Improvements in Shadow SWIFT using Selective Commit and Delayed Creation of Shadows

Alok Kumar
IIM

ABSTRACT
In normal shadow SWIFT commit protocol shadow of all the borrower transactions which satisfy a given criteria are created when the lock manager processes the request. This creates heavy overhead on the system and degrades its performance. In this paper, we have presented a new method of handling the requests by the borrower transactions. We create the shadow only when the borrower aborts due to abort of lender and can complete its execution in the remaining time. Further, the shadow uses the work already done by borrower transaction .We also propose the use of selective commit depending on Lender's commitment. Thus it prevents any overhead associated with the maintenance of the shadow as was the case in earlier shadow based protocols.

Keywords
Distributed database

1. INTRODUCTION
Database systems serve as a backbone to thousands of systems and applications. Some of the systems have very high demands for availability and very fast real-time responses are required. Usually, such systems keep generating workload of very large transactions for the distributed real time database, and a substantial part of the workload consists of write, read and also update kind of transactions. Poor availability of real time systems and/or slow responses in processing such transactions used by real time business applications could infact be financially devastating and also in worst cases, cause deaths or damages. For example: tele-communication systems, trading applications, online gaming systems, sensor network applications etc. Typically, a sensor network system consists of a large number of sensors (both wireless and wired) which give reports on the status of some real-life situations. The situations include motion, sound, pressure, temperature, moisture and velocity etc. Such sensors send their data to a central application which makes such decisions based on both past and present inputs. For enabling such networks for making better quality outputs, both the number of sensors and the frequency of their updates need to be increased. Hence, such sensor networks systems should be able to withstand an increasing amount of load. For systems like health care in hospitals, auto car driving applications, space shuttle control systems etc., data is required in real-time scenario, and should be extremely reliable since any such poor availability or extra time taken by delays can lead to significant loss of human lives.

Most of the systems listed above using Distributed Real Time Databases require a distributed transaction to be executed at multiple sites. A commit protocol is made to ensure that either all the effects of the transaction should persist or none of them persist at all even in case of failure of a site or communication link and the resulting loss of messages. Hence, it is required that the Commit processing transactions should add as little overhead as possible to the transaction processing. Hence, it shows that the design of a much better commit protocol is very important for Distributed Real Time Databases.

2. DISTRIBUTED REAL TIME DATABASE SYSTEM MODEL
In the distributed database system model, the global or central database is sub-divided into a group of local databases stored at various different- different sites and locations. A communication network is required which interconnects the various sites. There is nothing as such global shared memory in the system, and all the sitesneed to communicate through message exchanges over the communication network. We assume that all the transactions are firm and real time. Each of these transactions in the present model exists in the form of a coordinator that executes at the site of origination of the transaction and a group of cohorts which execute at various other sites, where the required data items are located. If there are any local data which are in the access list of the transactions, then in those cases one of the cohort is executed locally. Before accessing any data item, the cohort needs to obtain lock on the data items. Sharing of such data items in conflicting modes creates dependencies among the group of conflicting local transactions and cohorts, which constraints their commit order. We also assume that:

- The processing of a transaction needs the use of CPU and the data items which are located at a local site or remote site.
- Arrival of any transactions at a site is independent of the arrivals at any other site and uses Poisson distribution.
- Each cohort can make read and update accesses.
- Each transaction has to pre-declare its read-set (set of data items that the transaction will read) and the update-set (data items that the transaction will update).
- S2PL-HP is used for locking the data items.
- Cohorts are executed in parallel order.
- Any lending transaction can not lend the same data item in read/update mode to more than a cohort.
- Any cohort already in the dependency set of any other cohort can not permit another incoming cohort to read or update.
- A distributed real time transaction is said to commit, if the coordinator has reached commit decision before the expiry of the deadline at its site location. This definition applies ir-respective of whether all the cohorts have also received and recorded the commit decision within the deadlines or not.

- Studies have been made earlier for both main memory resident and disk resident database.
- Communication delays considered here is either 0 or 100 ms.
- For disk resident database, buffer space is considered sufficiently large to allow the retention of data updates until commit time.
- The updating of various data items is made in transactions own memory, not in place updating.

3. THE MODIFIED PROTOCOL

In this modified protocol, we propose that the creation of shadow be deferred until the abort of borrower due to Lender's abort, only if the shadow can complete its execution in the remaining time.

We note that shadow will be activated only when the borrower has to abort due to Lender's abort. So, we can remove the overhead of creating and storing of shadow at the time of its processing by the lock manager. Since, we don't already know whether the shadow will ever be used or not.

We propose the introduction of a new dependency: create-onabort.

This dependency is used to create the shadow if borrower aborts due to abort of Lender. This is applicable to all the cohort present in the Abort Dependency Set(ADS) of the Lender transaction.

Shadows will use the previous work done by the Borrower transaction, so it won't be required to repeat the work already done by the Borrower transaction. Shadows will only be required to place the old values of data items in place of the new values in the borrower transaction log and perform computations.

Since it does not require reading the data values again it saves time. The locks applied by the borrower can also be used by its shadow, so it saves time because the shadow is not required to apply the locks again.

The abort dependencies are created in cases of write-read and write-write conflicts. However, when the deadline time of borrower is approaching while lender is still deferring its commitment we can break the dependency so that the borrower is able to meet its deadline. We selectively run the borrower or its shadow if the lender has high chances of committing or we run its shadow if it has more chances of aborting respectively.

Obviously, in both cases we get an optimal result. Thus, we are able to remove all the overhead associated with the creation and storage of the shadows if the Borrower does not aborts or aborts after a time when the shadow cannot complete its execution. Since, in both these cases , there is no need to create a shadow because in first case the borrower transaction itself commits, not requiring the use of shadows. While in the second case, if the borrower aborts after a time such that the shadow cannot complete its execution there is no need to create a shadow. Thus, these improvements greatly increase system performance.

4. REFERENCES

[1] Abbott Robert and Garcia - Molina H., "Scheduling Real - time Transactions with Disk Resident Data," Proceedings of the 15th International Conference on Very Large Databases**,** Amsterdam, The Netherlands, pp. 385 - 395, 1989.

[2] Abbott Robert and Garcia - Monila H., "Scheduling Real - Time Transaction: a Performance Evaluation," Proceedings of the 14th International Conference on Very Large Databases, pp. 1 - 12, August 1988.

[3] Abdallah Maha, Guerraoui R. and Pucheral P., "One - Phase Commit: Does It Make Sense," Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS'98), Tainan, Taiwan, Dec. 14 - 16, 1998.

[4] Agrawal Divyakant, Abbadi A. El and Jeffers R., "Using Delayed Commitment in Locking Protocols for Real - Time Databases," Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Diego, California, pp. 104 -113, June 2 - 5, 1992.

[5] Agrawal Divyakant, Abbadi A. El, Jeffers R. and Lin L., "Ordered Shared Locks for Real - time Databases," International Journals of Very Large Data Bases (VLDB Journal), Vol. 4, Issue 1, pp. 87 - 126, January 1995.

[6] Aldarmi Saud A. and Burns A., "Dynamic CPU Scheduling with Imprecise Knowledge of Computation Time," Technical Report YCS - 314, Department of Computer Science, University of York, U. K., 1999.

[7] Aldarmi Saud A., "Real - Time Database Systems: Concepts and Design," Department of Computer Science, University of York, April 1998.

[8] Al - Houmaily Yousef J. and Chrysanthis P. K., "Atomicity with Incompatible Presumptions," Proceedings of the 18th ACM Symposium on Principles of Database Systems (PODS), Philadelphia, June 1999.

[9] Al - Houmaily Yousef J. and Chrysanthis P. K., "In Search for An Efficient Real - Time Atomic Commit Protocol,"

[10] Url = citeseer.nj.nec.com/47189.html.

[11] Al - Houmaily Yousef J., Chrysanthis P. K. and Levitan Steven P., "Enhancing the Performance of Presumed Commit Protocol," Proceedings of the ACM Symposium on Applied Computing, San Jose, CA, USA, February 28 - March 1, 1997.

[12] Al - Houmaily Yousef J., Chrysanthis P. K. and Levitan Steven P., "An Argument in Favor of the Presumed Commit Protocol," Proceedings of the IEEE International Conference on Data Engineering, Birmingham, April 1997.

[13] Arahna Rohan F. M., Ganti Venkatesh, Narayanan Srinivasa, Muthukrishnan C. R., Prasad S. T. S. and Ramamritham K., "Implementation of a Real - time Database System," Information Systems, Vol. 21 , Issue 1, pp. 55 - 74, March 1996.

[14] Attaluri Gopi K. and Salem Kenneth, "The Presumed - Either Two - Phase Commit Protocol," IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No. 5, pp. 1190 - 1196, Sept. - Oct. 2002.

[15] Audsley Neil C., Burns A., Richardson M. F. and Wellings A. J., "Data Consistency in Hard Real - Time Systems", YCS 203, Department of Computer Science, University of York, June 1993.

[16] Audsley Neil C., Burns A., Richardson M. F. and Wellings A. J., "Absolute and Relative Temporal Constraints in Hard Real Time Databases," Proceedings of the 4th Euromicro Workshop on Real - time Systems, IEEE Computer Society Press, Athens, pp. 148 – 153, June 1992.

[17] Bestavros Azer, "Advances in Real - Time Database Systems Research," ACM SIGMOD Record, Vol. 24, No. 1, pp. 3 - 8, 1996.