International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 3, Issue 3, March 2016

Tracking of Change in Database at API Level

Miss. Shivani Dhotre [2] Prof. V. S. Mahalle
Department of Computer Science and Engineering
Shri Sant Gajanan Maharaj COE,
Shegaon, Maharashtra, India
[1] Shivani3464@gmail.com, [2] vsmahalle@gmail.com

Abstract: -- **If the rising demand of real time data by modern applications is considered, the traditional techniques of keeping applications in sync with data in database server appears insufficient. Normally in client-server architecture, client needs to check the server frequently for data change. This approach is inefficient and non-reliable as the data may get change at any time irrespective of the time at which client check the server for changed data. As this task needs to be performed frequently it consumes system resources causing performance issues.**

Recently, some DBMS vendors are introducing APIs to cope with this issue. Each of the vendors follows different approach. Similarly, in this paper we are looking into Event driven approach of tracking data change. This is another such efficient approach of data change tracking. To implement this approach existing DBMS needs to be added with modules written in native language and client side application also needs to be added with API to register query for data change tracking. In addition we tried to focus solution for performance issues with the previously introduced approaches by different vendors. This approach will help applications which are using DBMS which lack data change tracking functionality to fulfill real time data need.

I. INTRODUCTION

Normally the client side applications query the database for fetching corresponding Result Set. But in cases where client application needs real time data as soon as it is updated in database, then such applications which are connected to DBMS need to poll the database frequently after equal interval of time to check the change in data. This approach of frequent polling is not fully reliable as the data in database can get updated at any point of time irrespective of the frequency of polling by client, this may lead to non-real time updates of data change. This will not full fill the need of real time tracking.

This traditional approach is also inefficient as client application needs to implement infinite loops for continuous polling. Also application has to make a network call to the database once in each execution of the loop. Such infinite loops generally results into high CPU and network usage, as infinite loops tries to occupy as much CPU cycles as possible, which will degrade the performance of other tasks running on the system and as each time application will do network call it will incur latency. The traditional approach of frequent polling from application to the database, to fulfill the data need of real time data application, contains a lot of time overhead and excessive CPU usage for tracking change of data. An increasing volume of data often leads to performance degradation in most of the enterprise applications. The technology infrastructure not only has to meet the current performance requirements but also continue to scale as the business demand grows.

In order to do this task efficiently, there should be some way such that server will notify to the client about change in data of desired result set instead of client polling the result set frequently. Recently, few vendors are providing different approach to cope with this issue. They provide APIs to create applications to get real time updates of data change. Event driven approach [1] of tracking change in database is one from such ways to get notified about change at real time efficiently. In this method client application that is connected to the database through adapter can register a data query for tracking change in its result set. Adapter connecting application and database adds this data query in data dictionary of database. It makes module in database to watch for change in Result set of same data query and application get notified from DBMS server as soon as watched data has been changed. It is a feature that enables client applications to register queries with the database and receive notifications in response to DML changes on the objects associated with the queries. The notifications are published by the database when the DML transaction commits.

II. INDUSTRIAL PERSPECTIVE

Before considering the Event Driven data change tracking, we should note that there has been great interest in this topic of data change tracking within the database the last few years. Most leading vendors like Oracle [2] and MS SQL Server [3]provides some data tracking functionalities which are limited to API in server side languages while several popular and second mostly used DBMS like MySQL

[4]still lacks such functionality. Even though Oracle and MS SQL server 2008started providing data change tracking, they need Performance tuning with respect to the large data and high frequency of data change .There is also a need of generic API for data change tracking in several other server side programming languages like Node.js.

The importance of data change tracking in the commercial segment appears to be due to a need for enterprises to deal with the increasing demand of real time data applications, and the desire to decouple such data tracking from general DBMS functionalities. Data change tracking currently supports non complex queries and need performance tuning while dealing with large data and high data change frequency. Nevertheless, performance tuning with complex query is a very broad area, so we focus this paper specifically on research issues associated with event driven data change tracking for DBMS like MySQL which lack data tracking functionality and possible future scope in tuning the performance of existing techniques provided by other vendors.

III. CONCEPT OF EVENT DRIVEN TRACKING OF CHANGE IN DATA

The concept of Event driven tracking of change in structured data in the database is to get notified about change at real time with the help of additional module that can be added into the database [5] . The notification will be received by event loop started by calling method in API in module added in client application .It will generate a change event for watched result. The event generation will occur in application as soon as the activity of update, deletion or insertion occurs on result set of watched query which will not require client application to poll the database.

Generally in the client server architecture, Client has to send the request frequently to get updated data. Similarly client application connected to database server needs to query the database for checking that whether data is updated or not. This can be modified by adding these additional modules into the client application and database server. In order to accomplish this concept the newly integrated module in client side application will make initial connection call to the DBMS server, this connection need to persist throughout the time until the application needs to track data change.

After initiation of such first network request call application can register the data query to be watched by the database for any modification. This will also need application to have a listener that will continuously listen on the persisted network connection for any response by the database. In case the listener in the application listen some notification from database side. Application will decode the response and will find that data has been changed and will generate internal event that will further run callback code.

IV. EVENT DRIVEN ARCHITECTURE (EDA) OF DATA CHANGE NOTIFICATION :

The whole system mainly consists of two major components, which are Database server that is DBMS and the middle layer application or client side application. The framework consists of the modules and API to be added into these two main components. Client side or middle layer applications are generally connected to the database with the help of adapter. This adapter provides API for connecting application to the database and transforming calls from application layer to the DBMS. But, in order to accomplish event driven tracking of change in database, we need to add it up with functionality which will establish a persistent connection to db. Along with this module will provide ability to register query for watching change in result set by any client connected to it.

The API or component added to the client application has methods to register the query, whose result set is to be watched for change. Also it has event loop to watch over persistent TCP connection for notification. The event loop will emit the event inside the client side application as soon as it receives notification of data change over persistent TCP connection. Database System or DBMS will be added up with module written in native language [6] and integrated with DBMS library with which client application will establish a persistent TCP connection. This added component in the DBMS will watch for data changes that might have occurred on the watched result set. It will dispatch notification over TCP connection once data get changed. This module will act as another sub-server which will handle request raised by added module inside client application. These modules also will help to make the connection persistent until client application itself closes it.

As shown in Fig. 1 the persistent TCP connection is established between the newly added modules in DBMS and client side application. This TCP connection will be used for two way communication.

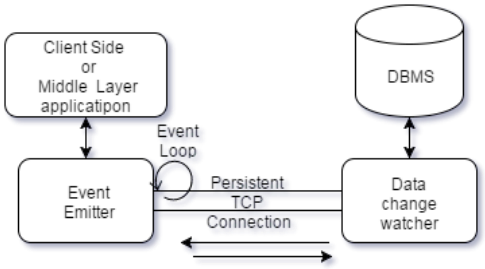

Fig. 1 Event Driven Architecture (EDA) of data change notification

Workflow:

Flow of Event Driven Tracking of Change in Database introduces step by step working and communication among internal modules of end to end framework. The framework consists of the module to be added into client application which needs real time data and the module to be added into the database server as described in the architecture. Considering that the initial setup is already done with client application and database application as mentioned in the architecture and show in Fig. 1, now data query can be registered using API provided by module added into the client side application.

If the application has code which registers the query using API, then while executing that code API will establish a persistent TCP connection from system where application is residing to the database server's system on the specified port. After the authentication is done client side API will send data query to the database server. Whenever the database server's additionally added module will receive the request for registering the query for data change tracking. Then the database module will initially validate the requested query against syntax and compilation error. After validation database server module will store that query in database's data dictionary [6] and will start a thread for watching the change in data.

The thread watching for change in the result set for registered data query will execute the query, every time commit operation performed by any transaction in the database system and then will compare the new result set with the old one. After verifying the old and latest result set, if the database server module finds data change in rows of result set, then the notification will be sent to the connected client server over a persisted TCP connection. Client application will have separate thread running event loop for watching over the persisted TCP connection to receive notifications send by the server. This thread will watch for data change notifications sent by the server.

As soon as the client will receive the notification, the event loop code will generate the event to notify client application about data has been changed. This notification can contain the row ids of the changed rows in the result set. Client application can have event listeners with callback functions attached to them, in order to perform desired task after data has been changed.

Performance Tuning For Earlier Approaches

As mentioned before that some of the vendors like Oracle and MS SQL Server are already into fully featured data change tracking and each of them has its own approach of change tracking. But, still the there are some performance issue that need to be tuned, mostly in the scenarios where number of daily database transactions are in millions.

MS SQL Server Change Tracking Cleanup Limitation

Microsoft SQL Server will not physically remove data from a page when you delete a row or update a row causing it to move. The old data is not visible using SELECT operations, but it is visible using operations such as DBCC PAGE[7]. If the data update or delete transactions happen more frequently then there are chances that sizes of such files will increase and therefore it may cause performance issue. In order to cope with it MS SQL Server provides Clean up procedures [7] to remove such deleted record's file data.

That is why it is necessary to run cleanup procedure after certain interval. It cannot be run more frequently as the execution of cleanup thread causes other database server transactions corresponding to same data source may get affected in terms of performance.

If we are having a database, with Change Tracking enabled, and the total number of transactions exceeds 14.4 million as in Eq. (1) for all of the tables that are tracked in that database then it is going to be in for some hurt .The problem is that the process to clean up the Change Tracking data in the (internal) *syscommittab* [7] table only runs once a minute, and only cleans up a maximum of 10,000 rows every time it runs.

10000 (rows) x 60 (minutes an hour) x 24 (hours a day) = 14,400,000

V. CONCLUSION

This work discusses an approach of data change tracking using Event Driven architecture and the performance tuning for the data change tracking feature of existing vendor's DBMS. The initially discussed framework relies on an Event-Driven Architecture (EDA), in which the DBMS server watching for data change emits notification over persistent TCP connection then the module on client side receives those notifications to determine whether registered stage constraints are met or not and generates the event to invoke callback code after data change.

Ongoing, we will monitor developments in the event-driven architecture approach of change tracking,

including progression standards, innovative uses of events and event processing and more performance improvements in existing change tracking feature enriched DBMS.

REFERENCES

[1] Brenda M. Michelson, "Event-Driven Architecture Overview", Sr. VP and Sr. Consultant, Patricia Seybold Group, pp. 2-3, Feb 2006

[2] DB-Engines (2016, Feb), DB Ranking - popularity ranking of database management systems.

[3] Oracle - Database Change Notification [online] https://docs.oracle.com/cd/E11882_01/java.112/e16548/dbchgnf.htm#JJDBC28815

[4] MS SQL Server - Track Data Changes (2016, Jan) https://msdn.microsoft.com/en-us/library/bb933994.aspx

[5] Cesar Galindo-Legaria, Torsten Grabs, Christian Kleinerman, Florian Waas, "Database Change Notifications: Primitives for Efficient Database Query Result Caching", Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005 pp. 1276-1277

[6] Charles A. Bell, "Expert MySQL", USA: Apress, 2007, pp.357-389 Kalen Delaney, Microsoft® SQL Server® 2008 Internals, Microsoft Press, 2009, pp. 77-85,146, 256–260

[7] K. S. Candan, D. Agrawal, W.-S. Li, O. Po, and W.-P.Hsiung. View Invalidation for Dynamic Content Caching in Multitiered Architectures. In Proc. of the Int'l. Conf. on Very Large Data Bases, pages 562–573, 2002.

[8] P.-A° Larson, J. Goldstein, and J. Zhou. MTCache: Transparent Mid-Tier Database Caching in SQL Server. In Proc. of the IEEE Int'l. Conf. on Data Engineering, pages 177–189, 2004.

[9] Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H.Woo, B. G. Lindsay, and J. F. Naughton. Middle-tier database caching for e-business. In Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data, pages *600–611, 2002.*