SDS-Assignment1

IMT2020548 - Tejas Sharma

August 2023

1 Data loading

We create the event data database first by using the *CREATE DATABASE event_data* command. To use the database, use the *event_data* command.
Now you can create the table mappings and data to load the data.
To create and populate the mappings table use the following queries -

```
CREATE TABLE mappings(
a_id VARCHAR (50) PRIMARY KEY,
c_id bigint
);
COPY mappings FROM '/home/tejas/Desktop/Academics/Sem7/SDS/StreamingDataSystems/Assignment1/data/mappings.tsv';
```

To create the data table use the following queries -

```
CREATE TABLE original_data(
event_time BIGINT NOT NULL,
w_id integer,
rank integer,
iteration integer,
event_type VARCHAR NOT NULL,
a_id VARCHAR NOT NULL
);

COPY original_data FROM '/home/tejas/Desktop/Academics/Sem7/SDS/StreamingDataSystems/Assignment1/data/data0.tsv';
COPY original_data FROM '/home/tejas/Desktop/Academics/Sem7/SDS/StreamingDataSystems/Assignment1/data/data1.tsv';
```

Note that the given data do not obey the primary key constraints that were given earlier, and therefore I have dropped that constraint from the table.
Now I can consider this as the complete batch of the data.
There are 2 approachs to deal with this batch of data. First, we can wait for the whole data to come and then compute the query on 10 second non-overlapping windows on the complete data, or we can compute the query for every 10 second window in the data as it is coming in real time.

2 Part A

Here we perform the computation on the whole dataset together. This means that the system will first have to wait till the whole dataset arrives then computes the query on it. That means that there is an initial wait for the whole

dataset, then the loading time and then the time taken for computation.
The query that has to be computed is as follows -

```
query = """SELECT m.c_id as campaignId, floor(d.event_time/10000) as event_time, count(*) as count
    FROM main_data d
    INNER JOIN mappings m ON m.a_id = d.a_id
    WHERE d.event_type = 'click'
    GROUP BY m.c_id, event_time
    ORDER BY m.c_id, event_time asc"""
```

Time taken to wait for the whole dataset is - 654.999 seconds
Time taken to load the dataset into the system is - 10.96 seconds

```
event_data=# COPY original_data FROM '/home/tejas/Desktop/Academics/Sem7/SDS/StreamingDataSystems/Assignment1/data/data0.tsv';
COPY 1965000
Time: 5263.346 ms (00:05.263)
event_data=# COPY original_data FROM '/home/tejas/Desktop/Academics/Sem7/SDS/StreamingDataSystems/Assignment1/data/data1.tsv';
COPY 1965000
Time: 5697.329 ms (00:05.697)
```

The time taken for running the query is - 1.302 seconds

```
event_data=# SELECT m.c_id as campaignId, floor(d.event_time/10000) as eventTime, count(*) as count
FROM  original_data d INNER JOIN mappings m ON
m.a_id = d.a_id WHERE d.event_type = 'click'
GROUP BY m.c_id, eventTime
ORDER BY m.c_id, eventTime asc;
Time: 1302.498 ms (00:01.302)
```

Therefore the throughput of the system is - 3930000/667.261 = 5889.75

3 Part B

Here, to improve the throughput, we can take data every 10 seconds and perform the query and store the partial result. This ensures that we dont have to wait for the whole data to come and we can start performing the queries in advance.
We wait for the first 10 seconds data to arrive and then run the query. The output is stored in a file. While we do this, we are also waiting for the next 10 seconds data to come and start computation on it as soon as we get the whole 10 second data.
We can also create an index on the data based on the click event to ensure faster query response time. It is done as follows -

The algorithm for it looks like -

```
curr_timestamp = start_timestamp
start_time = time.time()
overall_processing_time = start_time
query_processing_time = 0
upper_time = start_time
loadProcess_time = start_time
# print('STARTING TIME: %f, UPPER_TIME: %f', start_time, upper_time)
while(curr_timestamp <= end_timestamp):
    start_time = time.time()
    if(start_time <= upper_time + 10):  continue
    upper_time += 10
    startQuery_time = time.time()
    query = """CREATE TABLE main_data_%s AS
        SELECT event_time, w_id, rank, iteration, event_type, a_id
        FROM original_data
        WHERE event_time >= %s AND event_time <= %s"""
    cursor.execute(query, (curr_timestamp, curr_timestamp, curr_timestamp+interval_milliseconds))
    query = """SELECT m.c_id as campaignId, %s as event_time, count(*) as count
        FROM main_data_%s d
        INNER JOIN mappings m ON m.a_id = d.a_id
        WHERE d.event_type = 'click'
        GROUP BY m.c_id
        ORDER BY m.c_id"""
    cursor.execute(query, (curr_timestamp, curr_timestamp))
    endProcess_time = time.time()
    query_processing_time += endProcess_time - startQuery_time
    loadProcess_time = time.time()
    curr_timestamp += interval_milliseconds
```

Using this, the time taken for the whole dataset to be processed = 660.31346

```
(env) tejas@tejas-G3-3500:~/Desktop/Academics/Sem7/SDS/StreamingDataSystems/Assignment1$ python wrapper.py
FINAL PROCESSING IS:  660.313458645551
```

The throughput using this approach is = 3930000/660.31346 = 5951.71
Now, we can try improving the query performance by creating an index on the event_type column like -

```
event_data=# CREATE INDEX click_index ON original_data(event_type);
CREATE INDEX
Time: 4910.474 ms (00:04.910)
```

The time taken for the whole dataset to be processed = 660.218945

```
(env) tejas@tejas-G3-3500:~/Desktop/Academics/Sem7/SDS/StreamingDataSystems/Assignment1$ python wrapper.py
FINAL PROCESSING IS:  660.2189447879791
```

The throughput using this approach = 3930000/660.218945 = 5952.57

4 Conclusion

Therefore using the second approach increases the throughput from 5889 to 5952.51. Thus it is better to work on small batches than to wait for the whole dataset. Because the latency to run each query is small, the changes are not

visible. To explain it further, the time taken to load and run each 10 second window query is small i.e less than 10 seconds. Thus, it doesn't make a huge difference. But if the time taken was more than 10 seconds then it would have made a big difference as in time greater than 20 seconds, we are able to compute one window and also load the next window data.