PRAJÑĀ - Journal of Pure and Applied Sciences Vol. 26 : 38 – 46 (2018)
ISSN 0975 - 2595

DATA SYNCHRONIZATION IN APPLICATION SOFTWARE

L. N. CHAVALI, P. N. OJHA AND C. RAM LOHITH

Technology Practitioners and Consultants, Hyderabad

Email: lnchavali@yahoo.com

ABSTRACT

It is a well known fact that the data in database is bound to have changes with the passage of time as it is designed for applications in real world. As world changes day by day, it is challenging to handle the changes in database and to transport them to different target data stores. In application software under discussion in this paper, these changes are being transported to target data store by ETL (Extract Transform Load) from State Data Centre (SDC) to National Data Centre (NDC) and by Customized Synchronization (Sync) framework using stored procedures from offline locations to SDC. The primary focus of this paper is on Customized Sync which is implemented as a multi-threaded sync server that will synchronize data to and from different offline servers to SDC in a transparent manner to the users.

Keywords: synchronization, blackout, thread, offline

INTRODUCTION

Communication topology and propagation strategy are two vital aspects of data synchronization. 'Communication topology' defines the pattern of communications among systems /or nodes whereas 'Propagation strategy' defines how frequently system /or nodes synchronize with one another [2]. The topologies range from central master, ad-hoc peer-to-peer, hierarchical etc. The propagation strategies vary from eager to lazy strategies. Eager strategy may help updating the data immediately while the update propagation may be delayed in lazy strategy. The eager may minimize or reduce the risk of conflicts in terms of stale reads and write conflicts whereas resolution is required to address these conflicts in lazy strategy because of delay in communication and of large write-logs. However, lazy propagation offers an advantage that the systems can be operated in a disconnected state.

There are many methods through which the data synchronization can happen. Some of them may include [3]**: -**

1. **Trigger method** – Insert, delete and modify triggers are used to sync data by activating triggers.
2. **Log Analysis method** – It is nothing but analysing the information of the database log to capture changes in sequence of synchronization objects.
3. **Time Stamp based method** – A method where in every table of the application has a timestamp field to record the modification time of each table.
4. **Trigger and Log Table method** – This is closer to realizing real time database synchronization by employing web services with event driven mechanism.

The rules to resolve conflicts may be domain independent or may depend on application semantics i.e. data driven reconciliation in which systems can trigger reconciliation often enough to avoid conflicts by using application-specific knowledge. If the users decide to trigger manual reconciliation, it is likely to increase delays because of volume of data and therefore may result in increase in conflict rates. However, the chief advantage of manual reconciliation is control and allows user to synchronize the data as and when needed.

The tools like Google *Gears* is an open source browser extension that lets developers create web applications that can run offline. Nevertheless, the goal of *Gears* is not just to enable offline application, but to bridge the gap between web application and desktop application [7]. The popular utility like *rsync* which is widely deployed on UNIX systems is used primarily for synchronizing files and directories from one location to another mostly in backing up systems [8].

There are many proven commercial frameworks that are available in the market as of today. Some of them are described below: -

1. Microsoft Sync Framework

It is a data synchronization platform from Microsoft that can be used to synchronize data across multiple data stores. Sync framework can be used for offline access to data, by working against a cached set of data and

submitting the changes to a master database in a batch, as well as to synchronize changes to a data source across all consumers (publish/subscribe sync) and peer-to-peer synchronization [1] of multiple data sources. Sync framework features built-in capabilities for conflict detection such as changes to the data that has already been updated, and can flag them for manual inspection or use defined policies to try to resolve the conflict. There is no unique way or unanimously agreed method for data synchronization. This task differs from case to case, and even data synchronizations that should be simple at first glance can be complicated, due to the complexity of data structures. The implementations of data synchronization are rarely optimal [10].

2. Open Sync Framework

OpenSync is a synchronization framework that is platform and distribution independent. It consists of a powerful sync-engine and several plug-ins that can be used to connect to devices. OpenSync is very flexible and capable of synchronizing any type of data, including contacts, calendar, tasks, notes and files (https://www.openhub.net/p/opensync).
OpenSync is a successor project of MultiSync. OpenSync's main and most practical goal is to create a solution to synchronize PIM (Personal Information Management - address book contacts, calendar events and tasks, personal notes, etc.) data between mobile devices like mobile phones, PDAs (Personal Digital Assistant), desktop computer PIM tools and services [4]. SymmetricDS (https://www.symmetricds.org/doc) is open source software for database and file synchronization, with support for multi-master replication, filtered synchronization, and transformation. It uses web and database technologies to replicate change data as a scheduled or near real-time operation, and it includes an initial load feature for full data loads. The software was designed to scale for a large number of nodes, work across low-bandwidth connections, and withstand periods of network outage.

3. Oracle GoldenGate Replication Framework

.

GoldenGate can synchronize two heterogeneous databases, by reading the real-time history of all transactional changes of the source database. The delta is then sent over to the other database, again in real-time. In technical language, this is called Change Data Capture (CDC). Simplicity is the feature of killer applications; GoldenGate guarantees transactional integrity when copying between source and target databases. With the simple deployment of GoldenGate between databases, one can build a High Available i.e. redundant architecture with each database holding accurate copies of each other's data. Most importantly, GoldenGate permits active-active database replication. Oracle GoldenGate guarantees that even in an unstable environment where networks and host servers occasionally drop out, transactions will never be missed or skipped [5].

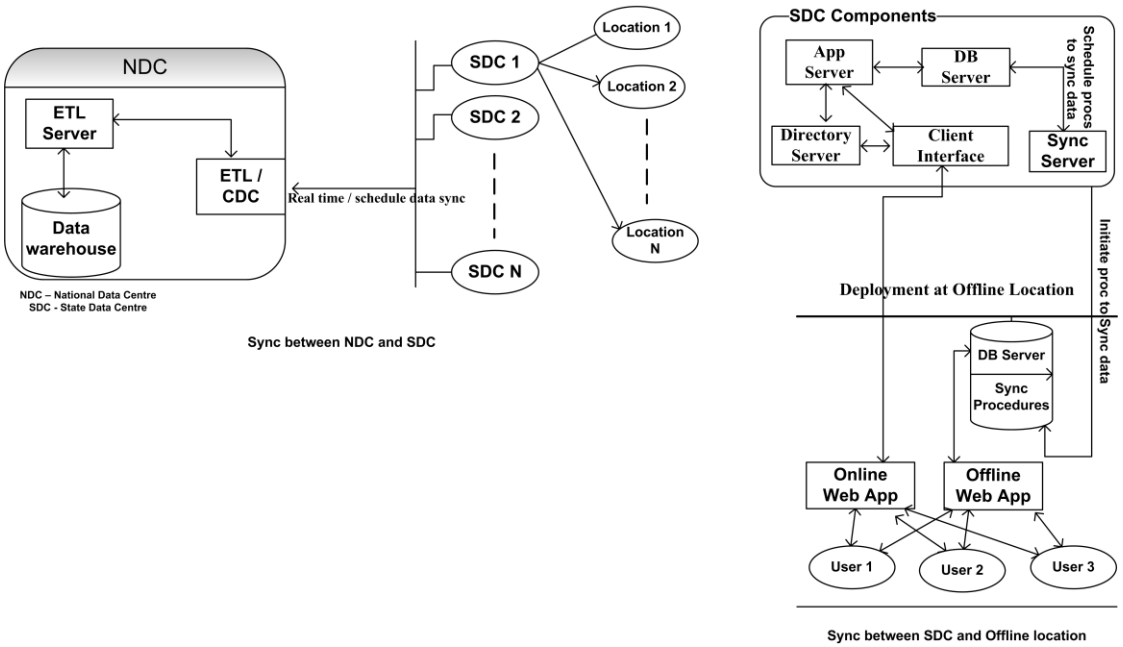
There are many off-the-shelf products like Aspera Sync from IBM (https://www.asperasoft.com) and Attunity (https://www.attunity.com/data-replication-tool/) for replication or synchronization of data.

4. Customized Sync framework

In applications, any data that is entered at source (i.e. offline) is stored at source's server and this server in turn will communicate to target server hosted at data center for data synchronization over internet/SWAN/intranet links. However, the data exchange over internet between source and target will take place through Secure Socket Layer (SSL). The data will be synchronized between server at State Data Centre (SDC) and server at source without interrupting user operations.

The synchronization in applications under discussion is designed for MySQL / MSSQL database server at SDC and MySQL as offline server at remote locations. The sync server is built using Java Development Kit (JDK version 6.0 or above), and the databases at both ends should be running for Sync to operate. The database at SDC is configured for row level lock only. Fig 1 given below briefly depicts the sync framework and the functional flow of synchronization

Fig 1 – Synchronization Framework



The table 1 describes the key feature comparison of custom sync with that of commercially available off-the-shelf products.

Table 1: Feature Comparison of different sync models

Key feature	Microsoft Sync	Oracle GoldenGate	Open sync	Custom Sync
Synchronize by using an n-tier or service-oriented architecture	Yes	Yes	Yes	No
Supports heterogeneous databases	Yes	Yes	Yes	Yes
Incremental change tracking	Yes	Yes	Yes	Yes
Conflict detection and resolution	Yes	Yes	Yes	Yes
Easily create data views on the client (UI)	Yes	Yes	No	No
Automatically initialize schema and data	Yes	Yes	No	Yes
Supports large data sets	Yes	Yes	No	Yes
Query processor is locally available	Yes	Yes	Yes	No
Use on devices	Yes	Yes	Yes	No

METHODOLOGY

1) Synchronization Model in Applications

In custom sync framework, the synchronization interval must be configured to sync the data between online resources and offline resources. If resource at the both ends is the database, then the data transmission is completed before purging the data from the offline database; and before reaching the database size limit.

Data Synchronization

Master data synchronization is done on the basis of client / server model. The data always moves from the server to the client, and therefore the synchronization is unidirectional. The data change i.e. incremental changes in masters is synchronized with offline database by stored procedures.

The custom sync framework supports two fold approaches viz.

a) Synchronize the metadata of online database i.e. database at SDC with that of offline database.
b) Bi-directional synchronization of the record or transactional data.

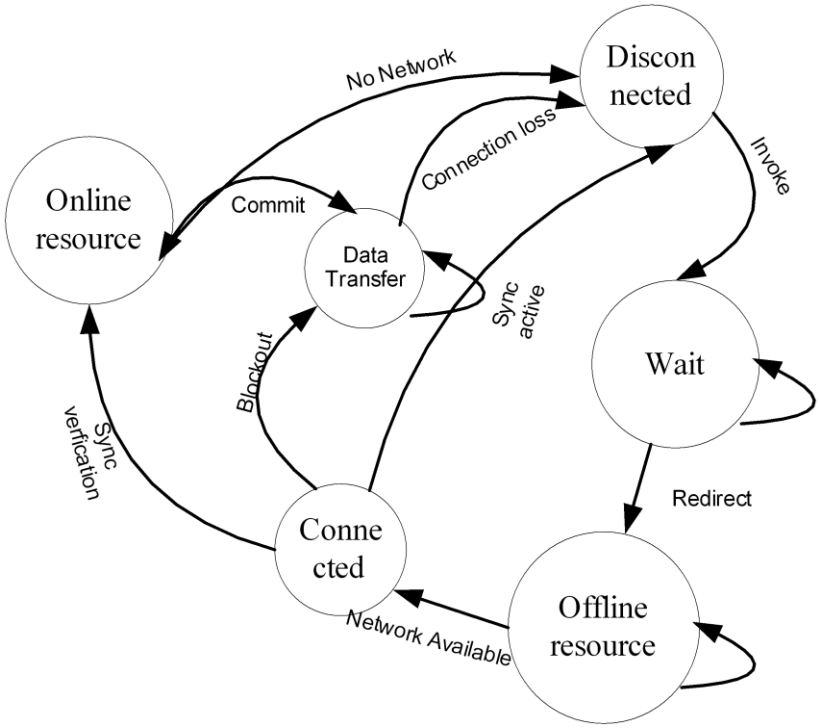
Method 1 – offline and online transmission of data over http links.
Method 2 – offline and online transmission of data over user socket.

The approach in this framework has a few limitations as briefly explained below.

1. When network connectivity is available, offline application is barred from operation.
2. When connection to the SDC server is re-established, user will be notified about the blackout period (i.e. no operation from online or offline) during which the background utility will synchronize the metadata, and subsequently, the record or transactional data will be synchronized. During the metadata synchronization, user is not allowed for any data submission from the application.

The given below figure illustrates various transition states of sync server.

Fig 2 – State diagram of Sync



From Fig 2, the transition from offline to online happen only by having valid network connection, declaring blackout and performing data transfer. The transition from online to offline happened as and when there is a loss of network connectivity. The user operates the offline application and all transactional data will be put in offline database. It is to be noted that the metadata is synchronized only after declaring the '*blackout'* period during which the offline users cannot operate the application from offline or online.

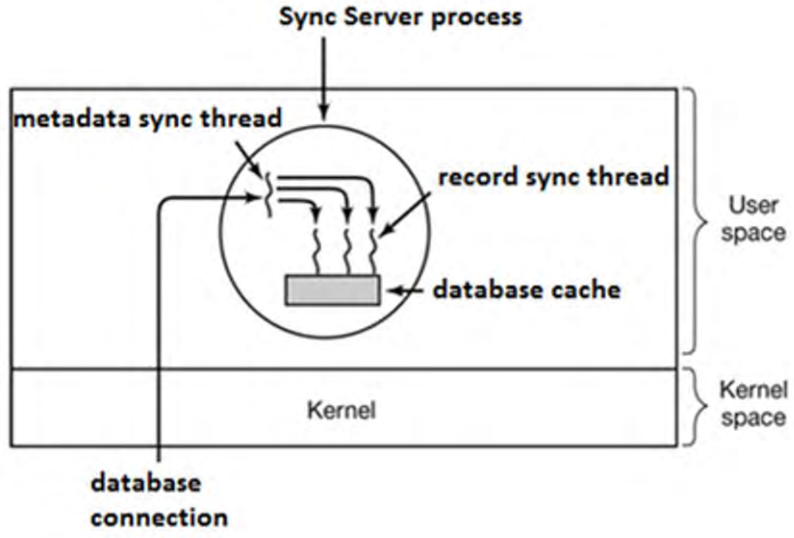
2) Sync Server Implementation

The sync application is built as multi-threaded server which is run from SDC. It executes various procedures for synchronizing the data between SDC and offline server. Thread is a path of execution in the process. Threads allow multiple executions to take place in the same process environment, to a large degree independent of one another. The main reason for having threads in sync application is that multiple offline stations/servers are required to be synchronized for data. Therefore, in order to perform the data exchange with many offline servers simultaneously from SDC, the synchronization tasks are decomposed into multiple threads which run in parallel / quasi-

parallel to perform the tasks. When a multithreaded process is run on a single-CPU system, thread scheduling is one dimensional and CPU should decide in which thread it should run next. It takes turns to run multiple threads. The CPU switches rapidly back and forth among the threads providing the illusion that the threads are running in parallel. When threads are scheduled on multiple processors, scheduling is two dimensional as the scheduler has to decide the thread to run on which processor there by making it more complicated. In multiprocessor scenario, the threads run simultaneously on virtual processors associate with each CPU. The virtual processors can be taken back by kernel in order to assign them to other needy processes or threads. All threads in the process share resources of the process like open files, signals etc. It is not possible to have protection between threads in the process and it may not be necessary as threads are expected to collaborate and to work together to perform the tasks.

The sync server in software applications is organized as shown in Figure 3.

Fig 3 - Multi threaded sync server



Sync server will have two threads per offline station apart from three other threads meant for logging, console and the main. The '*metadata sync*' thread checks the metadata at SDC and at offline for a module and synchronize offline metadata with that of SDC. '*record sync*' thread exchange all such records whose meta flags are verified by 'metadata sync' thread in both directions.
The pseudo code of the sync server is given below:

```
main ()
{

if (connection success from both DB)
{
/*Sync DB at SDC to offline DB */

                If
Establish Server Socket at port 19218;
If remote station code in host list == "true" && thread is not running)
{
//start the thread
while (1 == 1)
{
Start metadata sync thread; // may be referred in program as SDC to offline
Start record sync thread; // referred as offline to SDC
Establish the connection from Server DB at SDC;
Establish the connection from offline DB;

(RECORD_SYNC_ON = NULL)
{
Fetch Records of Server DB at SDC that are yet to be synchronized;
Convert the Records into XML format;
```

Insert XML records into offline DB;
Update Sync Flags at offline and SDC;
Commit;
}
/* Sync offline DB to Server DB at SDC */
If (RECORD_SYNC_ON = NULL)
{
Fetch Records of offline DB that are yet to be synchronized;
Convert the Records into XML format;
Insert XML records into Server DB at SDC;
Update Sync Flags at offline and SDC;
Commit;
}
} /* end of if - connection loop */
} /* end of while loop */
} /* end of if – thread */
Else no need to sync;
} /* end of program*/

The XML files which are created at run time for data transfer are deleted automatically at the end of the data transfer.

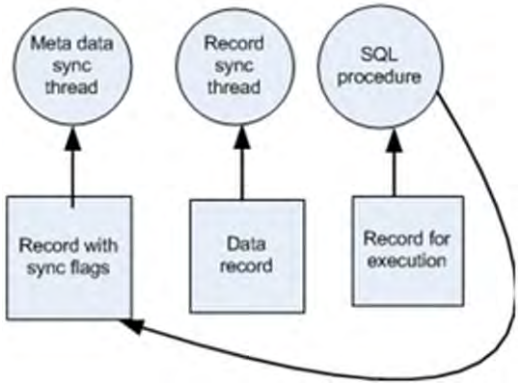
OBSERVATIONS

Deadlocks

Deadlock occurs when a group of threads have been granted exclusive access to some resources, and each one wants yet another resource that is being used by other thread in the group. All of them are blocked and none will ever run again.
It has been observed that whenever a new user is assigned in an offline application through Administrator, the new user is not visible in offline application, but it has shown up the previous user in offline. However, the new user works fine in online mode.

The above user problem is explained in Fig 4 to understand the deadlock scenario. This has been presented here for illustration purpose only.
In Fig 4, the sync flags of SDC and offline servers are allocated to metadata sync thread, data record to be updated is allocated to record sync thread and the record in cache is allocated to SQL procedure. The procedure is requesting for access to update the sync flags post data exchange whereas this is locked by metadata sync thread and thereby creating a deadlock.

Fig 4 – Deadlock situation



If procedure generates an exception, then the user threads will reschedule the data exchange after the schedule interval. However, if the procedure throws an exception again, then there is every possibility of a deadlock situation created.

There may be many reasons for a deadlock; nonetheless some well-known reasons for deadlock are listed below: -

1. Exhaustion of the thread table.
2. Limitation on the number of open files and finite swap space.
3. Contention for the shared resources.

Most of the errors observed in error table 2 may be indirectly related to contention of resources
Maintaining the data dependency across threads is vital. Disambiguating the addresses accessed by different threads, invalidating stale state in caches, making the state of a committing thread visible to all other threads, discarding incorrect state when a thread is squashed, and managing the speculative state of multiple threads in a single processor are important in speculative multithreading [6].

RESULT AND DISCUSSION

The configuration of the database server in the lab where sync is deployed and tested is Quad core and 20 GB RAM. Table 2 briefly lists out the errors that have been logged by Sync Server.

Table 2: Errors in sync log

S.No	Error type	Description
1	Procedure does not exist	Data sync from SDC to offline server A unable to locate the PROCEDURE "offline_db_sync_proc.PR_DS_UPDATE_PROP_EXT_SYNC_FLAG".
2	Duplicate Entry	Data sync from SDC to offline server B Duplicate entry '81121500213001501-99' for key 'PRIMARY'
3	Deadlock on locked resource	Data sync from DC to offline server A Transaction was deadlocked.
4	Station out of sink	Station C stop to sync automatically.
5	No operation allowed	Data sync from SDC to offline server D for Master update, user update and data entry. Connection was implicitly closed by the driver.

1. Performance Statistics

It is evident from Fig 4 that the memory consumption of db and sync servers is high, thereby forcing a restart of sync server. There is a marginal change in memory consumption and the number of threads as seen from Fig 5 and 6. Therefore, it can be concluded that only a few transactional activities are taking place. Out of all nine offline servers started, four offline servers were shut down for some reason during test and two offline servers out of the four were re-started. The number of threads at start up is 21. However, the base thread count remains the same despite reduction in the number of offline servers. On the contrary, it has been observed that the thread count is varying over the base count because of offline server's unstable connectivity.

Fig 5 – SQL memory versus Sync memory.

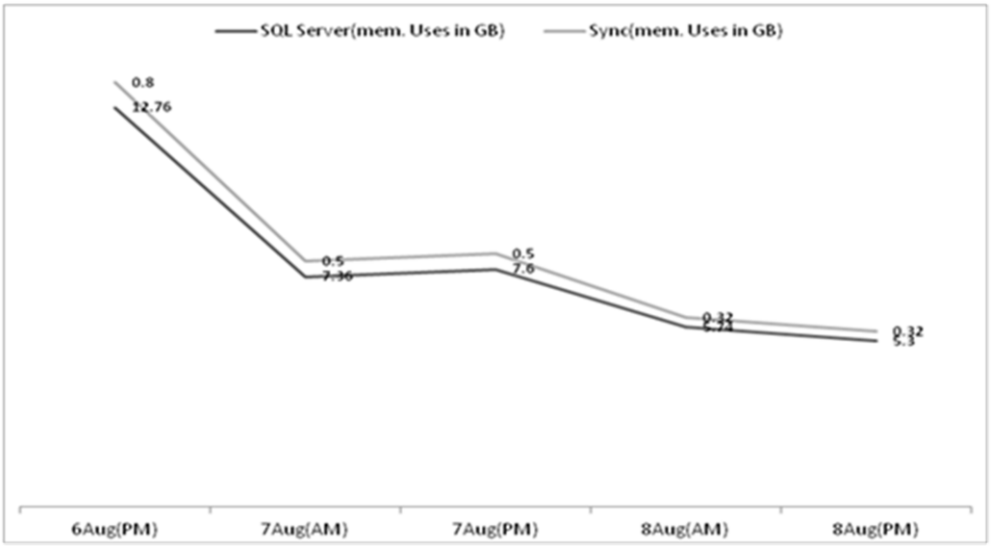
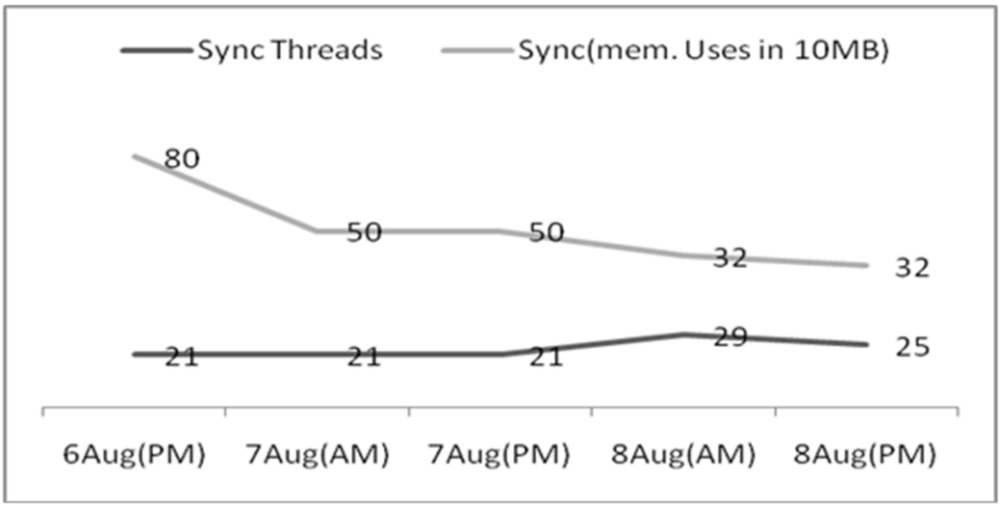


Fig 6 – Sync thread versus Sync memory



2. Query Response

The sync framework in application software replicates the data on transactional basis. There are 25 classes of transactions in the application software under test. Whenever synchronization happens for a specific transaction, it seems that all the related table data will be exchanged. Any new transactions outside the window of the existing set of transactions may not be synchronized.

The response times of five procedures that have been captured at two different instances during test are shown in Fig 7. The graph was generated with SQL profiler of Microsoft. It can be observed that the response time varies from 1.3 sec to 20 sec with few records in the database in the test lab. The expected user response time as per requirements at peak load is 3 sec. The query response times are based on the current execution plan. The same execution plan will deteriorate the response times for higher workload or higher volume of data.

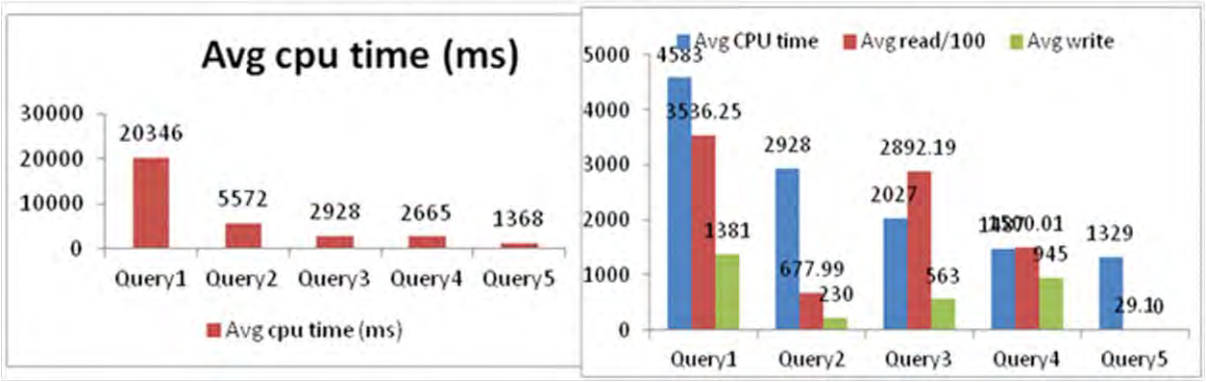
3. Root Cause Analysis

It has been observed that in some of the transactions, the long running queries are holding the database locks thereby causing other resources waiting in the queue. This has resulted in longer transaction commits to the database.

LIKE operator is being used widely in procedures to generate the primary key for some of the tables, which has caused full table scan. Due to this, the performance has been impacted. As the procedures are re-used by other modules, it has resulted in long wait, lock wait and time out of the transactions.

To alleviate this problem, the number generation logic of the primary key has been changed and the execution times of stored procedure have improved by many folds. The concurrency is controlled by modifying the throttling parameter.

Fig 7 – Response times of procedures



CONCLUSIONS

It is planned by design that the offline application is a contingency arrangement for no online access for any reason. In the custom model discussed in this paper, the tight coupling between offline and online is a cause of concern. Loss of data, different response times to the same transaction from different locations, performance impact on database due encryption or multi-lingual adaption are some of the bottlenecks which are yet to be addressed in the solution, possibly would be addressed in future releases of synchronization software. The customized framework is tailored to the customer requirements and will scale up easily as more and more locations are added. This framework is currently being used in one of the mission mode projects. The computing paradigm is slowly shifting from centralized to decentralized as evident with the advent of blockchain solutions. The emergence of distributed technologies has given birth to new ways of synchronizing data sets and files which is agnostic to the underlying transport. A new protocol viz. DAT is designed for syncing folders of data, even if they are large or changing constantly. DAT is a dataset synchronization protocol that supports public or private decentralized network and use public key cryptography for encryption [9]. The sync methods in future may become more decentralized and may replace the existing methods due to factors such as cost, vendor lock-in, speed, privacy and centralizatio.

REFERENCES

[1] Joe Kunk (2012): Database Synchronization with Microsoft Sync Framework, *Visual Studio magazine*, 34-38

[2] Mike Dahlin, Aslan Brooke, Muralidhar Narasimhan, Bruce Porter (2000): Database synchronization for distributed simulation, *European Simulation Interoperability Workshop*, pp 1-7

[3] A.F. Cardenas (1987): Heterogeneous distributed database management. The HD-DBMS, Proceeding of IEEE Volume 75(5):588 - 600

[4] Hui Ling Du, Benjamin C. Pierce (2009): Universal Data Synchronizer for Personal Calendars, *Conference Proceedings*, *Website*: https://pdfs.semanticscholar.org/

[5] Oracle Corporation (2018): Oracle Fusion Middleware Using Oracle GoldenGate for Heterogeneous Databases, 12c (12.3.0.1), Oracle Press USA, pp 15.1-22.4

[6] Luis Ceze, James Tuck, Calin Cas, Josep Torrellas. (2006): Disambiguation of Speculative Threads in Multiprocessors, IEEE 1063-6897/06

[7] Tushar Mohate, Vijay Rasal, Anil Panchal, Sarita Ambadekar (2015): Internet Data Synchronization Tool, IJCST Volume 3 Issue 2

[8] Salekul Islam and Mohammad Amanul Islam (2014): A Web-based Data Backup, Synchronization and System Administration, DOI: 10.5815/ijcnis.2014.09.01, *Website*: http://www.mecs-press.org/

[9] Maxwell Ogden, Karissa McKelvey, Mathias Buus Madsen (2017): Home page for Code for Science & Society. Retrieved from *Distributed Dataset Synchronization and Versioning, Website*: https://codeforscience.org

[10] Andrej Gajdos (2016): Home page for toptalTM, Retrieved from *Guide to Data Synchronization in Microsoft SQL*, toptalTM, *Website*: https://www.toptal.com/sql/