Streaming platform for traditional database workloads

Rakesh Reddy Yempalla
Computer Science
University of South Florida
Tampa, Florida, USA
rakeshreddy112@usf.edu

Voohitha Morampudi
Computer Science
University of South Florida
Tampa, Florida, USA
voohitha@usf.edu

Mythreye Pasulla
Computer Science
University of South Florida
Tampa, Florida, USA
mythreyep@usf.edu

Abstract : In response to the rising popularity of data streaming over batch processing, this paper investigates the increasing demand for real-time data processing. Our study focuses on implementing a robust streaming setup that ingests TPCH benchmark data and processes it within a single-node computational environment. The processed data is seamlessly integrated into a conventional PostgreSQL database. To evaluate system performance, we execute a suite of 22 predefined queries from the TPCH benchmark, serving as a performance benchmark for the new streaming infrastructure. This research aims to provide insights into the advantages and potential limitations of real-time data processing compared to traditional batch processing methodologies o.

Keywords – TPC-H , PostgreSQL, Data Streaming,

Real Time processing, Kafka.

Introduction:

In today's data-driven world, the demand for real-time data processing has grown exponentially. As organizations strive to make timely decisions, detect emerging trends, and respond swiftly to dynamic data patterns, the need for efficient real-time data processing solutions has become increasingly critical.This study embarks on a mission to bridge the gap between traditional batch processing and real-time data processing by implementing a robust streaming setup.Data streaming involves processing and analyzing data in real-time as it's generated, enabling immediate insights and actions. In contrast, batch processing collects and processes data in predefined chunks or batches, causing a delay between data arrival and analysis. Streaming is ideal for real-time applications, while batch suits offline analysis [2]. Our primary objective is to develop a system capable of seamlessly ingesting TPCH benchmark data and processing it in real-time. By doing so, we aim to harness the power of streaming data to provide immediate insights and enable rapid decision-making.

To achieve this goal, we will explore the integration of streaming data technologies, such as Kafka, with a conventional PostgreSQL database. The TPCH benchmark, a well-established industry standard for evaluating database performance, will serve as the foundation for our study [1]. We will adapt the benchmark's queries and data to fit the real-time streaming paradigm, enabling us to evaluate the performance of this new streaming infrastructure.This research endeavors to shed light on the advantages and potential limitations of real-time data processing when compared to traditional batch processing methodologies. By the end of this study, readers will gain valuable insights into the transformative potential of streaming data in handling traditional database workloads, paving the way for data-driven organizations to thrive in a rapidly evolving digital landscape.

with a feature, a voice narration detailing the thing being clicked is offered. This feature is very useful for users who are entirely blind since it improves their involvement and comprehension of the information, which further improves their overall platform experience.

Background:

TPC-H benchmarking employs a comprehensive data model comprising eight interconnected tables, representing entities such as orders, line items, customers, and suppliers. These are some of the features in TPC-H benchmarking:

Data Model: TPC-H comprises 8 tables representing entities such as orders, line items, customers, and suppliers. These tables are interconnected to form a relational schema, reflecting the structure of real-world transactional data. The dataset typically spans various sizes, measured in gigabytes or terabytes, to simulate diverse workloads.

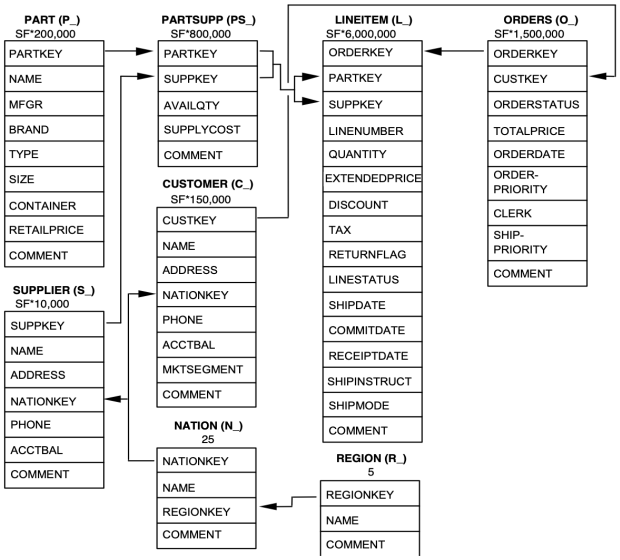

Figure 1 : TPC-H schema.

Queries: The benchmark includes 22 predefined SQL queries, each with varying degrees of complexity. These queries cover a spectrum of analytical tasks, including aggregation, filtering, and joining multiple tables. They mimic common business operations like sales reporting, inventory analysis, and market segmentation.

Performance Measurement: TPC-H measures database performance through several metrics. The most prominent metric is the query execution time, which quantifies the time taken to execute each of the 22 queries. Additionally, the benchmark calculates a throughput metric to gauge the database's overall processing capability.

Data Streaming Methodology:

Implementing the Data Streaming model is done in various phases. The first phase is to download the TPC-H tool kit from the official TPC website [3]. After downloading the tool kit, unzip it. To generate TPC-H compliant datasets, we must use the dbgen tool.Compile the dbgen tool by using command make -f makefile.suite. But before that modify makefile.suite by changing the following code:

CC = gcc (compiler., prefers using c compiler as makefile is written in c)
DATABASE = POSTGRES (name of the database that is used)
MACHINE = LINUX (operating system)
WORKLOAD = TPCH

It is a better practice to copy makefile.suite to another file that we create and perform changes in the new file. Now we can see data tables in xxx.tbl format in the dbgen folder. However, there is a bug in dbgen which generates an extra | at the end of each line [4]. We are also converting the xxx.tbl to .csv file. To fix it, run the following command [4]:

```
> csv_file="${tbl_file%.tbl}.csv"
> sed 's/|$//g' "$tbl_file" | awk -F '|' 'BEGIN {OFS=","} {print $0}' > "$csv_file".
```

After the dataset is available to use we need to set up the streaming model. Apache kafka is used for streaming the data. After Downloading Kafka we need to start the zookeeper. Apache Kafka relies on Apache ZooKeeper as a distributed coordination service.Zookeeper is used to track the status of cluster nodes in Kafka [6]. We need to run Apache ZooKeeper server with a specified configuration file.This file contains various settings and parameters that configure how the ZooKeeper server operates. The command to run the server :

>bin/zookeeper-server-start.shconfig/zookeeper.properties

Then we need to start kafka server using command

>bin/kafka-server-start.sh config/server.properties.

As we need some additional tools and features on top of Kafka we need to download the latest version.Confluent provides a suite of additional features and tools that extend the capabilities of Kafka. These include Confluent Control Center for monitoring and management, Confluent Schema Registry for schema management, and Kafka Connectors for easy integration with various data sources and sinks.

Then we need to create a topic in kafka. A "topic" is a fundamental concept that represents a logical channel or category to which messages are published by producers and from which messages are consumed by consumers. Topics serve as the primary mechanism for organizing and categorizing data within Kafka. We create a topic named tpch using the command: *sh kafka-topics --create --topic tpch --bootstrap-server localhost:9092--replication-factor 1 --partitions 1.* After creating the topic we need to list the topic to check if the created topic is present or not.

Streaming Data Ingestion :

To integrate the TPC-H dataset to kafka we need to create a producer file that needs to run on kafka server which takes data continuously and pushes it to kafka topic

Producer : Create a python producer file that takes datasets from tpc ,process the data and pushes the dataset to kafka topic. A critical aspect of this implementation is the Kafka producer configuration. This configuration includes essential parameters such as the Kafka broker's address (referred to as the bootstrap server) and buffering settings. These configurations play a pivotal role in optimizing data ingestion into Kafka.Import the necessary libraries (`os` and `Producer` from `confluent_kafka`). Import a custom data processing function `clean_and_process_data` from the `data_cleaning_processing module`. This function is responsible for cleaning and processing the data. Iterate through the list of TPC-H data files and, for each file, read each line of data. For each line of data, it calls the `clean_and_process_data` function to process and clean the data.The processed data is then sent as a Kafka message to the 'tpch' topic using `p.produce()`. It's important to note that the processed data is converted to a string (`'str(processed_data)'`) before sending it to Kafka.Regular calls to `p.poll(0)` are made to process delivery reports and ensure that messages are successfully published. After processing each CSV file, `'p.flush()'` is called to ensure that any pending messages are delivered.A final flush operation is performed at the end to ensure that all messages are delivered before the program exits.

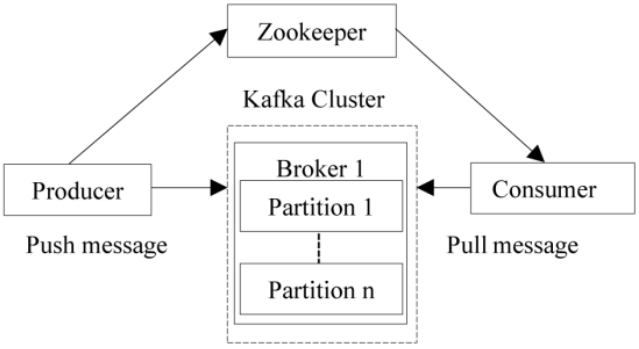

Figure 2 : Configuration of Apache Kafka architecture using a single broker(topic).

Data push Mechanism :

To integrate the streaming platform to postgres database we made changes in the confluent config folder. We created new files named ***Standalone postgres*** and ***sink quickstart postgres*** under confluent config**.** These files are responsible for having postgres connection url and database username and password.

Data is pushed from producer to kafka topic. Then it is the responsibility of the consumer to take this data from the kafka topic to consumer to postgresql database.

Consumer : Create a python consumer filr that takes data from kafka topic and push it to postgresql.Initialize a Kafka consumer (`'c'`) with specific configuration parameters, including the bootstrap server address, a consumer group identifier, and an auto-offset reset setting. The auto-offset reset is configured to start reading from the earliest messages in the topic.Initializes a connection to a PostgreSQL database (`'conn'`) with details such as the database name, user, password, host, and port. A cursor (`'cur'`) is created to execute SQL queries.The Kafka consumer is subscribed to the 'tpch' topic using `c.subscribe(['tpch']).` This ensures that it consumes messages from the specified Kafka topic.

We implemented conditional branches based on the length of `fields`. Depending on the length, it inserts data into different tables, such as 'region,'

'partsupp,' and 'nation'. Each `INSERT INTO` statement corresponds to the respective table and includes placeholders for data values.Data values are inserted into the tables using the `cur.execute()` method with the appropriate SQL query and a tuple of data values. The `commit()` method is called on the database connection to commit the transactions.The code always continues to loop, waiting for more messages to arrive in the Kafka topic and processing them accordingly.

Query Execution :

Understanding of TPThe dbgen file in TPC-H tool contains a folder named queries, it contains 22 predefined queries that needs to be runned in the database (postgres). These set of 22 predefined queries are designed to assess the performance of a database system in handling complex analytical and decision support queries [5].The TPC-H queries are intentionally complex and representative of real-world decision support queries. They involve various types of operations, including filtering, aggregation, joins, and subqueries, making them suitable for evaluating a database system's capabilities in handling sophisticated analytical workloads.Analyzing and understanding the TPC-H queries is a crucial step, as we need to study them in order to perform actions on the data tables

In our data streaming model we work on transformed TPC-H queries. We transformed these queries into postgres acceptable queries and modifications are made to these TPC-H queries while ensuring that the core functionality of each query remains unchanged.

```
TPC_H Query

-- $ID$
-- TPC-H/TPC-R Forecasting Revenue Change Query (Q6)
-- Functional Query Definition
-- Approved February 1998
:x
:o
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= date ':1'
        and l_shipdate < date ':1' + interval '1' year
        and l_discount between :2 - 0.01 and :2 + 0.01
        and l_quantity < :3;
:n -1

=========================================
Modified Query to suit Postgres DB
TPCHData=# SELECT CAST(SUM(l_extendedprice * l_discount) AS DEC(18,2)) AS revenue
                FROM lineitem
                WHERE l_shipdate >= DATE '1994-01-01' AND
                l_shipdate < DATE '1994-01-01' + INTERVAL '12 months' AND
                l_discount BETWEEN 0.06 - 0.01 AND
                 0.06 + 0.01 AND
                l_quantity < 24;
```

Figure 3 : Modified TPC-H query

Loading the entire dataset into the PostgreSQL database is a critical step since incomplete data uploads can result in missing outputs for these queries.The data loading process into the PostgreSQL database is time-consuming, primarily due to the substantial size of the tables.

Evaluation :

Having established a robust streaming setup, the next crucial step involves evaluating the efficiency of our data streaming system in comparison to traditional databases. To achieve this, we execute the TPCH queries within both setups and assess the execution time of these queries. This comparative analysis will provide valuable insights into the performance and effectiveness of our streaming infrastructure as opposed to conventional database systems. We can observe the execution time of both the setups. We made them run in the same environment. Even though we are running in the same environment there might be some additional metrics affecting the values. We tried to make this gap as slow as possible by providing every process in the system to perform query execution. It is evident that there is a significant disparity in the execution times between the two setups. Data streaming exhibits considerably shorter execution times compared to the traditional data processing approach.

TPC-H Query No	BP(ms)	DS(ms)
1	5,363.77	2.430
6	675.3	1.936
11	618	7.033
13	2423	143.455
16	1038.5	4.453
22	283	43.437

Table 1 : Execution time of some TPC-H queries (DS - Data Stream , BP- Batch processing)

We have calculated the bandwidth of our Kafka topic and message delivery from the consumer to PostgreSQL. During the execution, the bandwidth

for the setup measured at `819.77 bytes/second`. Notably, this bandwidth value is dynamically changing and continuously increasing, indicating that the consumer is processing data at an increasingly faster rate over time. These fluctuations in bandwidth may be influenced by various factors, including the volume of incoming data and network conditions.

Discussion :

The evaluation results unequivocally demonstrate the superior performance of data streaming compared to traditional batch data processing. In an era where data is generated incessantly, the continuous streaming of data proves to be more advantageous than storing and subsequently processing it in predefined batches. Consequently, numerous organizations have transitioned their platforms to embrace real-time data streaming. This shift is evident in various domains such as stock markets, weather data analysis, and many others, where the utilization of real-time data streaming has become a pivotal and transformative approach to gain immediate insights and drive timely decision-making.

However, Streaming data has its own disadvantages. Data Streaming tends to be more complex to design , develop and maintain compared to batch processing. Streaming Setup often requires more resources (CPU, memory) than batch processing , as they need to continuously process data as it arrives. This results in higher infrastructure costs. Continuous data streaming may lead to higher operational costs compared to batch processing, particularly when dealing with large volumes of data. Maintaining this flow consistently demands increased effort and a larger workforce.

Conclusion :

In today's data-driven landscape, the demand for real-time data processing has surged. Organizations seek timely insights and agile responses to dynamic data patterns, necessitating efficient real-time solutions. This study bridges the gap between traditional batch processing and real-time data processing by implementing a robust streaming setup. Data streaming, processing data as it's generated, contrasts with batch processing, introducing no delay between data arrival and analysis, ideal for real-time applications. Our goal is to seamlessly ingest TPCH benchmark data and process it in real-time. We explore integrating streaming technologies like Kafka with PostgreSQL, using the TPCH benchmark to evaluate performance. This research illuminates real-time data processing's advantages and limitations compared to batch processing. Readers gain insights into streaming data's transformative potential for traditional database workloads, empowering organizations to thrive in today's rapidly evolving digital landscape.

REFERENCES

[1] Barata, M., Bernardino, J., Furtado, P. (2015). An Overview of Decision Support Benchmarks: TPC-DS, TPC-H and SSB. In: Rocha, A., Correia, A., Costanzo, S., Reis, L. (eds) New Contributions in Information Systems and Technologies. Advances in Intelligent Systems and Computing, vol 353. Springer,

Cham. https://doi.org/10.1007/978-3-319-16486-1_61.

[2] Shahrivari, S. (2014, October 17). Beyond Batch Processing: Towards Real-Time and Streaming Big Data. Computers,3(4),117–129. https://doi.org/10.3390/computers3040117.

[3] https://www.tpc.org/tpc_documents_current_versions/curret_specifications5.asp

[4] https://gist.github.com/yunpengn/6220ffc1b69cee5c861d93754e759d08

[5] S. Ngamsuriyaroj and R. Pornpattana, "Performance Evaluation of TPC-H Queries on MySQL Cluster," 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops, Perth, WA, Australia, 2010, pp. 1035-1040, doi: 10.1109/WAINA.2010.167.

[6] M. T. Tun, D. E. Nyaung and M. P. Phyu, "Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming," 2019 International Conference on Advanced Information Technologies (ICAIT), Yangon, Myanmar, 2019, pp. 25-30, doi: 10.1109/AITC.2019.8920960.