International Advance Journal of Engineering, Science and Management (IAJESM)
ISSN -2393-8048, January-June 2022, Submitted in May 2022, iajesm2014@gmail.com

Study on a Replicated and Distributed Real-Time Database

Pawan Kumar Pnadey, Research Scholar (CSE), SunRise University, Alwar (Rajasthan)
Dr. Suraj Vishwanath Pote, Associate Professor, Dept. of CSE, SunRise University, Alwar (Rajasthan)

Abstract

Numerous real-time applications have a need for data services in decentralised settings. However, being able to provide such data services is difficult owing to the lengthy delays that are associated with distant data accessing and the severe time constraints that are associated with real-time transactions. When the amount of time required to compute the transactions in a set is more than the amount of time available on the processor, overload occurs. There have been many methods proposed to deal with the overload experienced by distributed replicated real-time database systems; nevertheless, there is not one strategy that deals with the overload experienced by the system that facilitates the dynamic environment. ORDER-RS is a dynamic replication method that allows several transactions to access distinct data items. These transactions may be filed at the same location. However, the overload that is created at site x as a result of the various transactions is not taken into consideration, despite the fact that this overload has the potential to prevent and hinder many significant transactions from meeting their deadlines, which in turn may result in the transaction being rendered worthless. We have developed an algorithm that deals with the overload that was created in the ORDER-RS with the help of the term 'importance value of the transaction.' According to this algorithm, important transactions cannot be prevented from meeting their deadlines because of the overload that was created in the system.

Keywords: environment, transactions, algorithms

INTRODUCTION

A database is a collection of data that is connected in some way. This is a collection of data that is connected to one another and has an implied meaning; as such, it is a database. A database is a representation of some facet of the actual world; this representation is sometimes referred to as the mini-world or the Universe of Discourse (UoD). The database is updated to reflect any changes made to the miniature world. A database is developed, constructed, and filled with data for the express purpose for which it was created. It already has a certain audience of users in mind, as well as some predefined applications that may be of interest to those consumers. A database, in other words, is characterised by the presence of a source from which the data are obtained, a degree of interaction with events that take place in the actual world, and an audience that is actively engaged in the contents of the database.

The database is managed by a centralised distributed database management system (DDBMS), which treats the data as if it were all stored on the same computer. In contrast, a distributed database is a database that is stored on many computers. which different parts of the database are kept on different computers that are connected together through a network .Users have access to the piece of the database that is located at their location, which enables them to access the data that is pertinent to their duties without

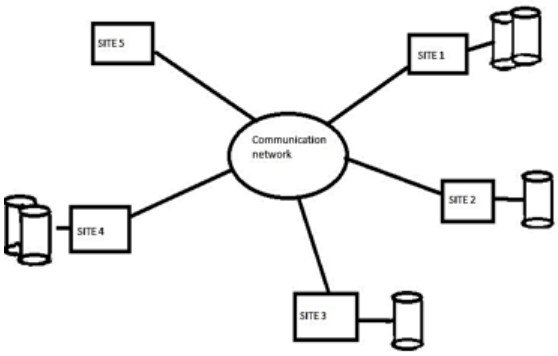

Figure 1: Architecture of Distributed Database System

interfering with the work that is being done at other locations. The DDBMS performs periodic data synchronisation on all of the data and, in situations where multiple users need access to the same data, ensures that any updates or deletes performed on the data at one location will be automatically reflected in the data stored elsewhere. In addition, the DDBMS synchronises the data so that, in cases where multiple users need access to the same data, it ensures that any updates Figure 1 depicts a distributed database system, which is the result of

the merging of two seemingly irreconcilable methods of data processing: database systems and computer networks. [6] This is shown in the accompanying figure.

CONCEPTS OF REAL TIME DATABASE

The quantitative concept of time is referred to as real time. The accuracy of the time is determined by the physical clock. When it is necessary to use a quantitative representation of time (also known as real time) in order to explain the behaviour of a system, we refer to that system as a real time system. Distributed platforms are being used in the development of real-time applications at an increasing rate. There is a multitude of factors contributing to the rise in popularity of distributed implementations. One key reason is that it is often more cost efficient to have a decentralised solution that makes use of many different pieces of inexpensive hardware as opposed to having a centralised solution that makes use of a complex and pricey machine. The absence of single points of failure is another another argument in favour of distributed implementations. tolerance is one of the key qualities that differentiates real-time systems from other types of computer systems that are not real-time. Because real-time systems may be used to such a vast array of different goods and applications, it can be challenging to generalise the features of these systems into a set that is relevant to each and every system.

Every real-time activity comes with its own unique set of challenges, including time limits. Deadlines that are attached to tasks are an example of a kind of time restriction. The period by which a job has to be finished and its outcomes produced is referred to as the deadline for the task. The real-time operating system, often known as RTOS, is the component of a computer system that is accountable for ensuring that all tasks complete within the allotted amount of time.

The New Correctness Criteria Are As Follows: In real-time systems, accuracy refers not only to the logical correctness of the outcomes, but also to the significance of the moment at which the findings are created. If the result was generated after the deadline, then it would be regarded an erroneous result even if it was logically valid.

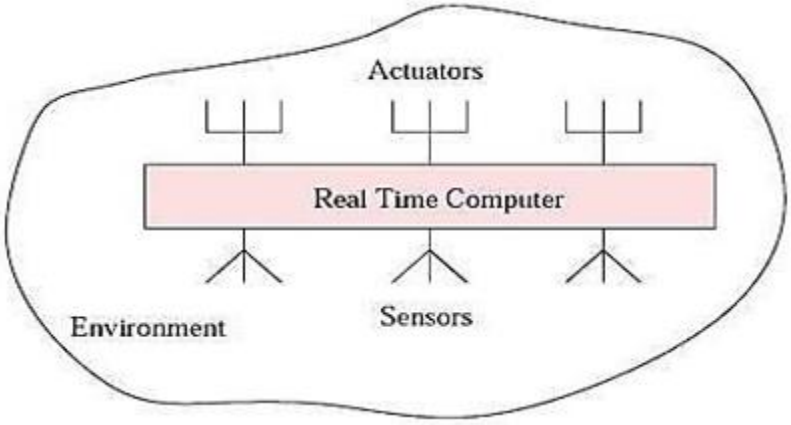

Figure 2: A Schematic Representation Of An Embedded Real Time System

Embedded: The overwhelming majority of real-time systems are now embedded in the surrounding environment. As depicted in figure 2, an embedded computer system is one that is physically "embedded" in its surroundings and often controls that environment.

Safety-Criticality: When it comes to conventional, non-real-time systems, safety and dependability are two separate concerns. On the other hand, these two concerns are inextricably intertwined in many real-time systems, which makes them very safety-sensitive. A safety-critical system is needed to have a very high level of reliability since the consequences of any failure of the system might be very severe.

Concurrency is the ability of a real-time system to react to several independent events in a very short amount of time while adhering to extremely severe time limits. Take, for instance, a chemical plant automation system as an example. This kind of system monitors the progression of a chemical reaction and regulates the pace at which the reaction occurs by altering various parameters of the reaction, such as the pressure, temperature, and chemical concentration. These characteristics are determined by the use of sensors that were

permanently installed in the chemical reaction chamber It's possible that these sensors will provide data asynchronously and at varying speeds. Because of this, the real-time system has to handle data from all of the sensors at the same time; if it doesn't, signals might be lost and the system could become unreliable.

Structures that are Distributed and Involve Feedback: In these types of systems, the various events of interest take place in geographically distant places. As a result, the sites where events are created are a logical choice for the placement of the sensors and the actuators. The feedback mechanism shown in figure 3 is present in a significant number of real-time systems, both centralised and distributed. In these types of systems, the sensors often take readings of the surrounding environment at regular intervals. The data that is sensed about the environment is analysed in order to establish the essential actions that need to be taken. The results of the processing are used to carry out the necessary corrective actions on the environment via the actuators, which in turn again create a change to the needed characteristics of the controlled environment, and so on and so forth.

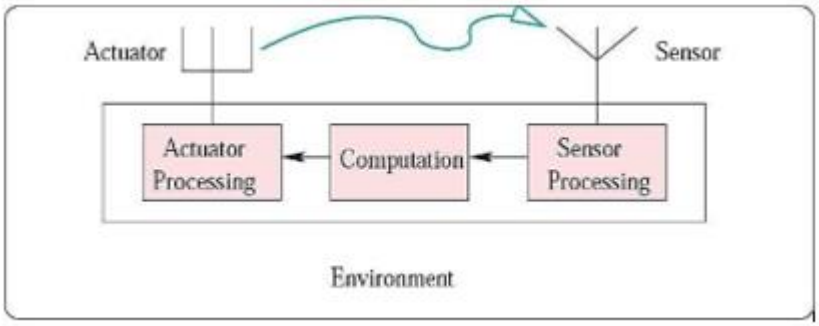

Figure 3: Feedback Structure Of Real-Time Systems

The criticality of a job may be seen as a measurement of the potential losses that might result from its failure. The importance of the outputs that a task produces to the efficient operation of the system is one of the factors that goes into determining the work's level of criticality. It's possible for a real-time system to have jobs with wildly varying degrees of importance. When designing for fault tolerance, it is only normal to anticipate that the criticalities of the various jobs would need to be taken into mind. When a task's degree of criticality increases, the degree to which it should be dependable also rises. In addition, it is essential to have quick failure detection and recovery procedures in place in the event that a very vital job fails. However, it is important to keep in mind that task priority is not the same thing as task criticality, and that task criticality is not the only factor that determines task priority or the sequence in which distinct activities are to be carried out.

Hardware Created and Built Specially for the Purpose Real-time systems are often implemented on specialised hardware that is designed and developed just for the purpose of running the system.Real-time systems are often described as being reactive. One definition of a reactive system is one in which there is a continuous interaction taking place between the computer and the surrounding environment. The output data of ordinary systems are generated by applying functions to the input data in order to calculate those functions.

Real-time systems, in contrast to more conventional methods of computing the output as a straightforward function of the data fed into it, do not generate any output data but rather engage into a continuous interaction with the environment in which they are embedded. The findings that were calculated are used in each phase of the interaction process to carry out various actions on the environment. The system takes a sample of the response of the environment and then feeds that information back into itself. As a result, the calculations in a real-time system have the potential to go on indefinitely. Figure 4 provides a diagrammatic representation of this reactive behaviour that is characteristic of real-time systems.

Stability: Under situations of overload, real-time systems are required to continue meeting the deadlines of the jobs that are considered the most vital, even while the deadlines of tasks that are not considered essential may not be fulfilled. This is in contrast to the demand of fairness that is placed on conventional systems even when they are operating at capacity.

Processing of Exceptions: The majority of real-time systems are designed to run unattended and so are capable of working around the clock. It is more difficult to take remedial steps in the event of a failure when there are no human operators present. It is to everyone's benefit, even if there is no immediate possibility of taking remedial steps, that a failure does not result in catastrophic circumstances. A failure should be identified, and rather than turning off suddenly, the system should continue to run in a state that is gracefully degraded rather than the original one.

REPLICATION

When it comes to enhancing the availability of data in distributed systems, replication is the most important element. The process of distributing information in order to ensure data consistency among redundant resources, such as software or hardware components, in order to enhance dependability, fault-tolerance, or accessibility is referred to as replication. If the same data is saved on many devices, it may be a case of data replication.

storage devices. The process by which directory data is routinely copied from one directory Server to another directory Server is known as replication. Any directory tree or sub-tree, each of which is maintained in its own database, may be replicated on other servers by using replication. The Directory Server, which stores the authoritative version of the information, is responsible for propagating any revisions to all of the replicas. Data that has been replicated is kept in numerous locations so that users may still access it even if some of the copies are unavailable as a result of problems at one of the storage locations. The requirement that repeated copies act and behave like a single copy, i.e., that both mutual consistency and internal consistency be maintained, is one of the most important limitations placed on the use of replication. Studies have been done on synchronisation strategies for replicated data in distributed database systems with the goals of increasing the degree of concurrency and decreasing the likelihood of a transaction rolling back. It is possible to store data in several locations when using a distributed system. The widespread availability of workstations and personal computers has made replication an appealing option for a number of different reasons. Replicating the data and storing it at several locations is one strategy for enhancing the availability of the data in a system that has sites that are not dependable for the kind of transaction being performed. Replicated data are not rendered inaccessible by the failure of a single site; the system is still able to retrieve the data even when failures occur at many sites, even if some of the redundant copies are unavailable. Replication improves the performance of a system by bringing the data that is required for a process closer to where it is being replicated. This results in enhanced availability. For instance, queries that are started at locations where the data are stored can have their results processed locally without any communication delays being incurred, and the workload associated with queries can be distributed across multiple locations where the query's individual subtasks can be worked on simultaneously. The implementation of replication techniques has become more cost-effective as a result of two significant advances in technology: the availability of inexpensive processors and memory, which has made it more cost-effective to develop large networks; and the development of new communication technology, which has made it feasible to implement distributed algorithms with substantial communication requirements. Nevertheless, it is necessary to weigh the advantages of data replication against the extra costs and difficulties that are incurred throughout the process of synchronising duplicated data.

It is impossible to ensure that all copies are identical at all times when updates are processed in the system due to the inherent communication delay that exists between sites that store and maintain copies of replicated data. This delay occurs between sites that store and maintain copies of replicated data. The primary objective of a synchronisation mechanism for duplicated data is to ensure that all changes are applied to each copy in a manner that ensures the mutual consistency. This is accomplished by ensuring that all updates are done in a synchronous manner. A distributed system must not only meet the requirement of mutual consistency, but also other constraints. In a system in which several users access and change data at the same time, it is possible that operations from various transactions will need to be

interleaved and permitted to work simultaneously on data in order to achieve a higher level of system throughput. The execution of read and write operations of transactions in an interleaved fashion may provide inaccurate results.

The process of coordinating concurrent requests to a database in order to get the same result as would be achieved by carrying out each request in a sequential order is referred to as concurrency control. The process of concurrency control in a distributed system is significantly more difficult than its counterpart in a centralised system. This is primarily due to the fact that the information that is used to make scheduling decisions is also distributed across the system, and it must be managed effectively in order to arrive at the best possible decisions. It is possible that an update may be missed, and erroneous retrieval will take place, if an appropriate concurrency control mechanism is not utilised to limit the techniques of interleaving the activities coming from distinct transactions.

ISSUES CONTAINED WITHIN A REPLICATED AND DISTRIBUTED REAL-TIME DATABASE

The design of a replicated DRTDBS has a number of challenges in order to meet its criteria the most significant of which are maintaining data consistency and increasing the system's capacity to scale. All of these essential computer systems need data to be retrieved and kept up to date in a timely manner. On the other hand, there are instances when the data that is required at a place is not accessible at the time that it is required, and acquiring it from a distant site may take too long, at which point the data may become invalid. Because of this, it is possible that a significant number of transactions may fail to complete before the deadline, therefore breaching the time restrictions of the transaction that requested them. Replicating data in real-time databases is one of the approaches that may be used to address the issue that was described before. Transactions that need to read remote data can now access the locally available copies, which helps transactions meet their time and data freshness requirements .This is made possible through the replication of temporal data items, which eliminates the need for transactions to ask for remote data access requests. To eliminate the unpredictability of network delays or network segmentation, replication is used in DRTDBs. This ensures that the database is completely duplicated to each and every node. Additionally, it enhances the fault tolerance of the data that is resident in the main memory. The most important concerns are detailed down below.

Synchronization of Replicated States and Maintaining Consistency: Data replication has become a technique that is well understood and actively used for a variety of purposes, including increasing the scalability, availability, fault tolerance, and responsiveness of distributed systems. This has been made possible by various research and active usage of the technique. When implementing replication of application data, a synchronisation mechanism has to be included into the distributed system in order to assure that the data copies adhere to a certain consistency model. Applications that are distributed are distinct from one another in a number of aspects, including the scale of the distributed system and the database. It has been possible to establish notions such as the total update rate of data or the ratio of read accesses to write accesses. Take, for instance, active or passive replication in conjunction with eager or slow synchronisation. Every one of them acts differently with relation to qualities such as scalability, fault-tolerance, or reactivity. The two primary update propagation approaches, eager replication and slow replication, which will be explored more below, are broadly introduced and debated on their appropriateness for usage in the innovative entity replication technique for interactive virtual environments that is provided here.

Eager Replication: The idea behind eager replication is to ensure that all replicated copies of the item being altered are brought up to date as part of the initial processing of the client request that was received. Therefore, an update operation is carried out on all of the distributed copies of the object before a notice is sent back to the client. This particular synchronisation strategy calls for the use of nested transactions in conjunction with a distributed commit protocol, such as the two-phase commit, or a reliable atomic multicast, in order to guarantee the atomicity of the distributed update. Before the procedure can be

completed and the answer can be sent to the client the servers that are keeping the copies need to interact with one another in a number of different phases.

Lazy Replication: Lazy replication does not instantly update all remote replications; instead, it merely modifies the state of the local copy on the replication server that was given the initial request before it sends the answer to the client. The updated state of the object is finally propagated to the other copies at some point in the future after some amount of time has passed. A quick response is sent to the client by the lazy replication technique since the server that was given the initial request already delivers a notification response once the local update has been carried out. However, in order to guarantee that the state of the replication application is always consistent, this method necessitates the inclusion of a coordinating mechanism. Without such a subsequent process, duplicated object copies would become inconsistent due to the fact that after performing the first client request, each server modifies the state of the objects in a way that is independent from the other servers.

PROTOCOL FOR DYNAMIC REPLICATION IN REAL-TIME DISTRIBUTED DATABASE

In medium-scale or large-scale distributed real-time database systems, there are numerous techniques of replication control, as well as a replication algorithm called On-demand. Real-time Decentralized Replication, also known as ORDER, was developed with the intention of functioning in an ecosystem in which all of the data kinds and relations present in the system are known a priori and in which transactions are short-term periodic operations. In large-scale distributed real-time databases, it's possible that the ORDER algorithm won't perform very well. To begin, when large-scale distributed systems are in use, it may be prohibitively expensive or perhaps impossible for each site that makes up the system to keep precise information on all of the data elements that make up the system. Second, it is possible that it will not always be productive to get all of the most recent data items straight from their official website. Instead, the site may get new copies from some of the active replicas that already exist and are located nearby. These are the replicas that can be accessed with less delays in the transfer of data. On the other hand, the On-demand Real-time Replication with Replica Sharing (ORDER-RS) technique may improve a replication algorithm's scalability. Using this approach, big distributed systems are broken up into smaller groups that are referred to as cliques. These cliques are based on the topology of the network. The members of a single clique are responsible for sharing the clones among themselves. In comparison to the ORDER algorithm, the ORDER-RS algorithm is able to significantly increase the performance of the system even when the size of the system expands to extremely large levels.

ORDER-RS REPLICATION ALGORITHM CONCEPT

By dynamically adjusting the update frequency and update duration of replicas, the ORDER-RS algorithm hopes to achieve greater efficiency compared to other replication schemes, such as complete replication. Each node in the database model may have a number of copies and temporal data items stored inside it. While the main copy of a data item receives periodic updates at a certain basic update frequency (BUF), the replicas of that data item get updates at various extended update frequencies (EUF), which are set by the incoming application transactions. Every replica of a data item receives an update that uses the most recent value taken from its parent copy. As a result, the BUF of the main copy serves as the upper constraint for the EUF of any particular data item. An active replica is a copy that receives regular updates, as defined by the definition of the term. If this is not the case, we refer to it as a dormant duplicate. If there are no more incoming application transactions that need an active replica, then that replica will go into a dormant state. It is referred to by its closure time (CT), which is the point in time when it goes into a dormant state. The transactions that make up the periodic transaction workload model are repeated at regular intervals and have well defined data needs and service durations. At any point in time, a transaction may come, and within each transaction there may be requests for many data objects originating from several websites. The following is an example of the specification for a transaction:

OBJECTIVE

1. To study on Order-Rs Replication Algorithm Concept
2. To study on protocol for dynamic replication in real-time distributed database
3. To study on Replicated And Distributed Real-Time Database

ALGORITHM DESCRIPTION

In the ORDER method, the update frequency and update duration of replicas are constantly regulated in order to fulfil the data freshness requirements of the incoming transactions. These criteria demand that the data be as current as possible. When a transaction is received by the algorithm, it first assesses the data requirements of the transaction and then generates data replicas for the transaction if it is determined that the transaction may be allowed in light of the existing system circumstances. Additionally, it keeps track of the update frequency and duration to the principal sites that these replicas are connected to. Within the framework of the algorithm, the receiving site is responsible for registering active replicas with their respective main sites. The main site is responsible for pushing updates to the active replicas at the increased frequency that are needed by the incoming transactions. When a transaction is accepted by a local site, the algorithm determines the appropriate update frequency and update duration for each distant data item that the transaction specifies. Let's say that the algorithm is contacted by site x with a request for the distant temporal data item i. In Fig. 3.1, the pseudo code for the replication technique is shown for your perusal.

When a new transaction is received, the algorithm checks to see if there is already an active replica of the remote data item being requested by the transaction. This is illustrated in Figure 3.1. When a new transaction arrives, the algorithm performs this check for each remote temporal data item being requested by the transaction. The update frequency that is currently being used by the replica is compared by the algorithm with the update frequency that is being requested by the new transaction if there is already an active replica in existence. The update frequency of the current replica is adjusted to match the new update frequency that is required by the transaction if the new transaction demands that the replica be updated at a greater frequency than before. In such event, the time at which the replica is set to close is adjusted to reflect the current time plus the amount of time required for the new transaction. If the update frequency of the replica that is being requested by the new transaction is lower than the update frequency of the original transaction, the algorithm does not need to take any action since the existing update frequency is already sufficient [19]. In the event that there is not already an active replica for the distant temporal data item, the algorithm will generate one for that data item. use the new transaction's update frequency and duration as our guide.

In order for the algorithm to be able to keep track of all transactions that utilise the data replicas until they become obsolete, this is a requirement for maintaining the minimum update frequency for all active replicas. In addition to this, the algorithm has to recalculate the update frequency of any replicas that are being accessed by a transaction that is about to expire. Figure 3.2 depicts the pseudo code that is used for the transaction departure. When a transaction is about to leave the system, its desired update frequencies on active replicas are verified. This process is shown in Figure 3.2. In the event that it demands the maximum update frequency for an active replica, the calculation for the active replica's update frequency as well as its closure time will need to be redone. The software needs to determine the maximum update frequency that may be requested without causing the transaction to time out. The expiration time of the transaction that demands the greatest update frequency is subsequently used to establish the closure time of the replica.

ORDER-RS: DYNAMIC REPLICATION ALGORITHM

An example is presented in Fig. 3.3. The image depicts a high-speed local area network (LAN) that connects two real-time databases, DB1 and DB2. An operational duplicate of a distant temporal data item may be found in DB1. The fresh data values from the original site are copied over to the active replica on a regular basis so that it may be kept up to date. Now, DB2 will allow a new transaction that requires the same remote data to proceed. Since there is already an active copy of the data value stored in DB1, which can be accessed without any difficulty, it would be inefficient for DB2 to get the data value from the distant site a second

time. The demand of both the network and the main site is increased needlessly when new data is fetched straight from the primary site. Instead, retrieving the data from DB1 is the better course of action to take. The system only needs to duplicate data from active replica 1 to active replica 2 if the update frequency of the active replica at DB1 can match the requirements of the new transaction in DB2. In this case, the system does not need to replicate data from active replica 3 to active replica 1.

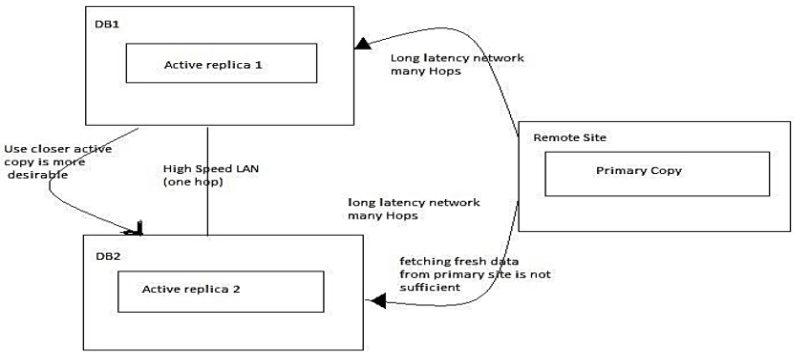

Figure 4 Fetching Fresh Data From Closer Active Replicas

If the new transaction needs a higher update frequency, the system is able to stop duplicating data from the main copy to the active replica 1 at this moment. This is only possible if the new transaction requires the higher update frequency. Instead, it makes a copy of the data by first copying it directly from the main copy to replica 2 (with a higher update frequency), and then it makes a copy of the data by first copying it from replica 2 to replica 1. Finally, it duplicates the data by first copying it from replica 2 to replica 1. An illustration of such an example may be seen in figure 3.4. In the picture, you can see an illustration of a database system that is an example of a medium-scale distributed real-time system. This infrastructure is equipped with a total of twenty-four real-time database servers. As a result of their being eight transmission stations, the real-time database servers are able to interact with one another. The topology of the network was used to categorise the various systems into six separate cliques, which were then grouped together. The wired high-speed networks that connect the real-time databases that are contained inside each clique are connected to the multiple-hop wireless networks that link cliques, and these multiple-hop wireless networks are connected to one another via the wired high-speed networks. Additionally, we take it as given that individuals who are a part of the same group are aware of the existence of other individuals who are a part of the same group. These presumptions are in line with the real configuration of the system that is used in the combat control systems. When it comes to these kinds of systems, the real-time database servers that are located on board a single ship or submarine are connected to one another by means of a high-speed Ethernet connection. On the other hand, communication between ships takes place across international wireless networks or on-land transmission stations.

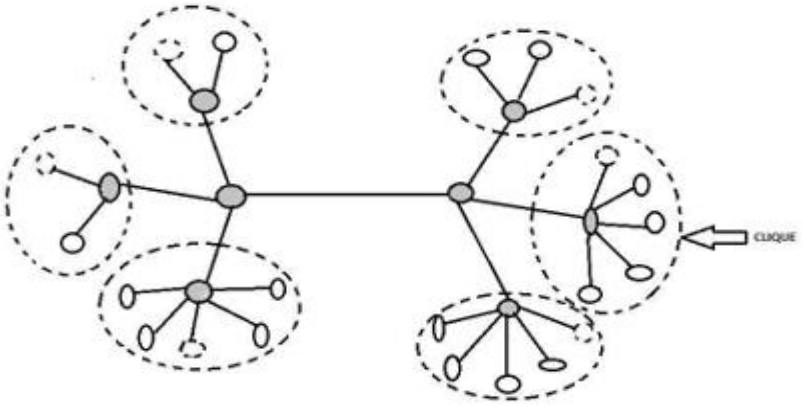

Figure 5 . Replica Sharing in Large Distributed Systems

Using the ORDER-RS algorithm, the database servers that belong to a certain clique will share the replicas that are located within that clique. There is a leader of each clique who is responsible for managing the replication process inside that particular clique. When members of a clique require temporal data items from database servers located in other cliques, the leader of that clique acts as a proxy for those members and controls all of the replicas included within that clique. When a member of the clique requires access to remote temporal data, it first investigates the location of the data item that is being sought. If the requested data item is held by a member of the same clique, it just has to send a request to that member to begin the replication process. This happens only if the requested data item is localised inside the same clique. If the data is located on a database server belonging to a separate clique, then the request is forwarded to the leader of the local clique. As soon as the request is received, the leader of the local clique checks to see whether there is already a duplicate of the data somewhere else inside the clique. If there is, the leader of the clique will adjust the replica update frequency so that it corresponds to the frequency that is now being requested the most, and all requests coming from members of the clique will now be able to share the replica. If the requested distant data item does not already have a replica that corresponds to it, the leader of the clique will create a new replica and give it the update frequency and duration that was requested. However, the replication between two clique members within a clique is handled directly by the clique members themselves, whereas the replication between database servers in different cliques is handled by the clique leaders. 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.

PERFORMANCE EVALUATION OF ORDER-RS PROTOCOL

We model a bigger distributed real-time database system in which ORDER, ORDER-RS, and the suggested algorithm are all being executed simultaneously. The network is comprised of 32 real-time database servers and is organised into a total of eight cliques. In addition to this, each clique is connected to four different websites. The transmission lags caused by the network Both the dynamics occurring within a clique and those occurring between cliques are modelled independently. The transmission that occurs within a clique is much quicker, taking just 0.5 millisecond to complete, however the transmission that occurs between separate cliques might take as long as 2 milliseconds to complete. These system parameters can be translated to a decentralised real-time database that is used by a naval fleet during a battle to facilitate the sharing of real-time data between ships and submarines.

CONCLUSION

At this point, we have taken advantage of the fact that it is difficult to access the duplicated data in the company RTDBS, specifically the concurrent execution of transaction problem in real time replicated databases. This new method can be readily incorporated and put into use in the systems that are currently in place. This suggested protocol surpasses previous protocols such as ORDER and ORDER-RS in terms of the proportion of transactions that are killed and the amount of system usage they need. Also, in this article, it is stated that a blocked transaction might borrow a data item after reaching the High Priority point by the executing transaction. As a result, the waiting time of transactions in queue will be shortened, which will be a positive outcome.

To establish this strategy as a value-based commercial offering, more effort, including a comprehensive real-world execution of this work, is necessary. This work will be included in future work.

REFERENCES

1. Chastek, Gary, Graham, Marc H., and Zelesnik, Gregory. Notes on Applications of the SQL Ada Module Description Language (SAMeDL). Tech. Rept. CMU/SEI-91-TR-12, Software Engineering Institute, June 1991.
2. Chung, J-Y. and Liu, J. W-S. Algorithms for Scheduling Periodic Jobs to Minimize Average Error. IEEE Real Time Systems Symposium, 1988, pp. 142-151.
3. Dayal et al. "The HiPAC Project: Combining Active Database and Timing Constraints". SIGMOD Record 17, 1 (March 1988), 51-70.

4. DeWitt, D. J., Katz, R. H., Olken, F., Shapiro, L. D., Stonebraker, M. R., and Wood, D. Implementation Techniques for Main Memory Database Systems. Proceedings of the ACM SIGMOD Conference on Management of Data, 1984, pp. 1-8.
5. Elhardt, K. and Bayer, R. "A Database Cache for High Performance and Fast Restart in Database Systems". ACM Transactions on Database Systems 9, 4 (December 1984), 503-525.
6. Eswaran, K. P., Gray, J. N., Lorie, R. A., and Traiger, I. L. "The notions of consistency and predicate locks in a database system". Communications of the ACM 19, 11 (1976), 624-633.
7. Fan, C. and Eich, M. H. Performance Analysis of MARS Logging, Checkpointing and Recovery. Proceedings of the 22 Annual Hawaii International Conference on System nd Sciences, 1989, pp. 636-642.
8. Garcia-Molina, H. "Using Semantic Knowledge for Transaction Processing in a Distributed Database". ACM Transactions on Database Systems 8, 2 (June 1983), 186-213.
9. Gray, J. N., Homan, P., Korth, H., and Obermarck, R. A Straw Man Analysis of the Probability of Waiting and Deadlock. Tech. Rept. RJ3066, IBM Research Laboratory, February 1981.
10. Hall, D. L. and Llinas, J. Data Fusion and Multi-Sensor Correlation. Slides for a course.