ISSN 2395-1621

Improved Distributed Query Processing

[#1]Prachi Bhonde, [#2]Saylee Dalu, [#3]Viraj Deshmukh, [#4]Tushar Shedge

[#1234]Department of *Computer Engineering*

AISSMS's IOIT

Savitribai Phule Pune University, Maharashtra, India

ABSTRACT

Systems like distributed message queue and stream processing platform are being used for scaling huge number of partitions of data streams and on the commodity hardware, this data streams are having high velocity. API used for programming by these systems is low level, so requires more coding which increases the maintenance and learning time of the programmer. These systems don't have the sufficient capability of querying in SQL like Hive, Impala or Presto big data systems. Here we are defining the minimal extension set to standard SQL for manipulation and querying of data streams. Streaming SQL have the prototype of above extensions. A tool for streaming SQL that compiles streaming SQL into physical plans performed on Samza which is an open-source distributed stream processing framework. Here we are comparing the performance of streaming SQL queries with similar Samza applications and discussing the improvements in usability.

Keywords-Samza Sql, Streaming data, Apache Kafka, Apache Zookeeper, Apache Calcite, Yarn

ARTICLE INFO

Article History

Received: 29th May 2017

Received in revised form :

29th May 2017

Accepted: 31st May 2017

Published online :

31st May 2017

I. INTRODUCTION

The architectures like Lambda and Kappa were time consuming. To overcome this problem proposed system has capability to execute query in less time with distributed query processing. It will make work of hours in minutes. This query processing model is a scalable and fault-tolerant. SQL based streaming query engine is implemented on top of Apache Samza. This project will be used to explore a unified framework which enables Kappa architecture style data processing pipelines based on well-known standard SQL. In this proposed architecture apache Samza and Apache calcite provide core functionality. Samza provides developers with a Java API similar to Map/Reduce for implementing streaming tasks, and a message serialization and deserialization API called Serde API to support different message formats (tektite, Avro, JSON or Thrift). While regular Samza jobs read and write to/from Kafka, Samza provides a separate Java API to plug in different input and output systems. A high-level view of Samza architecture. The features of Samza discussed here are available for proposed model to utilize it for executing streaming SQL queries. Samza comes with a built-in YARN client for submitting Samza jobs to a YARN cluster, and apache Samza job has an application master to perform input partition assignments and task scheduling. The Samza application master also takes care of fault tolerance. A query is a Samza job with proposed system's specific stream task implementation that performs the computation described in the query.

II. LITRATURE REVIEW

Milinda Pathirage [1] Technologies such as distributed message queues and streaming processing platforms that can scale to thousands of data stream partitions on commodity hardware are a response. Programming API provided by the previous systems are often low-level. That will increase programmers overhead to maintain code and learn new coding standards/syntaxes. Also there is lack of SQL querying capabilities which are popular on Big Data Systems. Here they are defining minimal set of extensions to standard SQL. It will support streaming queries.

Dmitry Namiot [2] This paper's goal is to provide a quick introduction and survey of the technical solutions for big data streams processing. In this survey, Machine to Machine communications, sensors data in Internet of Things as well as time series data processing. They have discussed the basic elements behind data streams processing. Existing technical solutions for implementation of data stream processing are also discussed.

Oscar Boykin [3] Summing bird is an open-source domain-specific language. It is implemented in Scala and designed to integrate online and batch MapReduce computations in a single framework. Hadoop can operate efficiently for batch processing and Storm for online processing. Summingbird can operate in a hybrid processing mode i.e. it can combine batch as well as online processing results. It imposes constraints on type of aggregations that can be performed, but these constraints are not found restrictive for broad range of analytics tasks at Twitter.

Supun Kamburugamuve [4] There is huge number of applications through which large amount of data generated in external environments is stored on servers for real time processing. These applications includes stock trading, sensor based monitoring, web processing, network monitoring and so on. This data generated from various sources can be seen as stream of events or tuples. In stream based applications data is handled as sequence of event tuple.

Qian Lin [5] In this paper they have proposed a noval model for stream joins, called join-biclique. It uses large cluster as bipartite graph. Join-biclique has several advantages over state of the art techniques such as memory efficiency, elasticity and scalability. These are essential features for building scalable and efficient streaming systems. Depending on join-biclique they have developed scalable distributed stream join system called BiStream. It supports full history joins, window join and also online data aggregation.

Mariam Kiran [6] Paper combines ideas from database management, cost models, query management and cloud computing to present a general architecture that could be applied in any given scenario where affordable online data processing of Big Datasets is needed. The results showcase a reduction in cost and argue benefits for performing online analysis and anomaly detection for sensor data.

Leonardo Neumeyer [7] S4 is a general-purpose, distributed, scalable, partially fault-tolerant, pluggable platform that allows programmers to easily develop applications for processing continuous unbounded streams of data. We show that the S4 design is surprisingly flexible and lends itself to run in large clusters built with commodity hardware.

Matei Zaharia [8] This paper says that big data applications should act on real time data. Also these applications should be fault tolerant and scalable as we are using them on larger scales. They should handle stragglers automatically. But distributed systems at that time were not fault tolerant and also recovery was too expensive or time consuming. Also they were not handling stragglers so new model was proposed called D-stream i. e. discretized stream that overcome these challenges.

Daniel J. Abadi [9] This paper tells about Aurora's basic processing model and architecture. It is new system for monitoring applications to manage data streams. There are various types of monitoring applications and variety of data is being produced from them. For management of this data Aurora architecture and stream oriented set of operators are discussed.

Arvind Arasu [10] In this paper CQL (Continuous Query Language) is discussed which is supported by STREAM (Stanford Data Stream Management System). CQL is SQL based query language that support querying on data streams as well as updatable relations. They begun by presenting an abstract semantics that relies only on black box" mappings among streams and relations. From these mappings we define a precise and general interpretation for continuous queries.

III.PROPOSED SYSTEM

In our system we provide the sql query as a input for system, system first interact with SamzaSQL using Sqline library the query planner convert the query into the tuple format and send to the input for sibling window operation logic. Each tuple is having some timestamp, by using that timestamp system identify time interval the omits the tuple and send the result to output window.

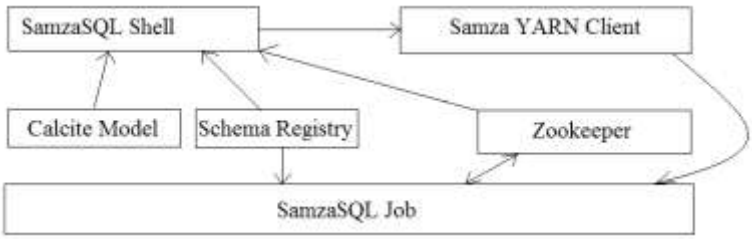

Fig. Streaming SQL Architecture

Figure illustrates out proposed system i.e. Streaming SQL architecture. Users interact with Streaming SQL through a special SQL shell build using SqlLine 3 library and a custom Streaming SQL specific JDBC driver implementation. SamzaSQL shell is a command line application that runs on users' desktop. SamzaSQL JDBC driver wraps the query planner that converts a streaming SQL query to a Samza job containing one or more SamzaSQL tasks. Query planner uses Samza YARN Client to submit streaming jobs to a YARN cluster. Query planner uses Calcite model files described in JSON format and Kafka Schema Registry to retrieve necessary metadata for query planning and uses Zookeeper to share metadata and configuration information between query planner and SamzaSQL streaming tasks. Information shared via Zookeeper
includes streaming SQL query to use during task query planning, Schema Registry location, message schema details, etc. We are also planning to utilize Zookeeper to store SamzaSQL shells session information including running queries, and including input/output streams and the status of the query.

IV. RELATED WORK

Stream:
The samza container reads and writes messages using the SystemConsumer and SystemProducer interfaces. You can integrate any message broker with Samza by implementing these two interfaces. Out of the box, Samza supports Kafka (KafkaSystemConsumer and KafkaSystemProducer). However, any message bus system can be plugged in, as long as it can provide the semantics required by Samza, as described in the javadoc. SystemConsumers and SystemProducers may read and write messages of any data type. It's ok if they only support byte arrays — Samza has a separate serialization layer which converts to and from objects that application code can use. Samza does not prescribe any particular data model or serialization format.

Windowing:
Sometimes a stream processing job needs to do something in regular time intervals, regardless of how many incoming messages the job is processing. For example, say you want to report the number of page views per minute. To do this, you increment a counter every time you see a page view event. Once per minute, you send the current counter value to an output stream and reset the counter to zero. Samza's *windowing* feature provides a way for tasks to do something in regular time intervals. If you need to send messages to output streams, you can use the MessageCollector object passed to the window() method.

In this example we have taken model.json as input file:

```
{
  "version": "1.0",
  "defaultSchema": "SALES",
  "schemas": [
    {
      "name": "SALES",
      "type": "custom",
      "factory":
"org.apache.calcite.adapter.csv.CsvSchemaFactory",
      "operand": {
        "directory": "sales"
      }
    }
  ]
}
```

It contains EMPS and DEPTS csv files. In following images we are displaying the results of various queries. There is also comparison between calcite query and Samza query. Execution time is displayed.

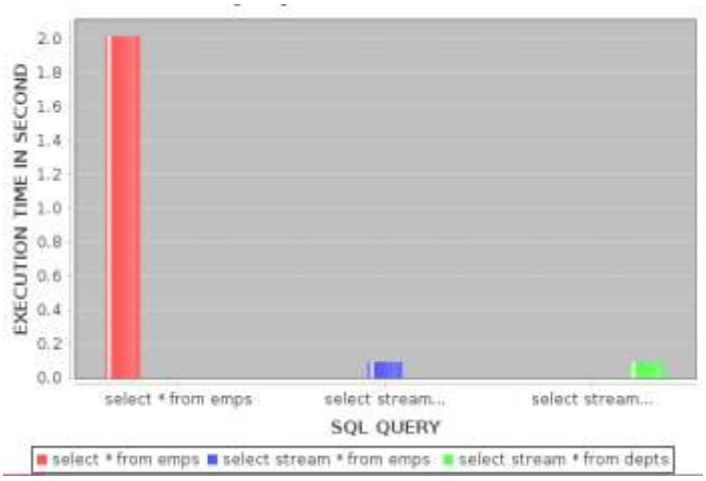

Fig.: Simple select query comparison

In above example we are displaying records of EMPS using simple calcite query and Samza Query. Also we are displaying records of DEPTS using Samza query.

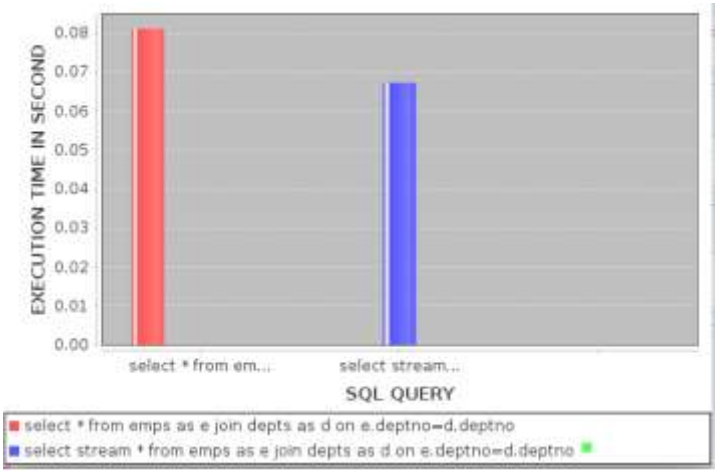

Fig.: Comparison of join queries.

in above example we are displaying comparison between two join queries on EMPS and DEPTS.

V. CONLUSION

Streaming SQL is SQL implementation on top of Apache Samza which utilize standard SQL as much as possible by moving streaming related details away from the language layer to the physical execution layer. Streaming SQL demonstrates the advantages of having access to check pointed local storage. We have implemented this system to increase throughput and increase efficiency of system.

REFERENCES

[1] Milinda Pathirage, Julian Hyde, Yi Pan and Beth Plale "SamzaSQL: Scalable Fast Data Management with Streaming SQL", 2016 IEEE

[2] Dmitry Namiot, "On Big Data Stream Processing", vol. 3, no. 8, 2015
[3] O. Boykin, S. Ritchie, I. O'Connell, and J. Lin, "Summingbird: A framework for integrating batch and online Mapreduce computations," Proceedings of the VLDB Endowment, vol. 7, no. 13, 2014.
[4] Supun Kamburugamuve, "Survey of Distributed Stream Processing for Large Stream Sources", For the PhD Qualifying Exam 2013
[5] Qian Lin, Beng Chin Ooi, Zhengkui Wang, Cui Yu, "Scalable Distributed Stream Join Processing", 2015
[6] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, Sartaj Singh Baveja "Lambda Architecture for Cost-effective Batch and Speed Big Data processing", 2015 IEEE International Conference on Big Data
[7] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in Data Mining Workshops (ICDMW), pp. 170–177, International Conference on. IEEE, 2010.
[8] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica, "Discretized Streams: Fault-Tolerant Streaming Computation at Scale", Nov. 3–6, 2013
[9] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," The VLDB Journal-The International Journal on Very Large Data Bases, vol. 12, no. 2, pp. 120–139, 2003.
[10] A. Arasu, S. Babu, and J. Widom, "The cql continuous query language: Semantic foundations and query execution," The VLDB Journal, vol. 15, no. 2, pp. 121–142, Jun. 2006

