Comparison of query execution time between Streaming Database and Relational Database Management System

Pavol Podstreleny and Morales Diego

Computer Science and Information Engineering Department,
National Taiwan University of Science and Technology, Taiwan
M10715804@mail.ntust.edu.tw and E10715004@mail.ntust.edu.tw

Abstract-- Nowadays most systems focus its efforts in store high volumes of data in different schemas: databases, file systems or other forms of massive storage. On the other hand there are many situations in the world where real-time analysis needs to be done. For this situations, Data Stream Management Systems, which prioritize data processing and computation over data storage, can be used. In this paper we analyze query execution time of Data Stream Management System and Relational Database System, especially we compared query execution time between PostgreSQL view, materialized view and PipelineDB continuous view in the specific use-case. Results showed that in our selected use-case the continuous view performed better than PostgreSQL view. Both materialized and continuous views had a relatively small and constant cost of query. Materialized refresh time problem was showed and discussed in result section too.

Keywords-- PipelineDB, PostgreSQL, Data Stream Management Systems, Streaming Databases

I. INTRODUCTION

Data volumes are growing rapidly, and this implies a huge challenge for all the businesses. [1] The world gathers a lot of use cases where high volumes of data are collected and used. All of them share the principle of "*process data and extract actionable insights from it*".[2]
Some of the process may take quite a lot of time to generate the corresponding output. In some use cases is much useful to know the results faster. Stream Processing is a computer programming paradigm, equivalent to data-flow programming event stream processing and reactive programming allows some applications to more easily exploit the data. From analyzing large volumes of data than was previously possible, to analyzing data in motion, whether the industry of concern is telecommunications, health-care or utilities, technologies for big data are needed.
The input to it must be processed in such a way that the quality data yields quality effective results. [2] Quality of Big Data has become an important factor to ensure that the quality of data is maintained at all Big data processing phases. [3]
Over past year more new technologies have been introduced to cope with the data-in-motion management. During past few years growing trend of companies interested in SQL streaming database systems could be observed. In this project report we focus on comparison between SQL streaming database system and relational database system, especially we compare PostgreSQL and PipelineDB systems that was created by same company. Real-time analysis use-case were created where query execution time of different views was measured. This use case should provide illustration for company which data are rapidly growing and real-time analysis is needed.
The main objectives of this project report is to benchmark the performance and characteristics of PipelineDB with PostgreSQL.

II. RELATED WORKS

Pre-processing data before performing any analytics is primeval. However, several challenges have been experienced at this essential phase of the big data value chain [4]. Data quality is one of them and it need to be highly considered in the context of big data. [1]
There are few experiments about the comparison between these kind of database systems, however there are similar experiments where they has tried to improve data quality. There are two strategies (1) data-driven and (2) process-driven. The data-driven strategy deals with the data as it is, using techniques and activities as cleansing to improve its quality. On the other hand, Process-driven attempts to identify the origin sources of poor data quality and redesign the process of the way data is created or recorded. [2]

III. METHODOLOGY

In order to analyse and compare the performance of PipelineDB against PostgreSQL in some specific queries, experimental use case was created. Detailed information about implementation of use-case are described below in the sub-section. Use-case chosen for experiments is focused on real time analysis, more precisely, on "movie" company that could potentially monitor popularity of different movie directors in the real time.

3.1 Use-case implementation

In the text below, schemas of tables, stream [6], continuous view [7], continuous transform [8], view and materialized view, used throughout the experiment, are shown.

```
CREATE TABLE movie_director
(movie_id  integer not null, name varchar(15)
not null, PRIMARY KEY(movie_id,name));

CREATE FOREIGN TABLE likes_stream
(movie_id integer, likes integer)
```

```
SERVER pipelinedb;

CREATE VIEW likes_ct WITH (action=transform)
AS SELECT t.name, l.likes FROM likes_stream l JOIN
movie_director t ON l.movie_id = t.movie_id;

CREATE VIEW likes_cview WITH (action=materialize)
AS SELECT name, sum(likes) as popularity FROM
output_of('likes_ct') GROUP BY name;

CREATE TABLE likes_table
(movie_id integer, likes integer);

CREATE MATERIALIZED VIEW likes_mview
AS SELECT name, sum(likes) as popularity
FROM likes_table l JOIN movie_director t ON
l.movie_id = t.movie_id GROUP BY name;

CREATE VIEW likes_view AS SELECT name, sum(likes)
AS popularity
FROM likes_table l JOIN movie_director t
ONl.movie_id = t.movie_id GROUP BY name;
```

In the PipelineDB scenario showed in Figure 1, incoming data in the stream (`likes_stream`) are read by continuous transform (`likes_ct`) that joins table (`movie_director`) with this stream. Output of this continuous transform represents another streams of data that is used by continuous view (`likes_cview`).

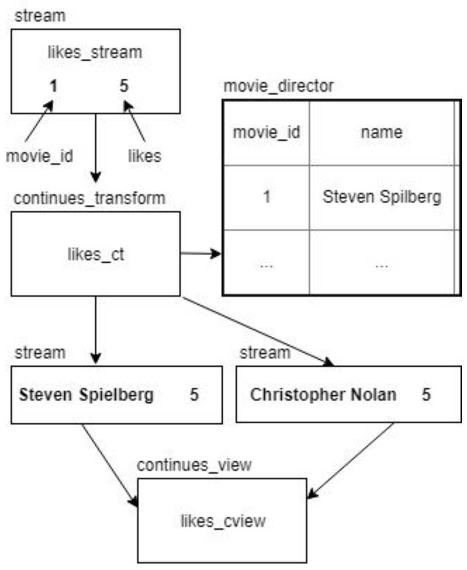

Figure 1 .PipelineDB Use-case data flow diagram

On the other hand PostgreSQL can not work with streams, continuous views and continuous transforms. Therefore another table (`likes_table`) with similar schema was created to substitute (`likes_stream`) stream. Moreover PostgreSQL's materialized view (`likes_mview`) and view (`likes_view`) were created in order to compare these type of views with continuous view in the experiments. Data flow of PostgreSQL is shown in the Figure 2.

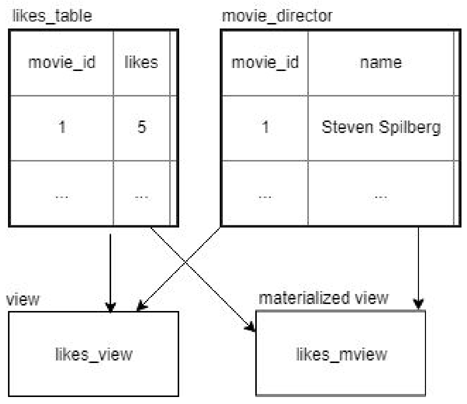

Figure 2. PostgreSQL Use-case data flow diagram

Throughout the first experiment some randomly generated data were added to stream/tables and afterwards “select query” described below were executed. These steps were repeated multiple times. Main objective was to measure and compare difference in query execution time between PipelineDB continuous view and PostgreSQL view, materialized view.

```
EXPLAIN ANALYZE SELECT * FROM likes_cview ORDER BY
popularity desc;

EXPLAIN ANALYZE SELECT * FROM likes_view ORDER BY
popularity desc;

EXPLAIN ANALYZE SELECT * FROM likes_miew ORDER BY
popularity desc;
```

In the second experiment, refresh time of materialized view was analyzed as number of data in the table were increasing.

IV. RESULTS

The result of the first experiment is shown in the Figure 3. Execution time of querying continuous view and materialized view did not change as number of data in the stream/table were increasing. On the other hand, increasing trend of query execution time on view could be observed as data were added into table. Querying the results in the continuous view was similar to querying a materialized view in standard SQL, it is fast and efficient.

The main difference with PipelineDB in this case was that while the continuous views were updated on the fly and access time is fast and constant, views need to be computed every time they are asked. This add a computation cost that can be noticed in the Figure 3.

One of the main differences between PipelineDB and PostgreSQL is that PipelineDB does not store streaming data in order to do analysis. On the other hand PostgreSQL streaming data has to be kept in the table.

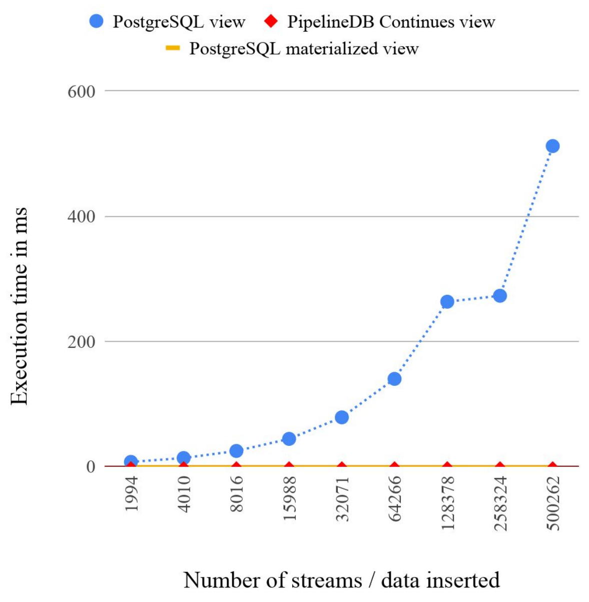

Figure 3. Measurement of execution time of PostgreSQL view, materialized view and PipelineDB continuous view

Materialized views store the result and only refresh its content in specific circumstances. It does not have capability of updating itself after every insertion. In this situation trigger could be created on likes_table to update materialized view every time there is insertion. Problem with this approach is that update of materialized view could be in progress when another insertion into table could occur. In this situation trigger could refresh the materialized view when a refresh is already in the progress. Materialized View needs to be refreshed before every view and the time that this refresh takes is proportional to the size of the table as shown in the second experiment in the Figure 4, since the refresh operation completely replaces the contents of a materialized view.

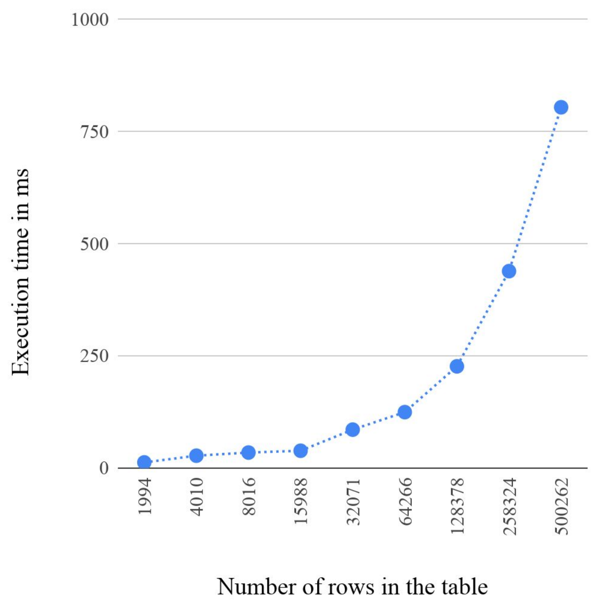

Figure 4. Measurement of execution time of refresh in PostgreSQL materialized view

V. CONCLUSION

In this project report streaming database system PipelineDB and PostgreSQL relational databases system were analyzed. The study focused on comparison of query execution time between PostgreSQL view, materialized view and PipelineDB continuous view. Thanks to PipelineDB that offers fast and high throughput data analysis without storing the stream data on any table and its internal view (continuous view), it can update itself regularly. That is why, the results shows that in our selected use-case the continuous view performed is better than PostgreSQL view due to while the data increasing, PostgreSQL view spends more time of execution. Both materialized and continuous views had a relatively small and constant cost of query. Disadvantages of using materialized view in real time systems is refresh time. Results shows that refresh time is increasing as number of data in the table increases. With this results we are able to support our demonstration of continuous view, materialized view and view, to provide which is better for companies and systems and keep their databases optimized and safe.

VI. REFERENCES

[1] Taleb Ikbal, Dssouli Rachida, “Big Data Pre-Processing: A Quality Framework”, IEEE International Congress on Big Data, pp. 191-198, 2015
[2] Gu Lin, Zeng Deze, Guo Song, Xiang Yong, Hu Jiankun, “A General Communication Cost Optimization Framework for Big Data Stream Processing in Geo-Distributed Data Centers”, IEEE Transactions on Computers, vol. 65, No. 1, pp. 19-29, 2016
[3] H. Hu, Y. Wen, T.-S. Chua, and X. Li, “Toward Scalable Systems for Big Data Analytics: A Technology Tutorial,” IEEE Access, vol. 2, pp. 652–687, 2014.
[4] Lee Yong-Ju, Lee Myungcheol, Lee Mi-Young, Hur Sung Jin, Min Okgee, “Design of a Scalable Data Stream Channel for Big Data Processing”, Big Data SW Platform Research Department, pp. 537-540, 2015
[5] PipelineDB: Streams [online]. [cit. 2018-12-26] from: http://docs.pipelinedb.com/streams.html
[6] PipelineDB: Continuous views [online]. [cit. 2018-12-26] from: http://docs.pipelinedb.com/continuous-views.html
[7] PipelineDB: Continuous transforms [online]. [cit.2018-12-26] from:http://docs.pipelinedb.com/streams.html