IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 6, Ver. I (Nov.- Dec. 2017), PP 41-46
www.iosrjournals.org

DynamicData Distribution for Merge Replication in Databases

Saadi Hamad Thalij[1], Veli Hakkoymaz[2]
Dept. of Computer Eng /Faculty of Electrical & Electronic Engineering.
Yildiz Technical University /Istanbul/ Turkey

Abstract:*Database replication is performed to back up the data in second server in order to provide efficient data to the required users. However there are numerous issues in the database replication process. In this paper, some of the major database replication issues are discussed in general followed by the presentation of a case study for merge replication. Specifically, different replication types, replication topologies and replication agents are discussed and the details about merge replication are provided. In particular, the necessary conditions for merge replication and the methods for data transfer among multiple servers are examined. For illustrating the case study in simple terms, only one server is used to replicate data. The components of replication process are the publisher, subscriber and distributor. Database tables reside on the same location. Several conflicts are generated in the replication case study. This paper reports how these conflicts occur and also show the synchronization status between the publisher and the subscriber. Finally, the status of various agents is shown. Thus, the paper aims at developing a novel case study to demonstrate the merge replication and has succeeded in its objective.*

Keywords: *Database,Replication, Merge Replication, Dynamic Data Transfer, DBMS*

Date of Submission: 23-10-2017 Date of acceptance: 04-11-2017

I. Introduction

Nowadays, many applications requiremaking data that live on one server available on other servers. There are many reasons for such requirement. One reason is to speed up cross-server queries by providing a local copy of the data.Another is to make the data available to resource intensive reporting queries without impacting the online transaction processing load. Replication is the process of distributing same data to multiple sites while maintaining the consistency when changes occur in any one of these copies. To do this, it is necessary to create and maintain multiple instances of the same database objects and share data among databases in different locations. Its obvious advantages are high availability, improved reliability, fault-tolerance and accessibility [1].

We define and maintain a process with replication whereby data is copied between databases on the same server or on different servers connected by the Internet [2]. Database replication is a technique through which an instance of a database is exactly copied to, transferred to or integrated with another location. Database replication is primarily used in distributed DBMS environments where a single database is deployed, used and updated simultaneously at several locations. Database replication is generally performed frequently in a transactional database that is routinely and dynamically updated. Typically, database replication is done to provide a consistent copy of data across all the database nodes [3]. Once a replica is changed other replicas must reflect this change. Synchronization is the process in which all replicas eventually become exactly the same by propagating in one replica to the other.

Rest of the paper is organized as follows: In section two, we give typical components of replication. In section three, we introduce replication types, topologies and agents. In section four, merge replication is explained in detail. In section five, conflicts in merge replication are described. Experimental results and discussion are given in section six. Finally, section seven concludes.

II. Typical Replication Components

Replication requires a number of components (i.e., processes) to work properly. Fig.1 shows a high-level overview of the pieces involved in a replication setup [1]. This setup includes a publisher and its publication database, a distributor and its distribution database, and several subscribers and subscription databases. It also includes the replication agents necessary to drive these processes.

The first component is the *publisher* which stores source database (i.e., which data are available for replication).It defines what is published through a publication (the actual database objects like tables, views, indexes and so on). The second component is the *distributor*. It is the intermediary between the publisher and subscribers. It receives published databases (i.e., snapshots) and stores it in distribution database. It then forwards these snapshots to the subscribers. The third component is the *subscriber*.It is the destination

databasewhere replication ends. Each subscriber can subscribe to multiple publications. It can send data back to publisher so as to publish data updates to other subscribers.*Subscription*is a request by a subscriber to receive a publication. There are two types of subscriptions: In *push subscription, the* publisher is responsible for sending all the changes to the subscribersafter a subscriber makes some changes. The*push subscriptions*are created at the publisher server. In *pull subscriptions*,the subscribers initiate the replication instead of the publisher [4].

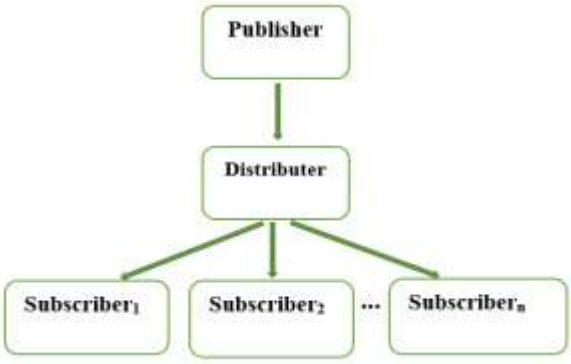

Fig.1. The components that make up a replication setup

III. Replication Types, Topologies and Agents

3.1 Replication Types

In general, the following replication typesare supported by many distributed database management systems (DBMS): i) snapshot replication, ii) transactional replication, iii) merge replication.

Snapshot replication is the simplest type of replication in which, all replicated data (replicas) will be copied from the publisher database to the subscribers' database(s) on a periodic basis. Snapshot replication is used for such replication applications that require infrequent changes on data and the size of replicated data not very large.

In *transactional replication*, all database changes (insert, update, and deletestatements) are captured and stored in the distribution database. These changes are then sent to subscribers from the distribution database and applied in the same order. The *transactional replication* is used mainly for such replication applications that data changes frequently and the size of replicated data is not small. Automatic changes of the replicated data on the publisher and on subscribers are not needed in transactional replication.

Merge replication is considered as the most difficult replication type. It makes possible automatic changes to replicated data on the publisher and on the subscribers. With merge replication, all incremental data changes are captured in the source and in the target databases.Conflicts can occur while several updates are merged into a single copy in the database. It is defined as the attempt to change the same data records by multiple actors simultaneously.Since updates are made at more than one server, the same data may be updated by the publisher or the subscribers. Conflicts are resolved according to rules in configuration file or using a custom resolver. Merge replication is used mainly for such applicators in which automatic changes of replicated data are supported on the publisher and on the subscribers [5].

3.2 Replica Topology

With topology, we mean that what the components are in the replication system, where they are located and what the interactions are like. In the hierarchical model, the replicas are placed at different levels, and they communicate with each other in a client-server manner. This model has been implemented in many replication systems [6].Following are some replication topologies: central publisher, central subscriber, and central publisher with remote distributor, central distributor and publishing subscriber. Due to space limit, we just show central publisher topology in this paper, other topologies are explained in [9].

3.2.1 Central Publisher

This is one of the most used replication topologies, as shown in Fig. 2. In this topology, thepublisher and distributor are located on server and other processes are configured as subscribers[5].

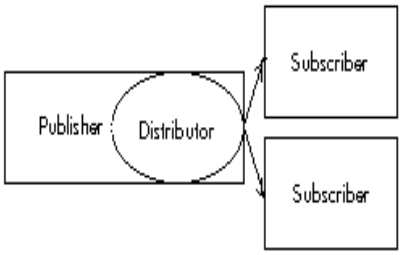

Fig.2. The central publisher

3.3 Replication Agents

The following replication agents are defined and available to help manage the replication system work: snapshot agent, log reader agent, distribution agent and merge agent.

The Snapshot Agent is a replication agent that makes snapshot files, stores the snapshot on the distributor, and records information about the synchronization status in the distribution database. It is used in all replication types (snapshot, transactional, and merge).

The Log Reader Agent is a replication agent that moves transactions marked for replication from the transaction log on the Publisher to the distribution database. It is not used in snapshot replication.

The Distribution Agent is a replication agent that moves the snapshot jobs from the distribution database to subscribers, and moves all transactions waiting to be distributed to subscribers. The distribution agent is used in snapshot and transactional replications.

The Merge Agent is a replication agent that applies initial snapshot jobs from the publication database tables to subscribers, and merges incremental data changes that have occurred since the initial snapshot was created. It is used only in merge replication [7].

IV. Merge Replication

Merge replication allows changes to be sent from one primary server, called a publisher, to one or more secondary servers, called subscribers. Merge replication is used for distributing data to various servers from a primary server. One important criterion is how frequently changes are made to it. Merge replication is the most complex type of replication because it allows both publisher and subscriber to independently make changes to the database. In this scenario, strictly speaking the publisher doesn't look like the primary server, because other servers can also make changes to the data. At any rate, the changes are then synchronized by merge agents that sit on both servers, as well as by a predetermined conflict resolution mechanism in case of clashing data changes. Such clashes may arise because merge replication does not require a real-time network connection between the publisher and the subscriber. It is possible that one server changes data and later on another server changes the same data to a different value [3].

Merge replication is commonly used by laptop and mobile users who cannot be constantly connected to the publisher, but still need to carry around a copy of the database that they can make changes to.A merge replication setup includes the following components') snapshot agent, ii) triggers, tables, and views and iii) merge agent.The Snapshot agent in merge replication plays the same role as in transactional replication. It creates a snapshot of the data in the publication database, which is used to initialize a subscriber. In merge replication, the role that the log reader agent fills in transactional replication is taken by a set of triggers, tables, and views. These objects are added to each subscription database as well as to the publication database. The merge agent applies the collected changes to the subscriber. Because merge replication isbi-directional, the merge agent also applies changes to the publisher [1]. This is shown in Fig. 3.

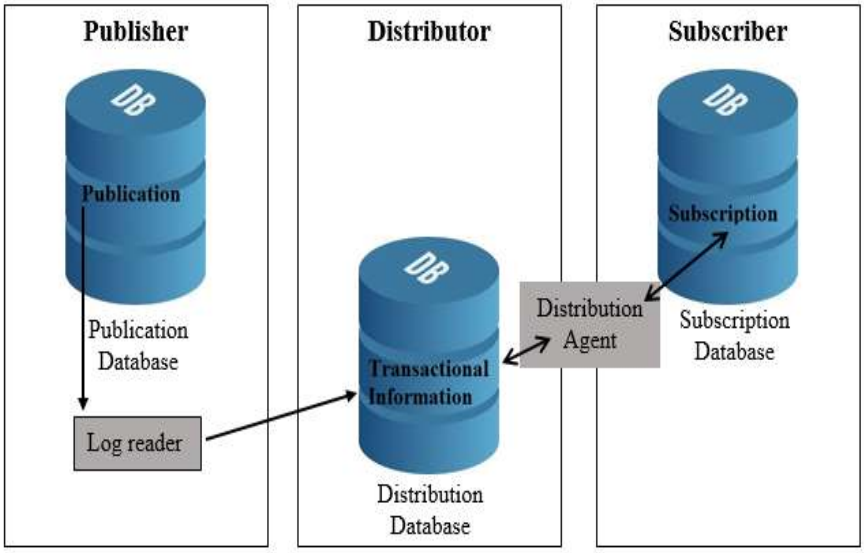

Fig.3. Merge replication agents and triggers

V. Conflicts

When the same data are modified at two different locations then we say that a *conflict* has occurred. There are several reasons for conflicts. A conflict occurs due to the fact that a row is changed on one node (publisher or subscriber) and then changed on another node. This type of conflict is called an *update-update* conflict. However, if a row is updated on one node, but deleted on another node, the resulting conflict is called an *update-delete* conflict. A conflict can also occur if a change that is applied to one node cannot be applied to another node. For example, a successful update on one subscriber might result in a constraint violation on another subscriber. This type of conflict is called a *failed-change* conflict. Failed-change conflicts can occur due to a number of reasons, such as mismatched constraint definitions or mismatched triggers [1].

5.1 Handling Conflicts in Merge Replication

We need to handle any conflicts that have occurred during the replication synchronization. A conflict occurs at a publisher and subscriber or it can occur at two different subscribers. Conflicts can be automatically resolved using the conflict resolver. However, the special tool like (i.e., conflict viewer) allows you to choose a different resolution for the conflict**. The following conflicts can occur in merge replications.

Update conflict: Update conflicts occur when the same data is changed at two locations. One change wins, and the other one loses. The options are to keep the existing data (the data that won), overwrite the existing data with the data that conflicted with it (the losing data), or merge the winning and losing data and update the existing data.

Insert conflict: Insert conflicts occur when a row is inserted at one location that violates some data consistency rule when merged with changes at other locations. The options areto keep the existing data (the data that won), replace the existing data with the data, or merge the winning and losing data and update the existing data.

Delete conflict: This conflict occurs when the same row is deleted at one location and changed at the other.

When conflicts are resolved during synchronization, the data from the losing row is written to a conflict table. After you accept the original resolution or choose a different resolution for the conflict, the logged conflict row is deleted from the conflict table. It is necessary to review conflicts periodically to help reduce the size of the conflict table [8].

***Note**: The Conflict Viewer is divided into two sections. The upper section of the dialog box shows the conflict list for the selected table. When an item, is clicked the details of the conflict are displayed in the lower section. For example, information describing why the conflict occurred (The same row was updated at both the publisher and the subscriber) is displayed. The conflict data is displayed in two columns (Winner and Loser). If the conflict is of type delete, there may be no data to show for the delete side of the conflict. In this case, the conflict viewer displays a message in one of the columns, indicating that the row was deleted at one location and updated at another.*

VI. Experimental Results and Discussion

In this section, the case study for merge replication is analyzed using a simulation model. The simulator has been used to study the merge replication, and for comparing it to the other replica models. The server is modeled for analysis. Client processes simulate the execution of transactions in the system. Each client process is associated with a single secondary site, and submits all its transactions to that site. Client processes are distributed uniformly over the secondary sites in the system. Other processes in the system are the (update) propagator and refresher. The Merge replica model has been designed to allow for the incorporation of additional datasets as they become available.The model is setup to use three replicas namely MergeReplicationPb, MergeReplicationSB1 and MergeReplicationSB2. Each replica contains one table named*Customers*. It includes 90 rows and 7 columns (with name CustomerName, ContactName, Address, City, PostalCode, Country and rowguide).This database will provide a valuable resource to those working in the field of database replication. By providing a consolidated dataset and powerful search facility, the case study model will also assist those working in related fields by making the existing datasets more accessible.

For each run, the resulting conflicts in the system are shown in Table 1. It has columns Subscription,#Rows in DB, #Rows to update, #Conflicts, #Conflict Winner, #Updated Rows and Time for Publisher to get synchronized.

Table.1. Testing Conflicts in Merge Replication

Subscription	*Rows in DB*	*#Rows to update*	*#Conflicts*	*#Conflict Winner*	*#Conflict Loser*	*#Updated Rows*	*Synchronization Time for Publisher*
MergeReplicationSB1	*0 Row*	*50 Rows*	*44 Rows*	*30 Rows*	*20 Rows*	*29 Rows*	*00:43.02*
MergeReplicationSB2	*0 Rows*	*70 Rows*	*44 Rows*	*20 Rows*	*0 Rows*	*70 Rows*	*00:50.32*

These conflicts happened as the updates occur at the same fields almost at the same time. The merge replication accepts last arriving update requestof two updates. In the test, subscriber1(MergeReplicationSB1) has requested an update of 50 rows to publisher. Then almost at the same time subscriber2(MergeReplicationSB2) has requested an update of 70 rows.The publisher accepts update request from MergeReplicationSB1. When it was executing that update a new update request on the same place arrives from MergeReplicationSB2. In this situation, the publisher accepts last update from MergeReplicationSB2 and reports 44 conflicts in the replication systems as shown in Fig. 4.

It can be seen that the database from the specified publication has produced 44 conflicts which means the case study of merge replication has parallel update process during this time. It is evident that the merge replication conflicts are well documented.

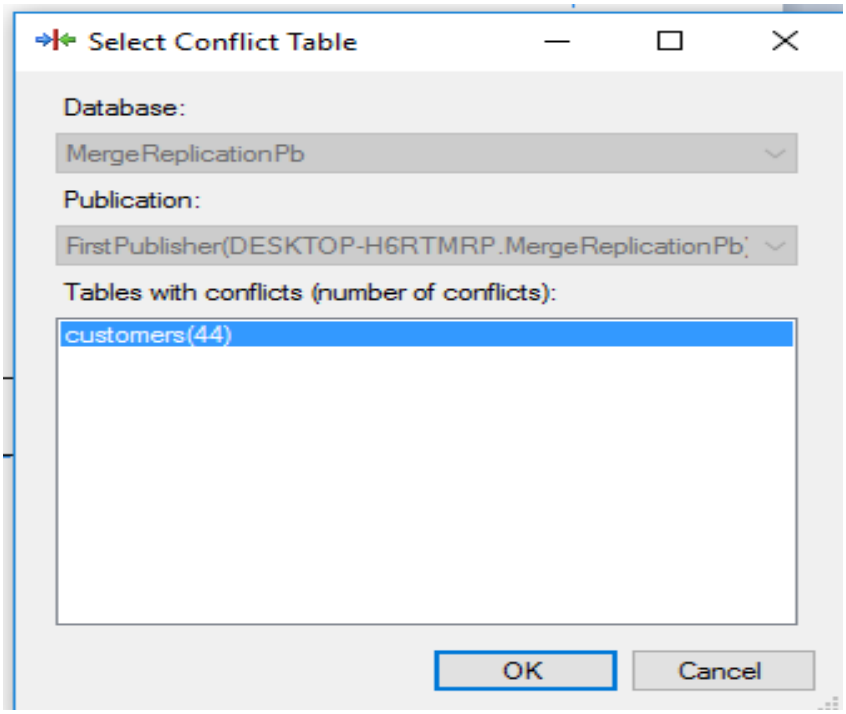

Fig.4. Conflict table

Conflicts happen when some operationsfail to satisfy their preconditions.The best approach is to prevent conflicts from happening altogether. Pessimistic algorithms prevent conflicts by blocking or aborting operation as necessary. Single-master systems avoid conflicts by accepting updates only at one site (but allow reads to happen anywhere). These approaches, however, come at the cost of lower availability. Systems with semantic knowledge of operations can often exploit that to reduce conflicts. The trade-off between syntactic and semantic conflict detection parallels that of scheduling: syntactic policies are simpler and generic but cause more conflicts, whereas semantic policies are more flexible, but application specific. Thus, this can be taken as the major step towards reducing conflicts.

As stated earlier, the synchronization process has to be performed to identify the conflicts. During synchronization, if only one of the replicas is found to be modified, the new value is copied to the other side. If both the modified bits are set, the system detects a conflict. Conflicts are resolved either by an application-specific resolver or manually by the user. After requesting the synchronizations between subscribers and publisher the status is shown in Fig. 5. The multi-master system will treat many of these status updates as conflicts andresolve them. On the other hand, pessimistic or single-master systems with the same aggregate update rate would experience an abort rate of only O[M], as most of the concurrent operations can be serialized without aborting, using local synchronization techniques. Still it requires remedies for the scaling problem.

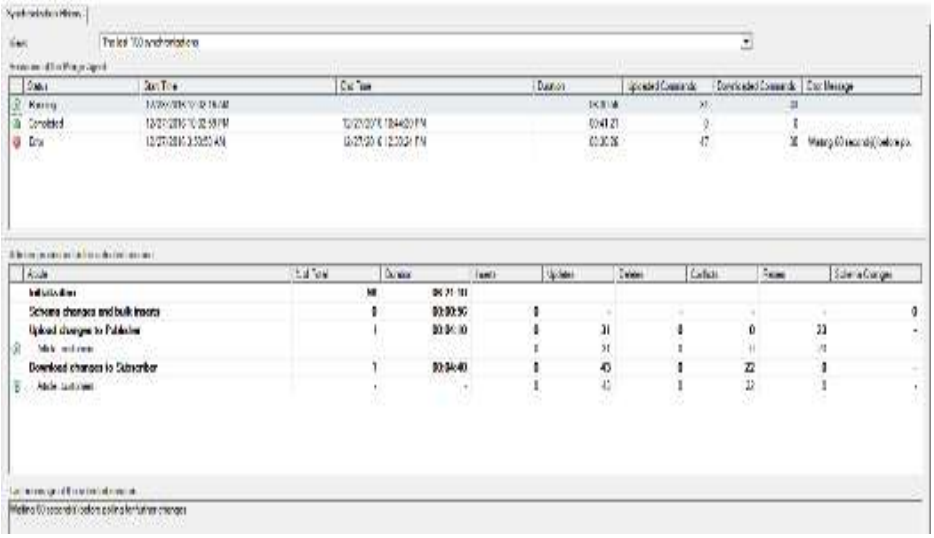

Fig.5. Synchronization

Snapshot agents are shown in Fig. 6. The Snapshot Agent prepares snapshot files containing schema and data of published tables and database objects, stores the files in the snapshot folder, and records synchronization jobs in the distribution database. From the status shown in Fig.6, the current status of the merge replication of the database can be determined. Thus, the case study of the merge replica model has been analyzed from which valuable inferences are made which are helpful in developing novel techniques as solutions for database replication problems. This aids in the establishment of reliable distributed databases for the clients.

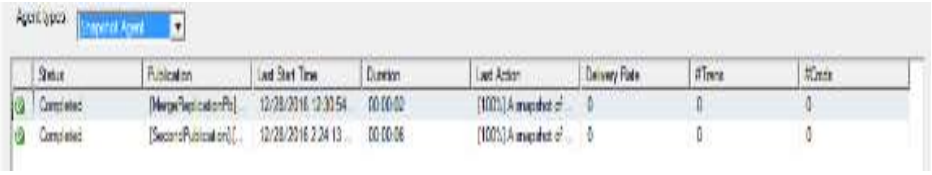

Fig.6. Status of snapshot agent

VII. Conclusion

The database replication is increasingly being used in various organizations in order to support the reliability and availability of distributed databases. In this paper, we presented a study about merge replication in which data transfer takes place dynamically. In our study, the replication system consists of three components, namely, a publisher and two subscribers. Initially, original data is transferred from one publisher to two subscribers. We apply a bunch of database operation (insert, delete and update) to the databases in the subscribers and ask replication system to synchronize. Then we calculate performance of the replication system, data transfer time, occurrence of conflicts and discuss options available to their resolutions.

References

[1] S. Meine, Fundamentals of SQL Server 2012 Replication, *TairwaysHandbook*, ISBN – 978-1-906434-98-4, 2013.
[2] M. C. Mazilu,Database Replication, *Database Systems Journal*, 1(2),2010, 33-38.
[3] Merge Replication, https://www.techopedia.com/definition/24730/merge-replication, Access Date: April 25, 2017.
[4] R.N. Sangwan, SQLServer2012Replication, https://www.codeproject.com/Articles, Access Date: April 25, 2017.
[5] Chigrik, A., and A. Chigrik. Setting up merge replication: A step-by-step guide. Online] Tersedia: http://www. databasejournal. com/features/mssql (2001).
[6] H. Lamehamedi, B. Szymanski, Z. Shentu, E. Deelman, Data Replication Strategies in Grid Environments, *5th International Conference on Algorithms and Architecture for Parallel Processing, IEEE Computer Society Press*, Los Alamitos, CA, pp. 378-383, 2002.
[7] Sitecorecms, SQL Server Replication Guide,Sitecore compelling web experiences, rev:2016.
[8] https://msdn.microsoft.com/en-us/library/ms189029.aspx, Access Date: April 25, 2017.
[9] Saadi H. Thalij, Veli HAKKOYMAZ, Design Issue in Database Replication, Yildiz Technical University, Computer Engineering Dept., TechnicalReport, April 2017.

Saadi Hamad THALIJ Dynamic Data Distribution for Merge Replication in Databases.” IOSR Journal of Computer Engineering (IOSR-JCE) , vol. 19, no. 6, 2017, pp. 41-46