we call the procedure write_set, where DTM subdivide the transaction into sub transaction if requesting site is single individual site, than write is performed and a highest new version obtained is sent to consistency coordinator through DTM to obtain a global version without prior storage of the new version on all the existing sites. But if the requesting transaction involve more than one site than the same highest version is taken from individual site and are taken together to form a global snapshot version. The proposed algorithm relaxes the condition of the compulsory existence of every version created during transaction processing on all the existing sites which was earlier proposed by [3].

C. Comparative Analysis

The paper [3] proposed a matrix to calculate the global consistent version. In the matrix [3] each site is involved in forming the global version as shown in the matrix below. In order to obtain a global version for a particular data item, it imposes the restriction on the presence of same highest version to all the existing sites which is practically impossible in distributed system. Consider there are three sites S1, S2, S3 and three transactions T1, T2, T3 along with local transaction identifier generated for the three sites in the distributed environment as shown in tables 1 below

Table 1: Matrix for All Transaction Running At Every Site [3]

LTIDs / Partitions / Transactions	S_1	S_2	S_3
Initial	0	0	0
T_1	1	2	3
T_2	2	3	1
T_3	3	1	2

Let X=10

Table 2: Matrix for a Particular Data Item

Transaction/Sites	***S1***	***S2***	***S3***
T1	**X=12**	X=10	**X=11**
T2	X=11	**X=12**	X=10
T3	X=10	X=11	X=12
T4	-	-	**X=13**

In the given matrix [3] from the table 1 and table 2 the global version for the data item X is obtained by considering the same highest version to be compulsory present at all the distributed sites(S1,S2,S3).It means that the highest version X=12 has to be present at all the existing sites.

The matrix proposed by Jie Shao [3] completely fails if we consider the different snapshot version at a particular site than we cannot obtain a global consistent version according to the given matrix and algorithm proposed in [3].We have consider the single highest version to be present at a particular site as shown in table 3.Now with X=13 at site S3 as shown in table 4,we can obtain the global version of X this relaxes the condition of the presence of the same data item on all the existing sites.

Table 3: Matrix for Our Proposed Approach

Transaction/Sites	***S1***	***S2***	***S3***
T1	**3**	1	2
T2	2	**3**	1
T3	1	2	3
T4	-		**4**

Let X=10

Table 4: Matrix for a Particular Data Item

Transaction/Sites	***S1***	***S2***	***S3***
T1	**X=12**	X=10	**X=11**
T2	X=11	**X=12**	X=10
T3	X=10	X=11	X=12
T4	-	-	**X=13**

V. CONCLUSION AND FUTURE SCOPE

The paper discusses the multi version locking in distributed database. The drawbacks in the matrix proposed by [3] have been identified and solution is proposed in this paper to obtain the consistent version for the particular data item. Our scheme involves the collection of local transaction identifier only from the participating site whether it can be one or many. The proposed algorithm is a refinement on the work done so far. Any restriction on the involvement of all the existing sites to obtain a global version has not been imposed. This reduces the burden on each individual site. The future scope lies in extending our work with an intend to validate our new matrix for read and write operation in multi version locking on distributed environment through simulation experiments.

REFERENCES

[1]Marc Lupon, Grigorios Magklis and Antonio González, "Version Management Alternatives for Hardware Transactional Memory",ACM Toronto, Canada, October 26, 2008

[2]David Lomet, Alan Fekete, Rui Wang and Peter Ward,"Multi-Version Concurrency via Timestamp Range Conflict Management",pp 714-725, IEEE 28th International Conference on Data Engineering,2012

[3]Jie Shao, Boxue Yin, Bujiao Chen, Guangshu Wang, Lin Yang Jian,liang Yan, Jianying Wang and Weidong Liu "Read Consistency in Distributed Database Based on DMVCC", pp 142-151, IEEE 23rd International Conference on High Performance Computing 2016.

[4]Yang Zhan and Donald E. Porter, "Versioned Programming: A Simple Technique for Implementing Efficient, Lock-Free, and Composable Data Structures", SYSTOR '16, Haifa, Isreal, ACM, 2016

[5] Kaloian Manassiev,Madalin Mihailescu and Cristiana Amza "Exploiting Distributed Version Concurrency in a Transactional Memory Cluster"pp 198-208, New York, ACM 2006

[6]Hoda M. O. Mokhtar and Nariman Adel Hussein " A Novel Mechanism for Enhancing Software Transactional Memory",pp 278-283, July 07-09, ,Portugal, ACM, 2014.

[7]Per-Ake Larson, Spyros Blanas, Cristian Diaconu, Craig Freedman, Jignesh M. Patel and Mike Zwilling" High-Performance Concurrency Control Mechanisms for Main-Memory Databases",pp 298-309, 38th International Conference on Very Large Data Bases,Proceedings of the VLDB Endowment, Vol. 5, No. 4, August 27th, 2012.

[8]Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk and Danny Dig "How Do Centralized and Distributed Version Control Systems Impact Software Change"pp 322-333, ICSE '14, ACM, June,2014

[9]Jose M. Faleiro and Daniel J. Abadi,"Rethinking serializable multiversion concurrency control",pp 1190-1201, Proceedings of the VLDB Endowment, Vol.8,No.11,41st International Conference on Very Large Data Bases, September 2015.

[10]Mohammad Sadoghi, Mustafa Canim, Bishwaranjan Bhattacharjee,Fabian Nagel and Kenneth A. Ross"Reducing Database Locking Contention Through Multiversion Concurrency"pp 1331-1342, Proceedings of the VLDB Endowment, Vol. 7, No. 13, 40th International Conference on Very Large Data Bases, September 1st 5th 2014, Hangzhou, China.

[11]Juchang Lee, Hyungyu Shin, Chang Gyoo Park" Hybrid Garbage Collection for Multi-Version Concurrency Control in SAP HANA"pp 1307-1318, SIGMOD, ACM. June 26-July 01, 2016, San Francisco, CA, USA.

[12]Justin Levandoski, David Lomet, Sudipta Sengupta, Ryan Stutsman, and Rui Wang" Multi-Version Range Concurrency Control in Deuteronomy",pp 2146-2157, Proceedings of the VLDB Endowment, Vol. 8, No. 13, 42nd International Conference on Very Large Data Bases, September 5th – September 9th 2016, New Delhi, India.

[13]Joao A. Silva, Joao M. Lourenço and Herve Paulino" Boosting Locality in Multi-version Partial Data Replication",pp 1311-1314, SAC'15, ACM, April 13–17, 2015, Salamanca, Spain.

[14]Eran Chinthaka Withana, Beth Plale, Roger Barga and Nelson Araujo" Versioning for Workflow Evolution", pp 756-765, HPDC'10, ACM, June 20–25, 2010, Chicago, Illinois, USA.

[15]Nirmit Desai and Frank Mueller" Scalable Distributed Concurrency Services for Hierarchical Locking" Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS'03) IEEE, 2003

[16]Robert Gottstein, Rohit Goyal, Sergej Hardock, Ilia Petro and Alejandro Buchmann" MV-IDX: Indexing in Multi-Version Databases", pp 142-145, ACM, July 07 - 09 2014, Porto, Portugal

