International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 18 (2018) pp. 13969-13977

Overload Handling in Replicated Real Time Distributed Databases

Anil Kumar Gupta[1], Vishnu Swaroop[2]

[1]*Computer Science and Engineering Department, Bhagwant University, Ajmer, India.*

[2] *Madan Mohan Malaviya University of Technology, Gorakhpur, India.*

Abstract

Many real-time applications need data services in distributed environments. However, providing such data services is a challenging task due to long remote data accessing delays [1] and stringent time requirements of real-time transactions. Overload occurs when the computation time of the transactions set exceeds the available processor time. A lot of algorithms have been suggested to deal with the overload of the distributed replicated real time database systems, but no algorithm deals with the overload of the system facilitating the dynamic environment. ORDER-RS, a dynamic replication algorithm enables two or more transactions submitted at site x to access different data items. But the overload generated at site x (due to different transactions) is not taken into consideration that can prevent and hinder many important transactions from meeting their deadlines which in turn can lead to worthless transaction. We have developed an algorithm that deals with the overload created in the ORDER-RS with the help of the term 'importance value of the transaction' according to which important transactions cannot be prevented to meet their deadlines due to the overload of the system.

INTRODUCTION

A database is a collection of related data [2]. This is collection of related data with an implicit meaning and hence is a database. A database represents some aspect of the real world; sometimes called the mini-world or the Universe of Discourse (UoD). Changes to the mini-world are reflected in the database. A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested. In other words, a database has some source from which data are derived, some degree of interaction with events in the real world, and an audience that is actively interested in the contents of the database [3].

A centralized distributed database management system (DDBMS) manages the database as if it were all stored on the same computer; however, a distributed database is a database in which portions of the database are stored on multiple computers within a network [4]. Users have access to the portion of the database at their location, so that, they can access the data relevant to their tasks without interfering with the work of others. The DDBMS synchronizes all the data periodically and, in cases where multiple users must access the same data, ensures that updates and deletes performed on the data at one location will be automatically reflected in the data stored elsewhere [5]. Distributed database system is the union of what appear to be two diametrically opposed approaches to data processing; database systems and computer network [6] as shown in figure 1.

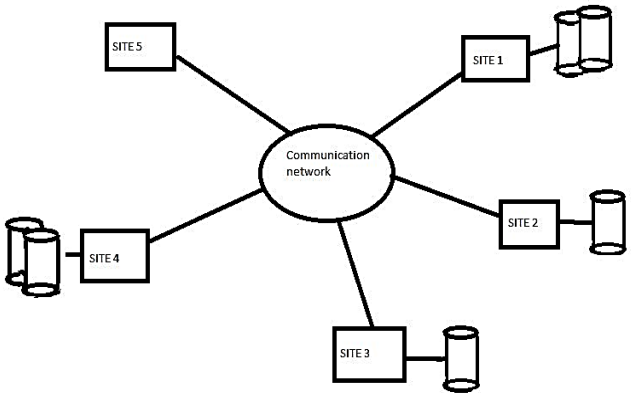

Figure 1: Architecture of distributed database system

CONCEPTS OF REAL TIME DATABASE

Real time is the quantitative notion of time. Real time is measured using the physical clock. A system is called a real time system, when we need quantitative expression of time (i.e. real time) to describe the behavior of the system. Real time applications are increasingly being implemented on distributed platforms. There are many reasons why distributed implementations are finding favor. One important reason is that it is often cost effective to have a distributed solution using many pieces of cheap hardware rather than having a centralized, sophisticated and costly machine. Another point going in favor of distributed implementations is fault-

tolerance. Key characteristics of real-time systems distinguish real time systems from non real time systems. Real time systems cover such an enormous range of applications and products that a generalization of the characteristics into a set that is applicable to every system is difficult.

Time constraints: Every real-time task is associated with time constraints. One form of time constraints is deadlines associated with tasks. A task deadline specifies the time before which the task must complete and produce the results. It is the responsibility of the real-time operating system (RTOS) to ensure that all tasks meet their respective time constraints [7].

New Correctness Criterion: In real time systems, correctness implies not only logical correctness of the results, but the time at which the results are produced is important. A logically correct result produced after the deadline would be considered as an incorrect result.

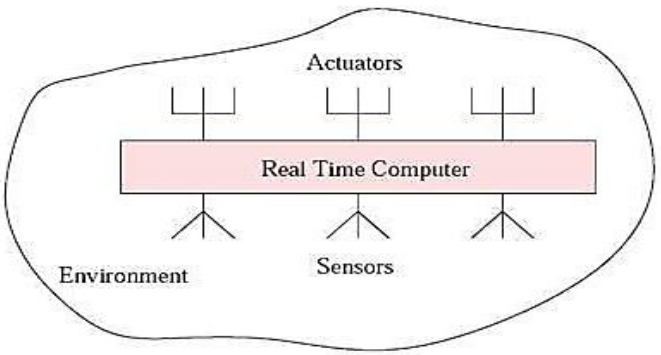

Figure 2: A schematic Representation of an Embedded Real Time System

Embedded: A clear majority of real-time systems are embedded in nature. An embedded computer system is physically "embedded" in its environment and often controls it as shown in figure 2.

Safety-Criticality: For traditional non-real-time systems, safety and reliability are independent issues. However, in many real-time systems, these two issues are intricately bound together making them safety-critical. A safety-critical system is required to be highly reliable since any failure of the system can cause extensive damages.

Concurrency: A real-time system usually needs to respond to several independent events within very short and strict time bounds. For instance, consider a chemical plant automation system, which monitors the progress of a chemical reaction and controls the rate of reaction by changing the different parameters of reaction such as pressure, temperature, and chemical concentration. These parameters are sensed using sensors fixed in the chemical reaction chamber. These sensors may generate data asynchronously at different rates. Therefore, the real-time system must process data from all the sensors concurrently, otherwise signals may be lost and the system may malfunction [8].

Distributed and Feedback Structure: In such systems, the different events of interest arise at the geographically separate locations. Therefore, the sensors and the actuators may be located at the places where events are generated. Many distributed as well as centralized real-time systems have a feedback structure as shown in figure 3. In these systems, the sensors usually sense the environment periodically. The sensed data about the environment is processed to determine the corrective actions necessary. The results of the processing are carried out the necessary corrective actions on the environment through the actuators, which in turn again cause a change to the required characteristics of the controlled environment, and so on.

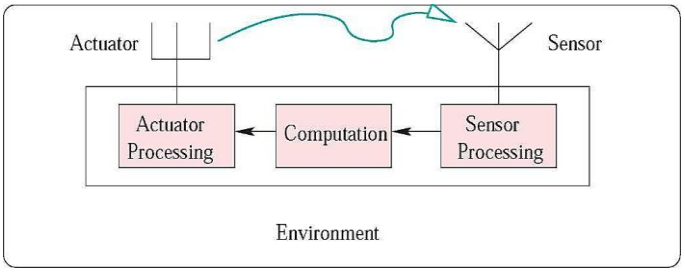

Figure 3: Feedback Structure of Real-Time Systems

Task Critical: Task criticality is a measure of the cost of failure of a task. Task criticality is determined by examining how critical are the results produced by the task to the proper functioning of the system. A real time system may have tasks of very different criticalities. It is therefore natural to expect that the criticalities of the different tasks must be taken into consideration while designing for fault-tolerance. The higher the criticality of a task, the more reliable it should be made. Further, in the event of a failure of a highly critical task, immediate failure detection and recovery are important. However, it should be realized that task priority is a different concept and task criticality does not solely determine the task priority or the order in which various tasks are to be executed [9].

Custom Hardware: A real-time system is often implemented on custom hardware that is specifically designed and developed for the purpose.

Reactive: Real-time systems are often *reactive.* A reactive system is one in which an on-going interaction between the computer and the environment is maintained. Ordinary systems compute functions on the input data to generate the output data.

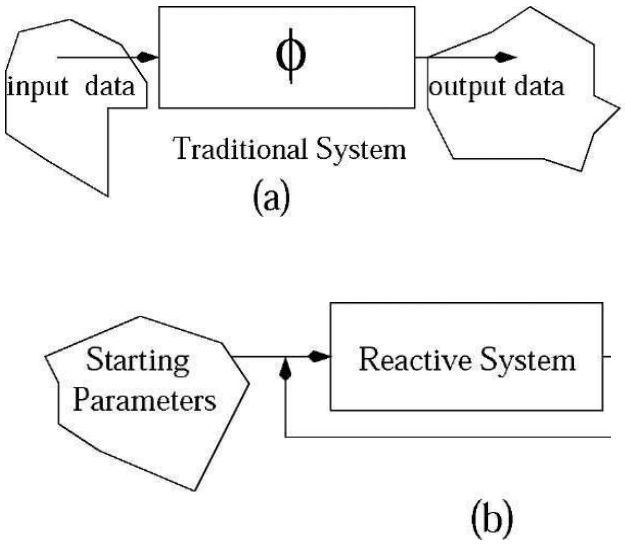

Figure 4: Traditional versus Reactive Systems

In contrast to the traditional computation of the output as a simple function of the input data, real-time systems do not produce any output data but enter an on-going interaction with their environment. In each interaction step, the results computed are used to carry out some actions on the environment. The reaction of the environment is sampled and is fed back to the system. Therefore, the computations in a real-time system can be non-terminating. This reactive nature of real-time systems is schematically shown in the Figure 4.

Stability: Under overload conditions, real-time systems need to continue to meet the deadlines of the most critical tasks, though the deadlines of non-critical tasks may not be met. This contrasts with the requirement of *fairness* for traditional systems even under overload conditions.

Exception Handling: Many real-time systems work round-the-clock and often operate without human operators. When there are no human operators, taking corrective actions on a failure becomes difficult. Even if no corrective actions can be immediate taken, it is desirable that a failure does not result in catastrophic situations. A failure should be detected, and the system should continue to operate in a gracefully degraded mode rather than shutting off abruptly.

REPLICATION

Replication is the key factor in improving the availability of data in distributed systems. Replication is the method of sharing information to make sure data consistency between redundant resources such as software or hardware components to improve reliability, fault-tolerance, or accessibility. It could be data replication if the same data is stored on multiple storage devices. Replication is the mechanism that automatically copies directory data from one directory Server to another. Using replication, any directory tree or sub-tree (stored in its own database) can be copied between servers. The Directory Server that holds the master copy of the information automatically copies any updates to all replicas. Replicated data is stored at multiple sites so that it can be accessed by the user even when some of the copies are not available due to site failures. A major restriction to using replication is that replicated copies must behave like a single copy, i.e., mutual consistency as well as internal consistency must be preserved. Synchronization techniques for replicated data in distributed database systems have been studied to increase the degree of concurrency and to reduce the possibility of transaction rollback [10]. In a distributed system, data can be stored at several sites. The proliferation of workstations and personal computers makes replication attractive for several reasons. One way of improving the availability of data in a system with unreliable sites for such a transaction is to replicate the data and store it at multiple sites. A single site failure does not make replicated data inaccessible; the system can access the data in the presence of failures even though some of the redundant copies are not available. In addition to improved availability, replication also enhances performance by placing the data closer to the process that requires it. For example, queries initiated at sites where the data are stored can be processed locally without incurring communication delays and the workload of queries can be distributed to several sites where the subtasks of a query can be processed concurrently. Two major technological developments have made the implementation of replication techniques cost-effective: inexpensive processors and memory making it cost-effective to develop large networks and new communication technology making it feasible to implement distributed algorithms with substantial communication requirements. However, the benefits of data replication must be balanced against the additional cost and complexities introduced for the synchronization of replicated data.

The inherent communication delay between sites that store and maintain copies of a replicated data makes it impossible to ensure that all copies are identical always when updates are processed in the system. The principal goal of a synchronization mechanism for replicated data is to guarantee that all updates are applied to each copy in a way that assures the mutual consistency. Mutual consistency is not the only constraint a distributed system must satisfy. In a system where several users access and update data concurrently, operations from different transactions may need to be interleaved and allowed to operate concurrently on data for better system throughput. An interleaved execution of read and write operations of transactions may produce incorrect results.

Concurrency control is the activity of coordinating concurrent accesses to the database to provide the same effect as if each request is executed in a serial fashion. The task of concurrency control in a distributed system is more complicated than that in a centralized system mainly because the information used to make scheduling decisions is itself distributed, and it must be managed properly to make correct decisions. Unless a correct concurrency control mechanism [11] is used to restrict the methods of interleaving the operations from different transactions, update may be lost, and incorrect retrieval would occur [12].

ISSUES IN DISTRIBUTED REPLICATED REAL-TIME DATABASE

Many issues affecting the design of a replicated DRTDBS is to maintain its requirements [13]; Data Consistency and Scalability are the main issues that are very important. All those critical systems need data to be obtained and updated in a timely fashion. But sometimes data that is required at a location is not available when it is needed and getting it from remote site may take too long before which the data may become invalid. This potentially leads to large number of transactions missing their deadline and violating the timing constraints of the requesting transaction. One of the solutions for the above-mentioned problem is replication of data in real-time databases [14]. By replicating temporal data items, instead of asking for remote data access requests, transactions that need to read remote data can now access the locally available copies which help transactions meet their time and data freshness requirements [15]. Replication in DRTDBs is used to remove unpredictability of network delays or network partitioning, that the database is fully replicated to all nodes. It also improves fault tolerance for the main-memory resident data. The key issues are listed below.

Data Consistency: There are two main approaches to maintaining consistency in a distributed database; pessimistic and optimistic. We say that the pessimistic approaches support immediate consistency, while optimistic approaches only guarantee eventual consistency.

Scalability: While relational database systems are a good fit for applications that follows ACID properties, they are not appropriate for applications that undergo rapid surges in user activities.

Design issues: Replication set size decides whether to replicate an entire table, a subset of a table, or data from more than one table. This is a trade-off among the amount of data that changes, the overall table size, and the complexity of the link. Replication in Distributed real-time database systems also used to remove unpredictability of network delays or network partitioning, that the database is fully replicated to all nodes. It also improves fault tolerance for the main memory resident data.

Replicated State Synchronization and Consistency: Various research and active usage has made data replication a well-understood and commonly applied technique for different purposes like increasing scalability, availability, fault tolerance and responsiveness of distributed systems. When using replication of application data, a synchronization mechanism must be incorporated into the distributed system to guarantee a particular consistency model of the data copies. Distributed applications differ in several characteristics like the size of the distributed system and the database. The overall update rate of data or the ratio of read and write accesses concepts has been developed. For example, active or passive replication with eager or lazy synchronization. Each of them behaves differently regarding characteristics like scalability, fault-tolerance or responsiveness. Two main update propagation concepts eager replication and lazy replication discussed below are generally presented and discussed regarding their suitability for being used in the novel entity replication approach for interactive virtual environments presented.

Eager Replication*:* The concept of eager replication updates all replicated copies of the object being changed as part of the original processing of the received client request. An update operation, therefore, is executed on all distributed copies of object before a notification is returned to the client. This synchronization scheme requires using nested transactions with a distributed commit protocol like the two-phase commit or a reliable atomic multicast for ensuring atomicity of the distributed update. The servers maintaining the copies must communicate in several steps before the operation can be performed and the response can be send to the client [16].

Lazy Replication: Lazy replication does not immediately update all distributed replications but only changes the state of the local copy at the replication server receiving the original request before sending the response to the client. The propagation of the new state of the object to the other copies is done eventually some time later. The lazy replication provides a fast response to the client because the server receiving the original request already sends a notification response after the local update execution. However, this approach requires an additional coordination mechanism for ensuring consistency of the replication application state. Without such a subsequent mechanism, replicated object copies would diverge because each server changes objects' state independently of each other when executing the original client request.

PROTOCOL FOR DYNAMIC REPLICATION IN REAL-TIME DISTRIBUTED DATABASE

There are several methods of replication control in medium-scale or large-scale distributed real-time database systems and a replication algorithm called On-demand Real-time Decentralized Replication (ORDER) is designed to work in an environment where all data types and relations in the system are known a priori and transactions are short-term periodic transactions. The ORDER algorithm may not perform well in large-scale distributed real-time databases. First, in large-scale distributed systems, it might be very costly or impossible for every site in the system to maintain detailed information about all data items in the system. Second, sometimes it may not be efficient to fetch all fresh data items directly from their primary site. Instead, the site can get fresh copies from some existing active replicas that are closer, i.e., the replicas that could be reached with less transmission delays. However, in term of scalability, algorithm On-demand Real-time Replication with Replica Sharing (ORDER-RS) can enhanced a replication algorithm. With this algorithm, large distributed systems are divided into small groups called cliques based on the network topology. The replicas within one clique are shared by the clique members. Compared with the ORDER algorithm, the ORDER-RS algorithm can further improve the system performance when the system size scales to very large sizes.

ORDER-RS Replication Algorithm Concept

The goal of the ORDER-RS algorithm is to gain efficiency over replication strategies such as full replication by dynamically changing the update frequency and update duration of replicas [17]. In the database model, each node may contain multiple temporal data items and replicas. The primary copy of a data item is updated periodically at a given basic update frequency (BUF) while its replicas are updated at different extended update frequencies (EUF) specified by the incoming application transactions. All replicas of a data item are updated using the fresh value from their primary copy. Therefore, the EUF upper bound for a given data item is the BUF of the primary copy. When a replica is updated periodically, it is called an active replica. Otherwise, it is called a dormant replica. An active replica will become dormant if it is no longer needed by any incoming application transactions. The time that it turns into a dormant replica is called its closing time (CT) [18]. In the periodic transaction workload model, the transactions are periodic with definite data requirements and service durations. A transaction may arrive at any time and each transaction may contain requests for multiple data objects from different sites. The specification for a transaction is as follows:

{TID, TD,EXETime, SF,DS}: Each transaction specification contains a transaction identifier (TID), transaction duration (TD), execution time (EXETime), slack factor (SF), and a data set (DS). The transaction's data set consists of elements of the following format:

DataObject {SiteID,DataT ype,DataID, FR}: The specification for one data object consists of information of the database identifier (site ID), data type, data identifier (data ID) and the freshness requirement (FR) of that data item.When a transaction arrives, it requests a data service with specific data freshness requirements and indicates the duration of the service. When this duration expires, the transaction no longer requests data objects at this site.

Algorithm Description

In the ORDER algorithm, the update frequency and update duration of replicas are dynamically controlled to satisfy the data freshness requirements of the incoming transactions. The algorithm, when receiving a transaction, evaluates the data needs of the incoming transaction and creates data replicas for the transaction if the transaction can be admitted based on the current system conditions. It also registers the update frequency and duration to the primary sites of these replicas. In the algorithm, it is the job of the receiving site to register active replicas to their primary sites. It is the duty of the primary site to push updates to the active replicas at the extended frequencies requested by the incoming transactions. When a local site admits a transaction, the algorithm calculates the proper update frequency and update duration for each remote data item specified by the transaction. Suppose the algorithm receives a request for remote temporal data item i from site x. The pseudo code for the replication algorithm is given in Fig. 3.1.

As shown in Fig. 3.1, when a new transaction arrives, for each remote temporal data item the transaction requests, the algorithm tests whether there is an existing active replica for the remote data item. If there is already an active replica, the algorithm compares the current update frequency of the replica with the update frequency requested by the new transaction. If the new transaction requests the replica to be updated at a higher frequency, the update frequency of the active replica is changed to the new update frequency requested by the transaction. In that case, the closing time of the replica is also changed to the current time plus the duration of the new transaction. If the replica update frequency requested by the new transaction is less than the original update frequency, the algorithm does not need to do anything because the current update frequency is high enough [19]. If there is no active replica for the remote temporal data item, the algorithm creates an active replica for that data item

using the update frequency and duration of the new transaction.

To maintain the minimum update frequency for all active replicas, the algorithm must keep track of all transactions that use the data replicas until they expire. The algorithm also needs to re-calculate the update frequency of replicas that are accessed by an expiring transaction. The pseudo code for transaction departure is shown in Fig.3.2. As shown in Fig. 3.2, when a transaction is exiting the system, its requested update frequencies on active replicas are checked. If it requests the highest update frequency for an active replica, the update frequency and closing time of the active replica need to be re-calculated. The system needs to find the maximum requested update frequency without the expiring transaction. The closing time of the replica is then set to the expiring time of the transaction that requests the highest update frequency.

```
//Current_EUF(i, x): The Current extended update
//frequency for temporal data i from site x.
//New_EUF(i, x): The new transaction requested
//update frequency for temporal data i from site x.
//Current_CT(i, x): The current closing time of the
//Replica for temporal data i from site x.
//TD: The incoming transaction requested duration.
If( ExistActiveReplica(i, x) )
{//There exist an active replica if(Current_EUF(i,
x) >= New_EUF(i, x))
{ //Use Current EUF
//Nothing to do here.
}
else {
//Use the New_EUF
Current_EUF(i, x) = New_EUF(i, x); //Set new replica
closing time Current_CT(i, x) = Current_Time + TD;
Register_Active_Replica (Current_EUF(i, x),
Current_CT(i, x));
}
}
else {
//There is no active replica Create_Active_Replica
(i, x); Current_EUF(i, x) = New_EUF(i, x);
Current_CT(i, x) = Current_Time + TD;
Register_Active_Replica (Current_EUF(i, x),
Current_CT(i, x));
}
```

Figure 3.1. Pseudo Code for Update Frequency and Duration Calculation (on transaction arrival)

```
//Current_EUF(i, x): The Current extended update
//frequency for temporal data i from site x. //EUF(i, x):
The expiring transaction requested //update frequency for
temporal data i from site x. //Current_CT(i, x): The
current closing time of the //replica for temporal data i
from site x.
//Find_Max_EUF(i, x): Find the maximum requested //EUF for
temporal data i from site x.
//Find_TD() : Find the transaction expiring time
//corresponding to the requested update frequency. If(
ExistActiveReplica(i, x) ) {
//There exist an active replica if(Current_EUF(i, x)
>EUF(i, x)) { //Nothing to do here.
}
else {

//Use the maximum requested EUF Current_EUF(i, x) =
Find_Max_EUF(i, x); //Set the replica closing time
Current_CT(i, x) = Find_TD(Current_EUF(I, x));
Register_Active_Replica (Current_EUF(i, x),
Current_CT(i, x));
}
}
```

Figure 3.2. Pseudo Code for Update Frequency and Duration Calculation (on transaction departure)

Order-RS: Dynamic Replication Algorithm

An example is given in Fig. 3.3. In the figure, two real-time databases DB1 and DB2 are connected by high speed LAN. DB1 has an active replica for a remote temporal data item. The active replica is updated periodically using the fresh data values from the primary site. Now, DB2 admits a new transaction which needs the same remote data. It is not efficient for DB2 to fetch the data value from remote site again since there is already an active replica in DB1, which can be easily reached. Fetching fresh data directly from the primary site unnecessarily increases the workload of the network and the primary site. Instead, it is more desirable to fetch the data from DB1. If the update frequency of the active replica at DB1 can satisfy the requirements of the new transaction in DB2, the system only needs to replicate data from active replica 1 to active replica 2.

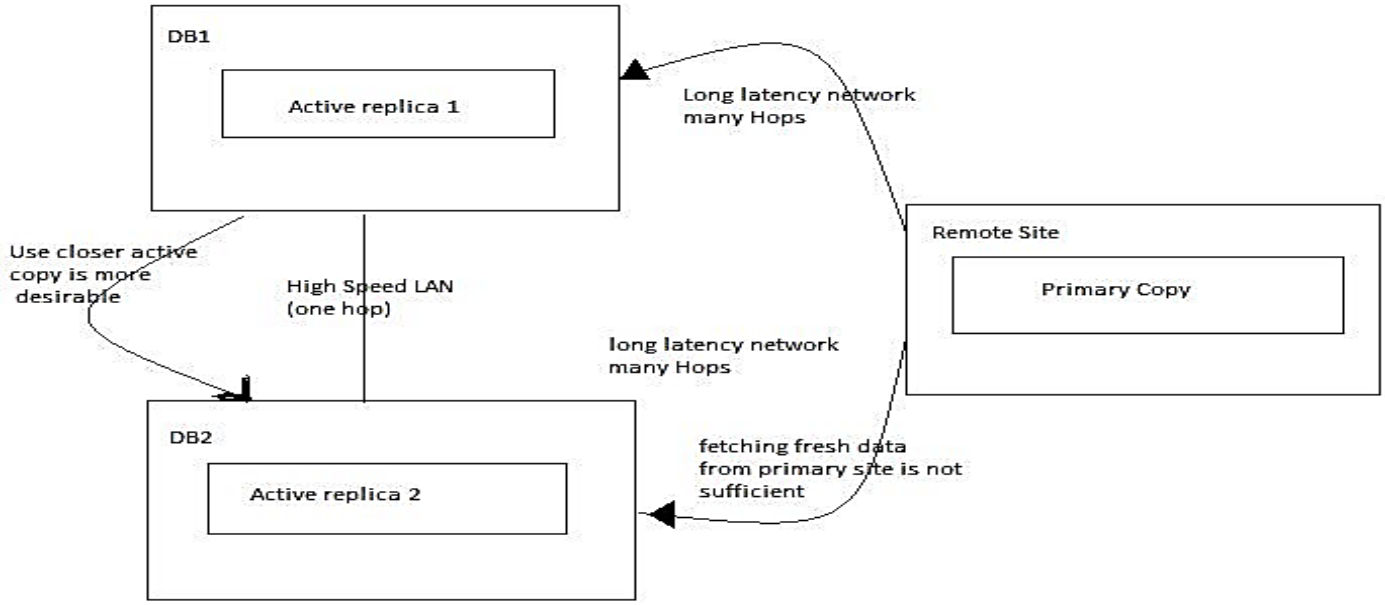

Figure 3.3. Fetching Fresh Data from Closer Active Replicas

If the new transaction demands higher update frequency, the system can now stop replicating data from primary copy to active replica 1. Instead, it replicates data directly from the primary copy to replica 2 (at higher update frequency) and then replicates the data from replica 2 to replica 1. An example is shown in Fig. 3.4. In the figure, there is a medium-scale distributed real-time database system which consists of 24 real-time database servers. The real-time database servers are connected by 8 transmission stations. The systems are divided into 6 cliques based on the network topology. The real-time databases within each clique are connected by wired high speed networks, while the cliques are connected by multiple-hop wireless networks. We also assume that the clique members from the same clique know the existence of each other. These assumptions match the real system configuration in combat control systems. In those systems, the real-time database servers within one ship or submarine are connected by high speed Ethernet while the communication between ships are via global wireless networks or on-shore transmission stations.

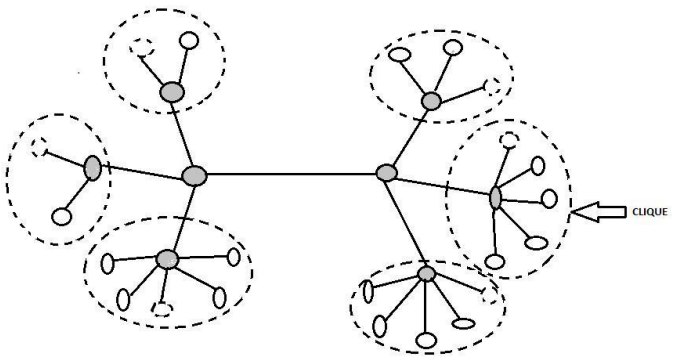

Figure 3.4. Replica Sharing in Large Distributed Systems

With the ORDER-RS algorithm, the replicas in one clique are shared by the database servers in that clique. In each clique, there is a clique leader which manages the replication process in that clique. The clique leader controls all the replicas in that clique and acts as a proxy for clique members when they need temporal data items from database servers in other cliques. When a clique member needs remote temporal data, it first checks where the requested data item resides. If the requested data item resides within the same clique, it just sends a request to the clique member that has the requested data to set up the replication process. If the data resides in a database server in a different clique, it sends the request to the local clique leader. Upon receiving the request, the local clique leader checks whether there is already a replica for the same data within the clique. If there is, the clique leader changes the replica update frequency to the current highest requested frequency and all requests from within the clique can now share the replica. If there is no replica corresponding to that requested remote data item, the clique leader sets up a new replica with the requested update frequency and duration. The update frequency and duration calculation processes on transaction arrival and departure are almost the same as those described in Fig. 3.1 and Fig. 3.2 except that the replication between two clique members within a clique is handled directly by the clique members themselves while the replication between database servers in different cliques is handled by the clique leaders.

PERFORMANCE EVALUATION OF ORDER-RS PROTOCOL

We simulate a larger distributed real-time database system running the ORDER, ORDER-RS and the proposed algorithm. The system consists of 32 real-time database servers and is divided into 8 cliques. Moreover, each clique has 4 sites associated with it. The network transmission delays

within a clique and between cliques are modelled separately. The transmission within a clique is faster, which takes 0.5 millisecond to finish, while the transmission between different cliques takes as long as 2 milliseconds. These system settings can be mapped to a distributed real-time database within a naval fleet where ships and submarines share the real-time data during a combat.

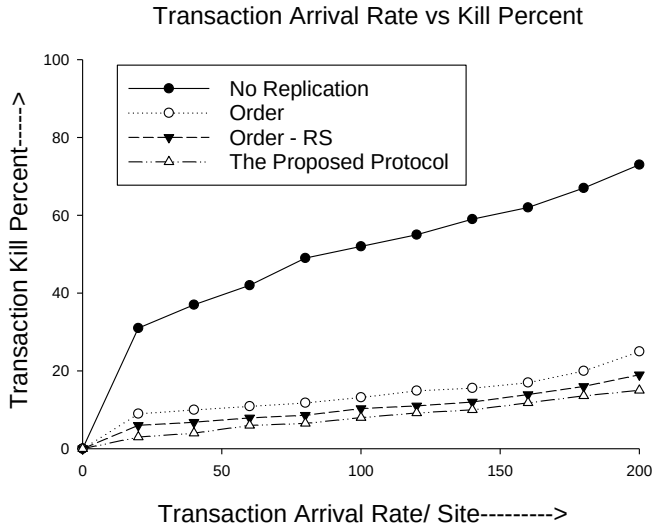

Figure 5. Comparative transaction kill percent of proposed algorithm with ORDER and ORDER-RS

The simulation results are shown in Fig. 5. As shown in Fig. 5, the ORDER and ORDER-RS and the proposed algorithm perform much better than the no-replication algorithm. In fact, the system without replication cannot deliver acceptable system performance at all. For example, the transaction miss ratio is as high as 30% when the average transaction workload is 20 transactions per second. It is due to the high latency of remote data access. Transactions with short execution time and short slack time can never meet their deadlines in a system without replication.

Both ORDER-RS and the proposed algorithm show acceptable transaction kill percentage. The proposed algorithm shows slightly better performance in different transaction workloads due to the replica sharing. The effect of replica sharing become more evident when the system workload gets higher. For example, the miss ratio difference between the ORDER-RS and the proposed algorithm is up to 2%. The reason is that when the transaction workload is higher, there are more replicas that can be shared. We do not show the performance of the full replication algorithm here because full replication is not applicable to systems of this scale. The replica update workload is so high that the database servers will be overloaded by only the replica update workload.

The system utilizations of the three algorithms are shown in Fig. 6. As we can see from the figure, among the three algorithms, our proposed algorithm gives the best system utilization results.

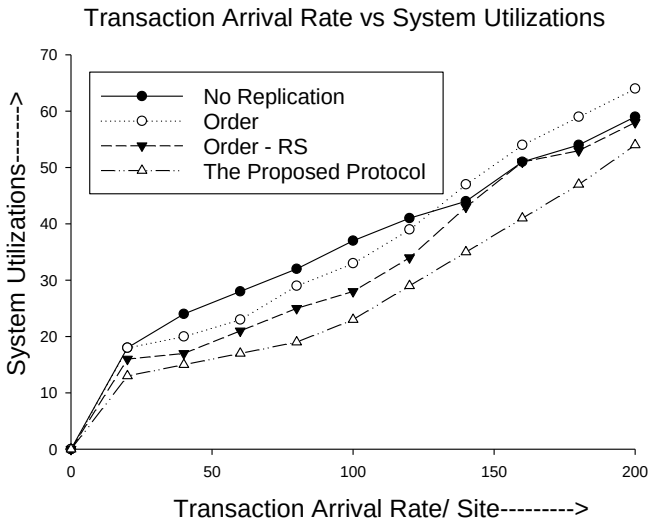

Figure 6. Graphical representation of system utilizations under Order, Order-RS and the Proposed Protocol

Note that the CPU utilization of the system without replication is higher than that of the system running ORDER algorithm when the transaction arrival rate is less than 140 transactions per second. Beyond that, the trend is reversed. The reason is that, at low system workload, the system without replication invokes a remote data operation each time there is a remote data access request, which tends to consume more CPU time than the ORDER algorithm. However, when the system workload is high, many transactions in the system without replication miss their deadlines while waiting for remote data accesses. The CPU utilization is relatively lower because many transactions never get fully executed.

CONCLUSION

At this point, we have exploited the difficulty of accessing the replicated data in firm RTDBS mainly concurrent execution of transaction in real time replicated database problem in conflict mode. This proposed mechanism can be easily integrated and implemented in current systems. This proposed protocol outperforms other protocols like ORDER and ORDER-RS in terms of system utilization and transaction kill percentage. Also, in this paper, it is said that a blocked transaction can borrow data item after reaching the High Priority point by the executing transaction, in this way the waiting time of transaction in queue will get decreased.

As a part of future work, an exhaustive real-life implementation of this work is required to establish this approach as a value-based commercial product.

REFERENCES

[1] U. Shanker, M. Misra and A. K. Sarje, "Distributed real time database systems: Background and literature review," International Journal of Distributed and Parallel Databases, Springer Verlag, vol. 23, no. 02, pp. 127-149,

2008.

[2] S. Pandey and U. Shanker, "Transaction Execution in Distributed Real-Time Database Systems," Proceedings of the International Conference on Innovations in information Embedded and Communication Systems, pp. 96-100, 2016.

[3] U. Shanker, M. Misra and A. Sarje, "Hard Real Time Distributed Database Systems: Future Directions," Proceedings of All India Seminar on Recent Trends in Computer Communication Networks, Dept. of ECE, IIT Roorkee, India, pp. 172-177, 2001.

[4] U. Shanker, M. Misra and A. Sarje, "Some performance issues in distributed real-time database systems," Proc. VLDB Ph.D. Work,Conv. Exhib. Cent. (COEX), Seoul, Korea, 2006.

[5] A. Srivastava, U. Shankar and S. Tiwari, "Transaction management in homogenous distributed real-time replicated database systems.," International Journal of Advanced Research in Computer Science and Software Engeering, vol. 2, no. 6, pp. 190-196, 2012.

[6] S. Pandey and U. Shanker, "IDRC: A Distributed Real-Time Commit Protocol," Procedia Computer Science 125, pp. 290-296, 2018.

[7] S. Pandey and U. Shanker, "Priority Inversion in DRTDBS: Challenges and Resolutions," Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (CoDS-COMAD '18), pp. 305-309, 2018.

[8] A. Srivastava, U. Shankar and S. Tiwari, " A protocol for concurrency control in real-time replicated databases system.," IRACST–International Journal of Computer Networks and WirelessCommunications (IJCNWC), vol. 2, no. 3, pp. 397-401, 2012.

[9] U. Shanker, M. Misra and A. K. & Sarje, "Priority assignment heuristic to cohorts executing in parallel," in 9th International Conference on World Scientific and Engineering Academy and Society (WSEAS), 2005.

[10] A. Srivastava and U. Shankar, "CCRTRD: A Protocol for Concurrency Control in Real-Time Replicated Databases System.," International Journal of Computer Science and Information Technology & Security (IJCSITS), vol. 4, no. 3, pp. 74-80, 2014.

[11] S. Pandey and U. Shanker, "On Using Priority Inheritance Based Distributed Static Two Phase Locking Protocol," Proceedings of the International Conference on Data and Information System (ICDIS), pp. 179-188, 2017.

[12] S. Pandey and U. Shanker, "CART: A Real-Time Concurrency Control Protocol," 22nd International Database Engineering & Applications Symposium (IDEAS 2018), Bipin C. Desai, Jun Hong, and Richard McClatchey (Eds.). ACM, New York, NY, USA., June 18-20, 2018 (Accepted).

[13] U. Shanker, M. Misra and A. K. Sarje, "SWIFT - A new real time commit protocol," Distributed and Parallel Databases, vol. 20, no. 01, pp. 29-56, 2006.

[14] A. Srivastava, U. Shankar and S. Tiwari, "A Replication Protocol for Real Time database System.," International Journal of Electronics and Computer Science Engineering , vol. 1, no. 3, pp. 1602-1608.

[15] A. Srivastava, U. Shankar and S. Tiwari, " Transaction processing in replicated data in the DDBMS.," International Journal of Modern Engineering Research (IJMER), vol. 2, no. 4, pp. 2409-2416, 2012.

[16] S. Pandey and U. Shanker, "A One Phase Priority Inheritance Commit Protocol," Proceedings of the 14th International Conference on Distributed Computing and Information Technology (ICDCIT) Bhubaneshwar, India, pp. 288-294, 2018.

[17] A. Singh, S. Srivastava and U. Shanker, "Singh, A.K., Srivastava, S. and Shanker, U., 2013. A survey on dynamic replication strategies for improving response time in data grids.," IJBSTR, 2013.

[18] P. Shrivastava and U. Shanker, "Replica Control Followi In Advances in Data and Information Sciences," in In Advances in Data and Information Sciences, 2018.

[19] A. Singh and U. Shanker, " A Hybrid Approach for Replica Placement-Replacement (Harp-R Algo) Algorithm," Data-Grid. i-Manager's Journal on Computer Science, vol. 3, no. 2, 2015.