International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 04 | April - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Difficulties Associated with Replicated Data in Distributed Real-Time Database Systems

Kaynat Zehra Naqvi[1], Dr. Ajay Singh[2], Prof. (Dr.) Pushpneel Verma[3]

[1]Scholar, M.Tech (CSE), [2]Associate Professor & HoD, [3]Professor & Assistant Director

Computer Science & Engineering Department

Bhagwant Institute of Technology, Muzaffarnagar

***Abstract*— In both Distributed and Real Time Databases Systems replication are interesting areas for the new researchers. In this paper, we provide an overview to compare replication techniques available for these database systems. Data consistency and scalability are the issues that are considered in this paper. Those issues are maintaining consistency between the actual state of the real-time object of the external environment and its images as reflected by all its replicas distributed over multiple nodes. We discuss a frame to create a replicated real- time database and preserve all timing constrains. In order to enlarge the idea for modelling a large scale database, we present a general outline that consider improving the Data consistency and scalability by using and accessible algorithm applied on the both database, with the goal to lower the degree of replication enables segments to have individual degrees of replication with the purpose of avoiding extreme resource usage, which all together contribute in solving the scalability problem for Distributed Real Time Database Systems.**

***Keywords*— Replicated database, Replicated Database Design, Replicated database protocols, Transactional replication, Data consistency and Scalability, Active and Passive replication.**

I. INTRODUCTION

Replication is the method of sharing information so as 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. Whereas computation replication if the same computing job is executed many times. A computational job is typically replicated in space, i.e. executed on separate devices, or it could be replicated in time, if it is executed repeatedly on a single device. The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be transparent to an external user. Also, in a failure scenario, a failover of replicas is hidden as much as possible.

A replication set is an identified set of data that exists within a single source, and it corresponds to the movement

set. It is the subset of the particular database that you want to acquire for replication. For example, if you want to replicate data to a laptop for salespersons to use in making daily calls, each person needs a replication set containing the details of the clients that they are going to call on that day. Replication set is made up of a group of replication units. A replication unit is the smallest amount of data that can be identified in a transmission. We try to describe the replication techniques used in both communities, compare them, and points out ways in which they can be integrated to arrive to better, morerobust replication protocols. There are many issues that affect modelling and designing distributed real-time databases. The need to improve the scalability is another important issue.

Synchronization is the process of ensuring that every copy of the database contains the same objects and data. When you synchronize the replicas in a replica set, only the data that has changed is updated. You can also synchronize changes made to the design of the objects in the Design Master. Database writes are sent to the master database server and are then replicated by the slave database servers. Database reads are divided among all of the database servers, which results in a large performance advantage due to load sharing. In addition, database replication can also improve availability because the slave database servers can be configured to take over the master role if the master database server becomes unavailable.[1]

Database replication is the creation and maintenance of multiple copies of the same database. In most implementations of database replication, one database server maintains the master copy of the database and additional database servers maintain slave copies of the database. Database writes are sent to the master database server and are then replicated by the slave database servers. Database reads are divided among all of the database servers, which results in a large performance advantage due to load sharing. In addition, database replication can also improve availability because the slave database servers can be configured to take over the master role if the master database server becomes unavailable.

Database replication can be performed in at least threedifferent ways-

A - Snapshot replication

Data on one database server is plainly copied to another database server, or to another database on the same server. The snapshot replication method functions by periodically sending data in bulk format. Usually it is used when the subscribing servers can function in read-only environment, and also when the subscribing server can function for some time without updated data. Functioning without updated data for a period of time is referred to as latency. Snapshot replication works by reading the published database and creating files in the working folder on the distributor. These files are called snapshot files and contain the data from the published database as well as some additional informationthat will help create the initial copy on the subscription server[2].

B - Merging replication

Data from two or more databases is combined into a single database. Merge replication is the process of distributing data from Publisher to Subscribers, allowing the Publisher and Subscribers to make updates while connectedor disconnected, and then merging the updates between sites when they are connected. Merge replication allows various sites to work autonomously and at a later time merge updates into a single, uniform result. Merge

replication includes default and custom choices for conflict resolution that you can define as you configure a merge publication. When a conflict occurs, a resolve is invoked by the Merge Agent and determines which data will be accepted and propagated to other sites.

C - Transactional replication

Users obtain complete initial copies of the database and then obtain periodic updates as data changes. In transactional replication, each committed transaction is replicated to the subscriber as it occurs. You can control the replication process so that it will accumulate transactions and send them at timed intervals, or transmit all changes as they occur. You use this type of replication in environments having a lower degree of latency and higher bandwidth connections. Transactional replication requires a continuous and reliable connection, because the Transaction Log will grow quickly if the server is unable to connect for replication and might become unmanageable. Transactional replication begins with a snapshot that sets up the initial copy. That copy is then later updated by the copied transactions. You can choose how often to update the snapshot, or choose not to update the snapshot after the first copy. Once the initial snapshot has been copied, transactional replication uses the Log Reader agent to read the Transaction Log of the published database and stores new transactions in the Distribution Database. The Distribution agent then transfers the transactions from the publisher to the subscriber.

Replication has been studied in many areas, especially in distributed systems (mainly for fault tolerance purposes) and in databases (mainly for performance reasons). In these two fields, the techniques and mechanisms used are similar, and yet, comparing the protocols developed in the two communities is a frustrating exercise that many researchers have unsuccessfully attempted. Due to the many subtleties involved, mechanisms that are conceptually identical, end up being very different in practice. As a result, it is very difficult to take results from one area and apply them in the other. In the last few years, as part of the DRAGON project [3], we have devoted our efforts to enhance database replication mechanisms by taking advantage of some of the properties of group communication primitives. We have shown how group communication can be embedded into a database [4, 5, 6] and used as part of the transaction manager to guarantee serilisable execution of transactions over replicated data [7]. We have also shown how some of the overheads associated with group communication can be hidden behind the cost of executing transactions, thereby greatly enhancing performance and removing one of the serious limitations of group communication primitives [8]. This work has proven the importance of and the need for a common understanding of the replication protocols used by the two communities. In this paper, we present a model that allows comparing and distinguishing existing replication protocols in databases and distributed systems. We start by introducing a very abstract replication protocol representing what we consider to be the key phases of any replication strategy. Using this abstract protocol as the base line, we analyse a variety of replication protocols from both databases and distributed systems, and show their similarities and differences. With these ideas, we parameterize the protocols and provide an accurate view of the problems addressed by each one of them. Providing such a classification permits to systematically explore the solution space and give a good baseline for the development of new protocols. While such work is conceptual in nature, we believe it is a valuable contribution since it provides a much needed

perspective on replication protocols. However, the contribution is not only a didactic one but also eminently practical. In recent years, and in addition to our work, many researchers have started to explore the combination of database and distributed system solutions [9, 10, 11, 12]. The results of this paper will help to show which protocols complement each other and how they can be combined.

In any real-time applications are inherently distributed in nature, and need to share data that are distributed among different sites. For example, military tracking, medical monitoring, naval combat control systems and factory automation etc. Such applications introduce the need for distributed real time database systems (DRTDBs) [13]. A DRTDBS is a collection of multiple, logically interrelated databases distributed over a computer network [14]. A real- time database has two distinguishing features: the nature of its data that has a temporal constrains, and marinating a real- time constraints on transactions [15]. Transactions in a real time database are classified into three types, viz. hard, softand firm.

Early work on replicated data in distributed database systems focused on providing replication transparency which requires tight consistency of replicated data .Implementation typically required that some number of copies of replicated data be updated atomically.[16,17,18] Loose consistency of replicated data has become the focus of recent research work. Alternatives transaction correctness criterion and different degrees of controlled inconsistency have been introduced [19, 20]. These proposals all require user input in handling temporary inconsistency of replicated data.However,the idea of snapshot was introduced very early[21]. It is interestingthat it has just been implemented in some commercially available systems.

II. CONCEPT AND ABSTRACTION IN DATABASE REPLICATION

A-Concept

Towards improved appreciate the technique behind Database Replication we start with the term “Replication” which represents the process of sharing information to ensure consistency between redundant resources, such as software orhardware components, to improve reliability, fault-tolerance, or accessibility. It could be data replication if the same data is stored on multiple storage devices or computation replication if the same computing task is executed many times [1].

Database mirroring can be used to improve availability of certain replication databases. Support for combining transactional replication with database mirroring depends on which replication database is being considered. Peer-to-Peer replication is not supported in combination with databasemirroring. The replication agents that connect to the publication database can automatically fail over to the mirrored publication database. In the event of a failure the agents that connect to the publication database will automatically reconnect to the new principal database.

In a replication building block, the source is generally a database that contains data to be replicated. One database can be the source for many replication building blocks. The source database can also serve as the target for another replication building block. For example, in the Master-MasterReplication pattern, the same pair of data stores swaps roles (source becomes target, and target becomes source) for a common movement set that is updateable in either data store.

Replication is the process of sharing information so as to ensure 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 or computation replication if the same computing task is executed many times. A computational task is typically replicated in space, i.e. executed on separate devices, or it could be replicated in time, if it is executed repeatedly on a single device. The access to replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be transparent to an external user. Also, in a failure scenario, a failover or replicas is hidden as much as possible.

Replicated is the key characteristic in improving the availability of data distributed real time systems. Replicated data is stored at multiple server sites so that it can be accessed by the user even when some of the copies are not available due to server/site failures [22]. A Major restriction to using replication is that replicated copies must behave like a single copy, i.e. mutual consistency as well internal consistency must be preserved, Synchronization techniques for replicated data in distributed database systems have been studied in order to increase the degree of consistency and to reduce the possibility of transaction rollback [23].

In replicated database systems, copies of the data items can be stored at multiple servers and a no. of places. The potential of data replication for high data availability and improved read performance is crucial to DRTDBS. In contrast, data replication introduces its own problems. Access to a data item is no longer control exclusively by a single server; instead,the access control is distributed across the servers each storing the copy of the data item. It is necessary to ensure that mutual consistency of the replicated data is provided; it must fulfill the ACID properties of database.

It is common to talk about active and passive replication in systems that replicate data or services. Active replication is performed by processing the same request at every replica. In passive replication, each single request is processed on a single replica and then its state is transferred to the other replicas. If at any time one master replica is designated to process all the requests, then we are talking about the primary-backup scheme (master-slave scheme) predominant in high-availability clusters. On the other side, if any replica processes a request and then distributes a new state, then this is a multi-primary scheme (called multi-master in the database field). In the multi-primary scheme, some form of distributed concurrency control must be used, such as distributed lock manager.

Load balancing is different from task replication, since it distributes a load of different (not the same) computations across machines, and allows a single computation to be dropped in case of failure. Load balancing, however,sometimes uses data replication especially for multi-user internally, to distribute its data among machines [24].

B- Abstraction

We regard as a distributed system modelled as a set of services implemented by servers processes and invoked by clients processes. The specification of the service defines the set of invocations that can be issued by the clients. Each server process has a local state that is modified through invocations. We consider that invocations modify the state of a server in an atomic way, that is, the state changes resulting from an invocation are not applied

partially. The isolation between concurrent invocations is the responsibility of the server, and is typically achieved using some local synchronization mechanism. This model is similar to "one operation" transactions in databases (e.g., stored procedure). In order to tolerate faults, services are implemented by multiple server processes or replicas.

The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be transparent to an external user. In addition, in a failure scenario, a failover of replicas is hidden as much as possible. In systems that replicate data the replication itself is either active or passive.

On the way to cope with the complexity of replication, the notion of group (of servers) and group communication primitives have been introduced [25]. The notion of group acts as a logical addressing mechanism, allowing the client to ignore the degree of replication and the identity of the individual server processes of a replicated service. Group communication primitives provide one-to-many communication with various powerful semantics. These semantics hide much of the complexity of maintaining the consistency of replicated servers. The two main group communication primitives are Atomic Broadcast (ABCAST) and View Synchronous Broadcast (VSCAST). We give here an informal definition of these primitives. A more formal definition of ABCAST can be found in [26] and of VSCAST can be found in [27] (see also [28,29]). Group communication properties can also feature FIFO order guarantees..

We converse about an active replication when the same request is processed at every replicated instance and about passive replication when each request is processed on a single replica and then its state is transferred to the other replicas. If at any time one master replica is designated to process all the requests, then we are talking about the primary-backup scheme (master slave scheme) predominant in high- availability clusters. On the other side, if any replica processes a request and then distributes a new state, then this is a multi-primary scheme (called multi-master in thedatabase field). Even thought the process of Data Replication it's used to create instances of the same or parts of the same data, we must not confuse it with the process of backup since replicas are frequently updated and quickly lose any historicalstate. Backup on the other hand saves a copy of data unchanged for a long period of time.

C- Active and passive Replication

Active replication, also called the state machine approachis a non-centralized replication technique. Its key concept is that all replicas receive and process the same sequence ofclient requests. Consistency is guaranteed by assuming that, when provided with the same input in the same order, replicas will produce the same output. This assumption implies that servers process requests in a deterministic way. Clients do not contact one particular server, but address servers as a group. In order for servers to receive the same input in the same order, client requests can be propagated to servers using an Atomic Broadcast. Weaker communication primitives can also be used if semantic information about the operation is known (e.g., two requests that commute do not have to be delivered at all servers in the same order) [30].

The main advantage of active replication is its simplicity (e.g., same code everywhere) and failure transparency. Failures are fully hidden from the clients, since if a replica fails; the requests are still processed by the other replicas.The determinism constraint is the major drawback of this approach.

Passive replication, also called Primary Backup replication, is that clients send their requests to a primary, which executes the requests and sends update messages to the backups. The backups do not execute the invocation, but apply the changes produced by the invocation execution atthe primary i.e., updates. By doing this, no determinism constraint is necessary on the execution of invocations, the main disadvantage of active replication. Communication between the primary and the backups has to guarantee that updates are processed in the same order, which is the case if primary backup communication is based on FIFO channels. However, an only FIFO channel is not enough to ensure correct execution in case of failure of the primary. For example, consider that the primary fails before all backups receive the updates for a certain request, and another replica takes over as a new primary. Some mechanism has to ensure that updates sent by the new primary will be "properly" ordered with regard to the updates sent by the faulty primary. VSCAST is a mechanism that guarantees these constraints, and can usually be used to implement the primary backup replication technique [31].

Passive replication can tolerate non-deterministic servers (e.g., multi-threaded servers) and uses little processing power when compared to other replication techniques. However, passive replication suffers from a high reconfiguration cost when the primary fails.

D- Circumstance of Database Replication

Replication Techniques in Distributed Systems organizes and surveys the spectrum of replication protocols and systems that achieve high availability by replicating entities in failure-prone distributed computing environments. The entities discussed in this book vary from passive un-typed data objects, to type and complex objects, to processes and messages.

Replication Techniques in Distributed Systems contains definitions and introductory material suitable for a beginner, theoretical foundations and algorithms, an annotated bibliography of commercial and experimental prototype systems, as well as short guides to recommended.

The transaction level data can be duplicated to the replica database. The result is greater data integrity and availability .However, the increased availability is dependent on how independent the database replica is from the primary database. Replica independence must be considered in termsof disk spindles, disk controller, power supplies, system, room, building and city.

While data copying can provide users with local and much quicker access to data, the problem is to provide these copies to users so that the overall systems operate with the same integrity and management capacity that is available within a centralized model. Managing a replicated data is significantly more complicated than running against a single location database. It deals with all of the design and implementation issues of a single location and additionally with complexityof distribution, network latency and remote administration [32].

III. AN OVERVIEW OF REPLICATED DATABASE SYSTEM AND DESIGN

Database replication is the process of creating and maintaining multiple instances of the same database and the process of sharing data or database design changes between databases in different locations without having to copy

the entire database. In most implementations of database replication, one database server maintains the master copy of the database and the additional database servers maintain slave copies of the database. The two or more copies of a single database remain synchronized. The original database is called a Design Master and each copy of the database is called a replica. Together, the Design Master and the replicas make up a replica set. There is only one Design Master in a replica set [1].

The Real-time replicated database designs the relationships between its objects and the external environment from time point of view. We study a distributed real-time replicated database system which consists of a group of main memory real-time replicated databases connected by high-speed networks. We assume that a reliable real-time communication is maintained, i.e., any messages sent over the network is eventually delivered and have predictable transmission time. According to the distributed nature of the database, the objective is to give the clients the illusion of service that is provided by one server and the clients have no knowledge about the data existence behind. Traditional RDBMS are based on the assumption that data resides primarily on disk. In a dynamic runtime environment, data might be on disk or cached in main-memory at any given moment. Because disk input/output (I/O) is far more expensive than memory access, main memory databases have been used because of the high performance of memory accesses and the decreasing main memory cost [33].

A replicated database system using the system replication server has four types of components:

1. The database servers which store copies of the replicated data.
2. The log transfer managers which exact transaction log records from a database containing primary copies of data and submit them to the replication server.
3. The Sybase replication servers which accept the transaction log records from log transfer managers and apply them to database containing replicate copies of the data.
4. The database servers which store the system catalogue information used by the replication servers.

A replicated database system may have many servers of the above types that are inter-connected by networks. This section uses an example to represent the design of a replicated database system. A replicated data base system with two replication servers RS1 & RS2.The two database DB1 & DB2 contain primary data (in add to replicate data).they are called primary database. The database DB3 contains only replicate data. It is known as replicate database. Each database is controlled by exactly one replication server. Of course, replication servers communicate with the database through data servers. The data servers are not included in the picture for simplicity. Each replicated database containing primary copies of data must have a log transfer manager. The log transfer manager is a database server dependent process which understands how to extract the log records from the database transition log. The DB1 & DB2 have log transfer managers LTM1 & LTM2 connected to them respectively. The DB3 contains replicate data only. It does not need a log transfer manager. [The RS1 and RS2 is a two replication servers and DB1, DB2 & DB3 are databases contain primary data and replicate data. LTM1 & LTM2 are log transfer managers they are connected to DB1 & DB2 respectively.[34]

Users must define to the replication server the location of the primary copy of a index containing replicated data as well as the columns and their data types in the index. These definitions of replicated indexes are called replication

definitions. Users can then specify they want the replicate copies of the index. This process is called subscriptions. Subscriptions can be entered dynamically. Information about replication definitions and subscriptions as well as other replication system information are stored in databases called replication server system databases.

IV. ISSUES IN DISTRIBUTED REAL TIME REPLICATED DATABASE SYSTEMS

Many issues affecting the design of A DRTDBS to maintain its requirements; Data Consistency and Scalability are the main issues that are considered in this paper. All of those critical systems need data to be obtained and updated ina timely fashion, but sometimes data that is required at a particular location is not available when it is needed and getting it from remote site may take too long before whichthe data may become invalid, this potentially leads to large number of transactions miss 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. 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. Replication in DRTDBs also, 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 themain-memory resident data. In order to suite the different needs of the distributed real-time systems such as different data workloads and database specifications, multiple ways to handle the replication control and different replication schemes are proposed. However, such a database has another scalability problem since an update to an object of a fully replicated database needs to be sent to all other nodes.

A-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. These concepts are elaborated in the following two paragraphs-

1) -Immediate consistency

Pessimistic techniques for consistency preservation ensure that all replicas of all logical database objects accessed by a transaction are consistent when that transaction commits, i.e., consistency is immediately achieved. There are several well- defined techniques for this, the most basic and well known of which is the two-phase commit protocol [35].

2)- Eventual consistency

Optimistic consistency preservation techniques are based on the concept of eventual consistency. The idea is to trade off immediate consistency at each transaction commit for increased predictability, availability, and performance. This means that a transaction may commit updates to a local replica of a logical database object without propagating the updates to the nodes containing additional replicas of that object. Thus, the local performance and availability are increased, since all overhead associated with communication protocols and

distributed commit protocols is removed. Examples of optimistic consistency preservation protocols are Distributed Optimistic Two-phase Locking [36] and Lazy Replication [37].

B- 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. For example, in a trading application it is a fundamental requirement to inform the customer if a bid is successfully placed or not, an RDBMS may suffice. Any time a transaction fails due to one database instance failure then the entire transaction is abandoned. The data updated at one instance is either available in all the instances or it is not. For web-based applications, ACID properties are too restrictive to satisfy the associated business needs due to surges in user activity; hence a different set of properties must be established to achieve scalability and performance.

C- Design issues

Replication set size. Decide 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.

Transmission volume Choose the right amount of data to transmit. The decision between sending all changes for any one row, or just the net effect of all the changes, is a key one.

Replication set data changes at the target. If these have to occur and if the source wants to see the changes, then try to make the changes naturally non-conflicting to avoid the need for conflict detection and resolution.

Replication frequency decides the appropriate timing of the replication for the requirements and optimizes the use of computing resources.

Replication unit defined earlier, a replication set consists of a group of replication units. Identify the unit of data that will be transmitted from the source to the target. In the extreme requirements, this will be a transaction as it has been executed on the source. A less precise but easier to achieve requirement is to move a changed row. For environments with a high risk of conflicts, it can also be an individual change in a cell within a record.

Initiator Decide whether the source pushes the data or the target pulls it, and make sure that throughout your replication topology these decisions do not cause later replication links to have problems meeting their operational requirements.

Locking issues verify that you can accept the locking impact of the replication on the source. If not, verify that a slight decrease in consistency at a point in time is acceptable for the targets so you can avoid lock conflict.

Replication topology Identify the players, their roles, and the overall integrity.

Security ensures that the replicated data is treated with the right level of security at the target given the source security conditions. Also, verify that your replication link is secure enough in the overall topology requirements.

Key updates. Verify whether the source allows updates to the key of records belonging to the replication set. If so, special care must be taken for a consistent replication of such operations. Key updates are SQL updates to the columns of the physical key within a replication set. Such key updates must be handled specially by the replication.

Referential integrity verifies whether the target has implemented referential integrity. If so, you need rules to prevent changes from the replication link being applied twice if the change triggers a target change in another replicated table.

The real-time scheduling part of our scheme has three components: a policy to determine which transactions are eligible for service, a policy for assigning priorities to transactions, and a policy for resolving conflicts between two transactions that want to access the same data object. None of these policies needs any more information about transactions than the deadline and the name of the data object currently being accessed.

Transactions that are unable to meet their deadlines are immediately aborted. When a transaction is accepted for service at the local site where it was originally submitted, it is assigned a priority according to its deadline. The transaction with the earliest deadline has the highest priority. High priority is the policy that we employed for resolving transaction conflicts. Transactions with the highest priorities are always favoured. The favoured transaction, i.e. the winnerof the conflict, gets the resources that it needs to proceed.

D- Conflict Resolution

Earlier than presenting the conflict resolution scheme ofthe algorithm, we first outline the token-based approach for comparison. A detailed description of the token based approach is presented in [38]. In token-based approach, a token designates a read-write copy that contains the latest version of a data object. The site which has a token copy of a logical data object is called a token site, with respect to the logical data object. Each data object is associated with two values; the after-value and the before-value. The system remembers the before value for the duration of the transaction that performs an update operation on the data object, so that it can be restored if the transaction is rolled back. The transaction managers that have been involved in the execution of a transaction are called the participants of the transaction. The coordinator is one of the participants which initiates and terminates the transaction by controlling all otherparticipants.

Towards read a data object X, the coordinator checks first whether its read operation conflicts with the write operation that has already been issued by another transaction on the same data object. In case such a conflict occurs, coordinator waits for the termination of the update transaction or reads the before-value of X and proceeds, depending on whetherthe conflicting transaction is an older or a younger one respectively. For a write operation to be processed, the coordinator checks for conflicts with other read or write operations. In case that there is a conflict with an older transaction, current update transaction has to be aborted. If it conflicts with a younger transaction, coordinator will have either to wait for the younger transaction to terminate or will proceed but not terminate before the other transaction terminates. After all conflicts, if any, are resolved the value of the data object to be written is broadcast to all token sites where a token-copy of the data object resides. A logical write operation is considered completed when the update request messages are sent. When a write operation is successfully performed and the transaction is committed, a new version is created which replaces the previous version of the token copy. A transaction commits when all token-sites of each data object in the write-set of the transaction have recommitted, each data object in the read-set of the transaction is read, and there is no active

transaction that has seen before-value of any data object in the transaction's write-set. The majority consensus approach is different from the token-based approach in that all the data objects in a transaction's read-set are read at one time. Hence in this approach, two transactions conflict if the intersection of one transaction's read-set and the other transaction's write-set is not empty. In the token- based approach, even though one transaction's read-set overlaps with the other transaction's write-set, these two transactions may not conflict. Because data objects are read and written one at a time, a transaction might be terminated before the other transaction reads or writes the common data objects. In terms of conflicts between two transactions, themajority consensus approach inherently allows less concurrency than the token-based approach.

A transaction execution in the majority consensus approach is divided into two phases: voting phase and commit phase. A transaction remains in the first phase if it is not accepted. It moves into the second phase after it is accepted, but before it commits. A transaction can only be aborted when it is in the first voting phase. A transaction in the second phase cannot be aborted in order to preserve the database consistency. In a real-time database system, a transaction is aborted whenever it is found to have missed its deadline. The transaction manager for each transaction periodically checks whether or not the transaction will be able to meet its deadline taking into consideration the fact that the transaction has to update the data objects in its write set at each database. If the system's current time plus the time to update all data objects in a transaction's write-set is greater than the transaction's deadline, it means that this transaction will not be able to commit before its deadline is reached. In order not to waste any system resources, the transaction will be aborted and removed from the system.

If a transaction is decided to be eligible for system service by the above screening process, it is assigned a priority basedon its deadline. For two conflicting transactions, the transaction with more urgent deadline is assigned a higher priority. Compared with token-based approach, the conflict resolution policy for the majority consensus approach is simpler. Conflicts between two transactions are based on the intersection of their read-sets and write-sets. Another reason for the simple conflict resolution policy is that a transaction's time-stamp does not contribute to the resolution of conflicts. The only deciding factor in a conflict is the transaction's deadline. Whereas in the real-time token-based approach, both the transaction's deadline and time-stamp are factors in resolving a conflict.

Real-time database systems support applications which have severe performance constraints such as fast response time and continued operation in the face of catastrophic failures. Real-time database systems are still in the state of infancy, and issues and alternatives in their design are not very well explored. In this paper, we discuss issues in the design of real-time database systems and discuss different alternatives for resolving these issues. We discuss the aspects in which requirements and design issues of real-time database systems differ from those of conventional database systems. We also discuss requirements in the design of real-time distributed database systems, and specifically discuss issues in the design of concurrency control and crash recovery. It is felt that long communication delays may be a factor in limiting the performance of real-time distributed database systems.

Replication in Distributed real-time database systems also, 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. In order to suite the different needs of the distributed real-time systems such as

different data workloads and database specifications, multiple ways to handle the replication control and different replication schemes are proposed. However, such a database has another scalability problem since an update to an object of a fully replicated database needs to be sent to all extra nodes.

V. CONCLUSION

A universal schema to model a Replicated Distributed Real-Time Database Systems has been designed for small and medium scale distributed real-time database systems. Both data and transactions timing constrains are maintained. The accessible frame has been absolute to model extensive database systems. This has been done by presenting a general outline that considers getting better the data consistency and scalability by using an existing static segmentation algorithm applied on the entire database on node allocation. It is desirable to implement this frame on a distributed real-time database system and evaluate it with real transactions requirements. We are presently using a replication control algorithm. Frame to model real-time replicated database will be used as a frame for our real-time database, which is not completely supported for distributed database. We are also looking into implementing the segmentation part as a contribution in solving the data consistency and scalability problem of distributed real-time database systems. The design issues raise in this paper is the challenges for new researchers. We are reviewing the issues and strive to develop some protocols for distributed real time databases which improvethe performance of replicated database.

REFERENCES

[1] Marius Cristian MAZILU, "Database Replication", Database Systems Journal Vol 1, no. 2/2010.

[2] Mark A.Linsenbardt,Shane Stigler “McGraw-Hill/Osborne Media BookSQL Server2000Administration”-Chap.10,’.Replication’.

[3] Information & Communcations Systems Research Group, ETH Zürich and Laboratoire de Systèmes d’Exploitation (LSE), EPF Lausanne. DRAGON: Database Replication Based on Group Communication, May 1998.

[4] D. Agrawal, G. Alonso, A. E. Abbadi, and I. Stanoi. “Exploiting atomic broadcast in replicated databases“. In Proceedings of EuroPar (EuroPar’97), Passau (Germany), 1997.

[5] F. Pedone, R. Guerraoui, and A. Schiper.“Transaction reordering in replicated databases“. In Proceedings of the 16th Symposium on Reliable Distributed Systems (SRDS- 16), Durham, North Carolina, USA, Oct. 1997.

[6] F. Pedone, R. Guerraoui, and A. Schiper.“Exploiting atomic broadcast in replicated databases“. In Proceedings of EuroPar (EuroPar’98), Sept. 1998.

[7] B. Kemme and G. Alonso.“A suite of database replication protocols based on group communication primitives“. In Proceedings of the 18th Internationnal Conference on Distributed Computing Systems (ICDCS), Amsterdam, The Netherlands, May 1998.

[8] B. Kemme, F. Pedone, G. Alonso, and A. Schiper.“Processing transactions over optimistic atomic broadcast protocols“. In Proceedings of the International Conference on Distributed Computing Systems, Austin, Texas, June 1999. to appear.

[9] M. Raynal, G. Thia-Kime, and M. Ahamad.“From serializable to causal transactions for collaborative applications“. Technical Report 983, Institut de Recherche en Informatique et Systèmes Aléatoires, Feb. 1996.

[10] I. Stanoi, D. Agrawal, and A. E. Abbadi.“Using broadcast primitives in replicated databases“. In Proceedings of the International Conference on Distributed Computing Systems ICDCS’98, pages 148–155, Amsterdam, The Netherlands, May 1998.

[11] E. Pacitti, P. Minet, and E. Simon.“Fast algorithms for maintaining replica consistency in lazy master replicated databases“. In Proceedings of the 25th International Conference on Very Large Databases, Edinburgh - Scotland - UK, 7–10 Sept. 1999.

[12] J. Holliday, D. Agrawal, and A. E. Abbadi.“The performance of database replication with group multicast“. In Proceedings of IEEE International Symposium on Fault Tolerant Computing (FTCS29), pages 158–165, 1999.

[13] Lee Juhnyoung, “Concurrency Control Algorithms for Real-time Database Systems”, PhD Thesis, Department of Computer Science, University of Virginia, 1994.

[14] Y-W.Chen, and L.Gruenwald, “Effects of Deadline Propagation on Scheduling Nested Transactions in Distributed Real - Time Database Systems,” Journal of Information Systems, Vol. 21, No. 1, pp. 103 - 124, 1996

[15] K.Ramamritham and C.Pu. "A Formal Characterization of Epsilon Serialisability". IEEE Transaction Journal on Knowledge and Data *Engineering, vol 7, no.6, pp.:997–1007, December1995.*

[16] P.A.Bemstein, N.Goodman:“An algorithm for concurrency control and recovery in replicated distributed databases“.ACM Transactions on database systems 9,4(december1984)

[17] P.A.Bemstein, V.Hadzilacos,N.Googman:“Concurrency Control and Recovery in Databases Systems“. Addition- Wesley,1987.

[18] D.Skeen:“Nonblicking commit protocol“s.Procedding of ACM SIGMOD Conference on management of data,1982.

[19] A.R.Downing,I.B.Greenberg,J.M.Peha:PSCAR:“A System for week- consistency replication“.Proceedings of the workshop on the management of r eplicated data,1990.

[20] C.Pu,A.Leff::“Replica control in distributed systems:An Asynchronous“ Approach.Proceeding of ACM SIGMOD Conference on management of data,1991.

[21] M.E.Adiba,B.G.Lindsay:“Database Snapshots“.Proceedings of th VLDB,1980.

[22] E. Leontiadis, V.V. Dimakopoulos and E. Pitoura.“Creating and Maintaining Replicas in Unstru ctured Peer-to-Peer Systems.“, EURO-PAR 2006

[23] Sang Hyuk Son, "Replicated Data Management in Distributed Database Systems", ACM Simgod,, Vol. 17, Issue 4, Dec 1998, Newyork,USA, pp-62-69.

[24] E. Leontiadis, V.V. Dimakopoulos and E. Pitoura.“Creating and Maintaining Replicas in Unstru ctured Peer-to-Peer Systems“, EURO-PAR 2006.

[25] K. P. Birman.“The process group approach to reliable distributed computing.“. Communications of the ACM, 36(12):37–53, Dec. 1993.

[26] V. Hadzilacos and S. Toueg.“ Fault-tolerant broadcasts and related problems.“ In S. Mullender, editor,

Distributed Systems, chapter 5. adwe, second edition, 1993.

[27] A. Schiper and A.“Sandoz. Uniform reliable multicast in a virtually synchronous environment“. In Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS-13), pages 561–568, Pittsburgh, Pennsylvania, USA, May 1993. IEEE Computer Society Press.

[28] K. P. Birman and T. A. Joseph. „Exploiting virtual synchrony in distributed systems“. In Proceedings of the 11th ACMSymposium on OS Principles, pages 123–138, Austin, TX, USA, Nov. 1987. ACM SIGOPS, ACM.

[29] K. P. Birman, A. Schiper, and P. Stephenson.“Lightweight causal and atomic group multicast.“ ACM Transactions on Computer Systems, 9(3):272–314, Aug. 1991.

[30] F. B. Schneider.“Implementing fault-tolerant services using the state machine approach“: A tutorial. ACM Computing Surveys, 22(4):299– 319, Dec. 1990.

[31] R. Guerraoui and A. Schiper.“Software-based replication for fault tolerance“. IEEE Computer, 30(4):68–74, Apr. 1997.

[32] G.Schussel, "Replication the nexy generation of Distributed Databasetechnologyvia Internet“ www.dci.com/speaker/archive/replica
.html.

[33] J.Baulier, P.Bohannon, S.Gogate, C.Gupta, S.Haldar, "DataBlitz storage manager: Main memory database performance for critical applications", in Proc. ACM SIGMOD international conference on Management of data, vol 28, no.2, pp.519–520, June 1999.

[34] Yongdong Wang, Jane Chiao, "Data Replication in a distributed hetrogenous database environment: An open System Approach", IEEE, 1994.

[35] Gray, J. & Reuter, A. (1993), Transaction Processing: Concepts and Techniques, Morgan Kaufmann, chapter 10.

[36] [Carey, M & Livny, M. (1991), Conflict detection tradeoffs for replicated data, ACM Transactions on Database Systems 16(4), 703- 746.

[37] Breitbart, Y. & Korth, H. (1997), Replication and consistency: Being lazy helps sometimes, in Proceedings of the Sixteenth ACM SIGACT- SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14,Tucson, Arizona.

[38] S.H.Son and S.Kouloumbis,”AToken Based synchronization scheme for distributed real-time databases”,information systems, vol.18,no.6,dec.1993,pp 375-389.