THE IMPLEMENTATION OF REAL-TIME DATABASE QUERIES IN PUSH-BASED QUERYING TECHNIQUE

Waseem Ahmad, Taimoor Hassan, Muhammad Atif Hasnain

Department of Computer Science, Lahore Garrison University, Lahore, Pakistan

KEYWORDS

Implementation, Real-time Database Queries, Push-based Querying Technique

ABSTRACT

Traditionally database can manage their data in application-independent structure where data are updated at fixed internal on user interaction do not fulfill all needs for software development. There is a need for getting information in real-time fashion which helps in making decisions in an ever-changing reality. When data access is accessed from different locations in a DBMS atmosphere, data retrieval delay becomes one of most critical issues in meeting transaction deadline i.e. stock exchange market. We have identified underlying techniques methods that are major hurdles in efficient implementation of real-time database queries and explore problems exists in traditional databases. The "aim of paper is to explore" the old /new technologies for getting the better and efficient results in push-based querying fashion.

INTRODUCTION

Traditionally databases are designed to support only pull-based techniques to access the data in which the client only gets data when they request to server. This model is only suitable for that type of application which only requires static data i.e. work on common data set and isolated from each other. Lot of new applications mainly in web applications like stock exchange data, messengers require continuously up-to-date data activity with respect to time. Hence, the client would be able to get result updates as soon as changes occurred in the results. Unfortunately, only some OLTP (online transaction processing) databases support real-time querying updates to clients afar triggers-based approaches and applications developers face lack of functionalities and uses frameworks to achieve the targeted goals. They designed the applications in such a way that application server's polls for data updates after some interval of time except maintaining the query outcomes in real-time behaviors. It can help in delivering up-to-date notifications to the users, but user-defined queries are not handled easily.

LITERATURE REVIEW

A "real-time database system" (RTDBS) guaranteed all features of conventional DBS like independence of data and "concurrency control", also, imposing real-time constraints involved in applications (Buchmann, 2006). Like, the operational database system, an RTDBS works like a repository of data, offers well-organized storage, and providing the information retrieval and manipulation. Real-time database manages "time-constrained data and time-constrained" transaction. Real-time "databases are generally used in real-time computing applications that require timely access to data. Definition of timeliness is not measured, for some applications, it is milliseconds", and minutes also (Stankovic, Son, Hansson, 1999). It provides all sorts of traditional system database like concurrency control and independence data, at same time it forces which applications needs. These databases added a requirement to meet the deadlines of the system. They are "commonly used in real-time computing applications that require" time-constrained data.

The traditional "database systems differ from an RTDBS in many" features. Three main aspects on which these two databases types differ from each other. Real-time databases are associated with timing constraints with different operations that are carried out, real-

time databases deal with temporal data which lost its validity and become stale after some period time and the last one is, the performance matrices in terms of transactions missing deadline per unit time. The aim of the database real-time is to maintain the data consistency along with the transaction processing to achieve the timing constraint of the transactions. A real-time data value becomes stale, until the condition "where it does not reflect anymore the state" of data with the passage of the time. Hence, "to maintain the consistency between" actual and the reflected state can lead to temporal consistency. The temporally "consistency can be measured in terms of absolute and relative consistency" (Ramamritham, 1993).

Absolute consistency deals with two states i.e. data is only valid between fixed intervals of time. It represents targeted and actual environment consistency. Relative consistency involves resulting data derived from others. The Real-time database needs to manage timing constraint of data implicitly or explicitly in predictable fashion i.e. to deal with transaction deadlines or periodicity limits bounded with tasks. Transactions "in a real-time database environment are subdivided into three" classes: update-only transactions, read-only transactions, and write-only transactions. Update-only "transactions are used to update the values of real-time data" including read and write operations to satisfy the concurrency control in databases. These transactions deal with non-real data but reads only real data (Idoudi, Duvallet, Sadeg, Bouaziz, Gargouri, 2009). Real-time databases are designed for specific purpose, therefore, design of a real-time database needs special architecture to push real-time updates to the clients in the push-based style, so the user can stay up-to-date with information they needed.

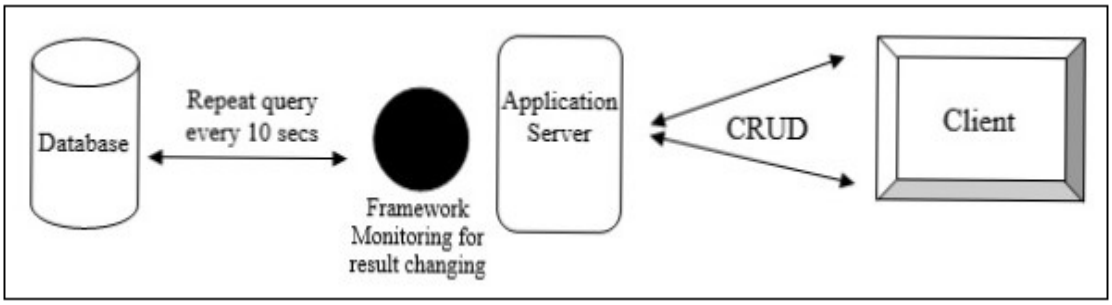
Problem Statement

with the significant advances in databases, the data can be processed so fast and there is an increasing demand for various databases to process data in a real-time fashion. When remote access to data is considered in a Database Management System environment for detecting query results changes, data access delay becomes one of most serious problems in meeting deadlines like stock exchange market or air traffic control system. Currently, different push techniques are used as follow:

i. Polling database at a specific interval of time or when data is updated, it is sent to the client automatically in push-based style, where it is displayed as updated data or adding them into old data.
ii. Data updates only occur at the server-side.
iii. Automatically updates the data which clients require by using logical predicates.

However, the reason behind these types of databases performing real-time push-based using programmed pull techniques, which involves middleware for observing changes via polling i.e. the client machine is in contact with server periodically specific intervals of time. The major drawback of this technique is that it concerns with specific interval of time for refreshing updates rather than data Fig 1.

Figure 1 Traditional Result Changes Monitoring Using Polling Technique



In this paper, we purpose a new publish/subscribe architecture for real-time database that has unique features which is based on scalable push-based query processing in databases, providing the following benefits:

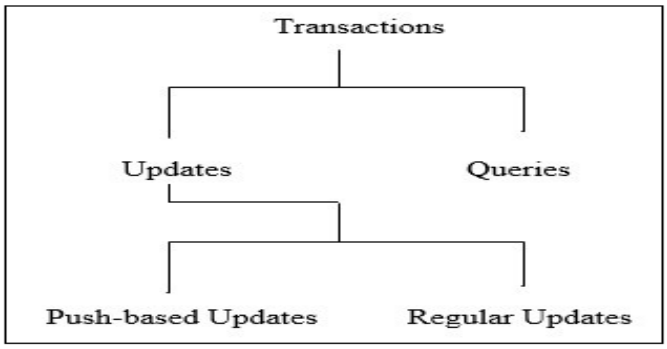
- ✓ Push Based Notification Change: Our "proposed architecture provides real-time notifications" to subscribed clients as an "additional feature to existing" databases in standalone systems.
- ✓ Scalable: The system is capable of maintaining continuous queries as well as update throughput to scale out the system for queries handling.
- ✓ Viable Query Engine: Through a viable query engine, the proposed technique is applicable to different databases, not only system-specific.

In our proposed architecture, clients just define their parameters in the query for one time and get the subscribed query changes notifications of those data without refreshing or querying the database again for up-to-date results. The "rest of the paper is organized as follows: in the next" section, we discuss about the database processing and transaction scheduling techniques. The section 3 describes scenarios and describes existing database issues that provide notifications updates for real-time monitoring. Then, we describe the experimental methodology and describe the overall architecture for push-based query notifications changing for real-time databases in section 4. We discuss related work in section 5, and the conclusion can be found in section 6.

TRANSACTIONS IN REALTIME DATABASES

In many applications, an update to the database may be of interest to other clients in the system. For such type of requirements, updated data must be pushed to all subscribed clients. Such type of updating data is done through push-transactions. For assumption of such updated data, it must have timing constraints associated with them. Transaction processing on which our model is based has been discussed. Figure no 2 shows types of transactions being used for processing in our proposed architecture: regular updates, push-based updates, and queries. The push-based updates concern with automatic data update mechanism while regular and query updates concern with client-server DBMS. The push-based updates have time constraint while other does not support the timing constraints. Henceforward, the deadlines are only imposed on push-based updates (push transactions) since our goal is to meet the timing constraints to provide the up-to-date information to clients.

Figure 2 Types of Transactions



Transaction Scheduling Techniques

Transaction scheduling in real-time databases "plays an important role in deciding the performance" of RTDBS system. The performance is measured in terms of transactions completed per unit time (Choudhary, 2013). Priorities in RTDBS "can be assigned in the number of ways", but most commonly used techniques are:

First Come First Serve

This is a "top priority assignment scheme where the priorities are assigned according to its arrival" time. First arriving transaction has assigned higher priority. The main disadvantage of this rule is this delivery "scheme is that it does not use the deadline information". The FCFS will be discrimination in contradiction of a newly discovered transaction, with an urgent term an old transaction that may or may not have later no short period of time. This is not suitable for "real-time systems", but we consider FCFS for comparing with other schemes.

Earliest Deadline First

In EDF, the highest priority will be assigned to those transactions with the closest deadline and the one with the longest delay to the lowest priority. This priority allocation system is known to achieve "better performance in the real-time systems under low and moderate load levels. However, one disadvantage of EDF is that it can assign the highest priority to task that will miss its deadline". When it happens, "system allocates resources to a transaction that cannot meet its deadline in favor of a transaction that could meet its due date. Thus, EDF may not be desirable for heavily loaded systems".

Minimum Slack Time First

The "slack time of a transaction is the cushion that can" be captivated by random delays without "missing the deadlines. Like, if t_a is the arrival time, E is the estimated execution time and d is the deadline, then the slack time at the time of arrival of a transaction is $S = d - \{t_a + E)$. If the transaction has already received some service, say R, then $S = d - \{t_c + E - R)$, where *it* is the current time. A negative slack time implies that the transaction has either already missed its deadline" or when it has been estimated that it cannot meet its deadline. The transaction with the minimum slack time is scheduled first by this technique.

Push Transactions

A push transaction defines the operations performed at the client's cache on database object copy. When the push transaction has arrived at the client-side, it executes the transaction locally to maintain the consistency of the database object copy. Push based transaction is assigned highest priority than regular transactions as they have meeting deadlines requirements. Push transactions are classified into two types, absolute and relative updates. The absolute transactions have fixed values while relative updates have dynamic values. Absolute updates are pushed to subscribed clients at any time without any risk. But, relative updates are pushed to the clients in order as they are performed, for successful execution.

THE SCENARIOS

There are many applications that requires up-to-date data as the data changes, such type of applications relies on the server-side notifications for monitoring the changes without requiring them to know where and when to look for changes, which are categorized as follow:

- ✓ Notifications: Notifications update to subscribed clients upon the result changes in the user query so that the clients or users will stay up-to-date about occurred changes.
- ✓ Invalidations: When the results of query are changed it becomes invalidated in caches of memory. Query results are also served by caches and database so that the workload is taken off and speed of query results will be increased by caches located near the users.

The commonly requested queries are kept in a separate table to relieve primary database called materialized or logical views. Amendment in querying maintenance of database to retrieve data is possible which may not be otherwise available. Traditionally, pull-based databases are incapable of push-based notifications to notify the clients about the change of notifications in real-time fashion and have one of the critical issue in the last years. Since early models are based on append-only strategy (Terry, Goldberg, Nichols & Oki, 1992), while modern designs are based on the updated and removed statistics, therefore, applications buildings become easy. Complex Event Processing CEP (Abadi, Carney, Çetintemel, Cherniack, Convey, Zdonik, 2003; Abadi, Ahmad, Balazinska, Cetintemel, Cherniack, Hwang, Tatbul, 2005**)** systems are designed especially for complex events handling like fraud detection from low-level events or sensors malfunctioning. Query does not have only constraints, but also have temporal limitations of the data or any relationship between the different events.

While databases that eternally store data and then provide updates are based on streams of ephemeral data and can just holds state that is derived out like aggregate functions in the memory little period of time. The databases that are based on time-series (Dunning & Friedman, 2014) are designed to save and respond to unlimited set of different events in the form of functions when it happened like sensors data is indexed by time. Although they permanently save data and some data in form of continuous querying techniques for example Influx DB3, they are typically designed for OLAP queries, virtualized views and for no expanding of the notifications change. Meteor is a platform for building web applications, which provides reactive notifications change to subscribed queries, which uses MongoDB as bases for self-maintaining results of queries. Meteor application server checks each continuous query periodically for comparing actual and reflected to detect results changes and only sends relevant updated information to users. Hence, poll-and-diff technique added latency of several seconds in results update. Also, it increases load on application server and database.

The parse is an application development framework which is also based on MongoDB, similarly to providing result changes and notifications. This involves the multiplication across the distributed level of the machines, but this also adds limitation of single-node Redis illustration reserved for messaging (Parse & Scalability, 2016). Database services accommodated by Parse are going to fall down in January 2017 and number of people contributing to the development has been decreasing over the previous months (Parse & GitHub, 2016). Other contributors have also publicized the provision of the Parse Software Development (Glenn, 2016) while in future Parse platform is un-predictable. The another distributed SQL database is the Oracle 11g with change of notifications in complex queries that provisions streaming joins with the firm restrictions (Witkowski, Bellamkonda, Liang, Sheng, Smith & Yu, 2007). The continuous queries involving materialized views are sustained by applying dedicated changed operations on-request, periodically or upon the commit transactions (Alpern, Arora, Barclay, Bronnikov, Chang, Chen & Elson, 2011).

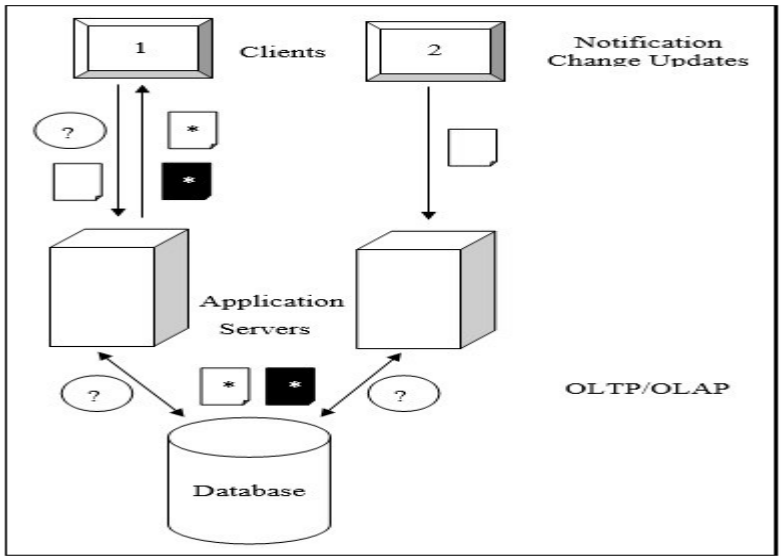
This also has scalability issue due to the strict consistency necessities and distributed-disk architecture. Cloud information base that delivers notifications concerning modified information, however, it solely provides limited question quality owing to the limit underlying information model that desires information to be ready during a tree of objects and lists. Pipeline sound unit improved PostgreSQL by alert notifications for complicated queries. Whereas the open-sourced version will solely run on one node, the enterprise version provisions a clump mode that partition continuous views and connected calculation across completely different machines. But, all the DML (Insert, Update and Delete) operations are accessed synchronously via two-phase commit among

all nodes **(**Pipeline 2015) while Pipeline DB Enterprise version is only scalable up-to modest level of cluster sizes.

PROPOSED MODEL AND TECHNIQUES

The publish/subscribe communication model (Agarwal, 2014) is based on asynchronous messaging paradigms which provide larger scalability of the distributed systems. This model helps to meet challenges of providing push-based notifications, efficient storage, and scalability between different systems. In this age of modern application development especially web applications, that requires application servers to provide real-time updates proactively to the subscribed clients. However, in traditional databases, they use pull-based strategies, other frameworks and implements APIs for streaming real-time updates and provide real-time push-based updates to the subscribed clients. Monitoring for result changes in query is very difficult to handle while detecting for changes in individual is easy. The push-based design is very difficult to implement on the top of available infrastructure, presently databases typically supports pull-based techniques to retrieve data. Therefore, our proposed architecture model that handle query change notification for subscribed client, which is based upon push-based strategy for real-time query using continuous queries.

Figure 3: Proposed Model Architecture



In this proposed architecture shown in Figure 4.2, common OLAP/OLTP tasks are managed by application servers that deal with the database rather than the client's direct access. To manage additional real-time tasks, application server maintains continuous queries and generates notifications whenever results changes in the database. To design continuous questions that keep stands itself, the appliance server needs to supply the period scheme with all necessary information i.e. initial question results and complete information objects on each write operation. every continuous question is assessed once direct and at the moment sent to the information aboard all matching objects. Every DML operations, i.e. every insert, update and delete, is distributed to the information at the side of a complete after-image of the written object, i.e. with the whole information object when the operation has been performed. Apart from this, associated application server subscribes to notices for the continual queries of its shoppers and forwards them dependably.

The task of similar the results of arriving operations against every continuous query are executed in appropriated form and subdivided both by written objects and look after the

queries. In this way, each processing node is responsible for a subset of entirely queries and a subset of each operation. Changes are noticed based on whether an item used to be a match and whether it still is a match for a query. For each change, a notification is referred to the subscribed clients. The planned system proposal eliminates the barrier of resource wants additionally as failure domains for key storage (persistent knowledge, pull-based access) from one viewpoint and period options (change notifications, push-based access) on the opposite. in the meantime, the continual question capability is handled in period information, for incessantly keeping question results will be earned, whereas determined knowledge could be unbroken in a very powerfully reliable single-node system. Utilizing a planned design and asynchronous communication throughout the important process path, we have a tendency to avoid obstacles and therefore succeed linear quantifiability and low latency.

CONCLUSION

This report provides an understanding of real-time database for building application like web applications in real-time manners, by presenting proposed model which improve efficiency, scalability of RTDBS to subscribed clients in push-based way as the changes occurred in subscribed queries it passes it to the client, so the user can stay up-to-date with refreshed data without reconnecting with server. The push-based architecture of real-time database increases the usefulness of query change notification and reduces the difficulty of traditional database. Therefore, real-time databases are designed to develop and implement reactive applications, since they keep user's critical information updated and notify about changes. In this paper, we deeply study the fundamentals of traditional as well as real-time databases and illustrate the benefits of real-time databases. Through the research work presented a real-time database model and proposed a new model for a push-based querying identified. We figure out the hurdles in implementing real-time databases nearly, to enables clients to define data and stay informed about. We proposed an architecture for push-based databases that is scalable and low latency. And the final conclusion is that this is the best solution for real-time application.

REFERENCES

Abadi, D. J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J. H., & Tatbul, N. (2005). The Design of the Borealis Stream Processing Engine. *In CIDR, 5, 277-289).*

Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., & Zdonik, S. (2003). Aurora: A new model and architecture for data stream management. *The VLDB Journal: The International Journal on Very Large Data Bases, 12(2),* 120-139.

Agarwal, N. (2014). The Publish-subscribe based communication model (Doctoral dissertation).

Alpern, D., Arora, G., Barclay, C., Bronnikov, D., Chang, T., Chen, L., & Elson, D. (2011) Oracle Database Advanced Application Developer's Guide, 11g Release 2 (11.2) E10471-04.

Buchmann, A. (2006). The Real-Time Databases, Idea Group.

Choudhary, S. R. (2013). Performance Evaluation of Real-Time Database Systems in Distributed Environment. *The International Journal of Computer Technology and Applications,* 4(5), 785.

Dunning, T., & Friedman, B. E. (2014). Time series databases: new ways to store and access data. O'Reilly Media.

Glenn, G. (2016). Azure welcomes Parse developers. Microsoft Azure-Blog, February 2016.

Idoudi, N., Duvallet, C., Sadeg, B., Bouaziz R., & Gargouri, F. (2009). How to model a real-time database? *In: Proceedings of the 12th IEEE international symposium on object-oriented real-time distributed computing (IEEE ISORC'2009). Tokyo, Japan:* IEEE; a. p. 321–5.

Parse. Scalability. Parse Live Query documentation, 2016. Accessed: 2016-09-20. Parse. GitHub: contributors to Parse Platform/parse-server. 2016. Accessed: 2016-09-19.

Pipeline, D. B. (2015). Two-Phase Commits. Pipe-line DB Enterprise documentation, 2015.

Ramamritham K. (1993) Real-time databases. *Journal of Distributed and Parallel Databases (Springer), 1(2):*199–226.

Stankovic, J. A, Son S. H., & Hansson, J. (1999). The Misconceptions about real-time databases, *IEEE Computer*, 32 (6): 29–36.

Terry, D., Goldberg, D., Nichols, D., & Oki, B. (1992). *Continuous queries over append-only databases,* 21 (2), 321-330). *ACM*.

Witkowski, A., Bellamkonda, S., Li, H., Liang, V., Sheng, L., Smith, W., & Yu, T. (2007). Continuous queries in oracle. *In Proceedings of the 33rd international conference on Very large databases (1173-1184). VLDB Endowment*.