Weather data application using SQLstream

Axel Lewenhaupt, Richard Nysäter, Magnus Olsson, and Tobias Reinhammar

School of Computer Science and Communication (CSC)
Royal Institute of Technology KTH, Stockholm, Sweden

Abstract As the amount of data we generate and process is increasing at a dramatic pace, the need to efficiently handle this data is now more important than ever. Traditional database management systems are good at storing and organizing data, but struggle when faced with the challenge of processing an overwhelming amount of incoming data in real-time. In attempting to solve this problem, a new breed of databases and data stream management systems have been developed. This project explores the use of one such system, SQLstream to process large amounts of weather data in simulated real-time for a web application. This project was a proof of concept to test whether or not it is suitable to apply a streaming database for this purpose. Several problems were encountered along the way and had to be overcome but eventually a working system was produced. As such the concept was deemed feasible, although there are many issues which need to be addressed.

1 Introduction

1.1 Background

Traditional database management systems (DBMS) are used in many applications today in order to handle and store data. DBMSs use a "process-after-store" model, where data is first stored and indexed before being processed [1].

Typically, this means that the data is first sent to the DBMS, which indexes and stores the data. An application which wants to query data must then send a request to the DBMS, which will then process the data and return the result. This is problematic when a large amount of streaming data is used for input, as high latency and overhead will slow down the queries [2].

Data stream management systems (DSMS), while being quite similar to DBMS, provides additional functionality by allowing queries to search the data being streamed directly from a source, without storing and indexing the data beforehand [1].

1.2 SQLStream

There are quite a few different DSMSs with different features and uses. For our application the SQLstream DSMS is used. SQLstream provides support for standards-compliant SQL for querying live data streams and is very fast, having exceeded over a million 1 Kbyte records per second per server [3].

2 Application

For this project a website was created, providing weather reports using a streaming database with continuously updating queries. A continuous query is run against the streaming data and the results received are updated as new data is committed. The size of the stream is controlled by a sliding window, which determines the timeframe for the data which is queried.

One problem with a DSMS project is acquiring enough data for the streaming database to be challenged. For this project the weather data provided by the National Oceanic and Atmospheric Administration (NOAA) is used to simulate real-time readings from a large amount of weather stations.

3 Data set

The data set was acquired from the NOAA and consists of daily weather reading from more than 40,000 weather stations distributed across all continents. The dataset is the world's largest collection of daily climatological data. The 24GB data set consists of a total of 1.4 billion data values includes values for maximum and minimum temperatures, precipitation totals, and observations for snowfall and snow depth. Some station records extend back to the 19th century[4].

4 Architecture

The backend consists of two layers on top of a streaming database server running SQLstream. The first layer directly on top of the database is a basic http server implemented in java which forwards the queries and responds with the results in JSON format. The frontend consists of a web page using AngularJS and JQuery to present the query results. Below is an example of how a query is received from the front end website and processed step by step in the application to return the result.

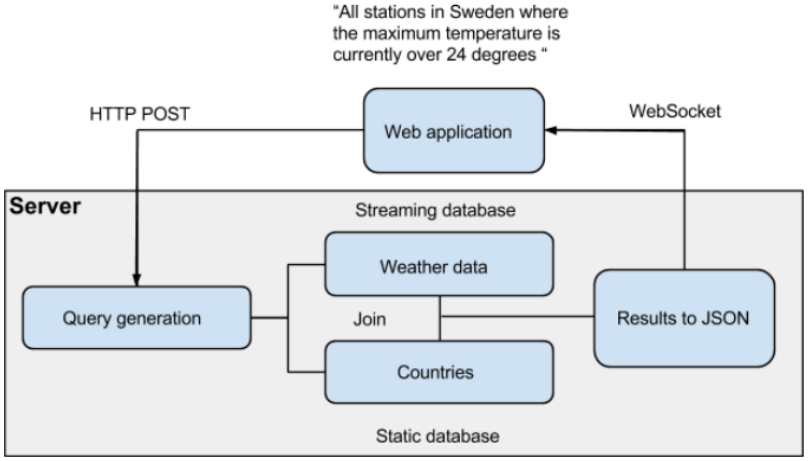


Figure 1: Depiction of the architecture

4.1 Database structure

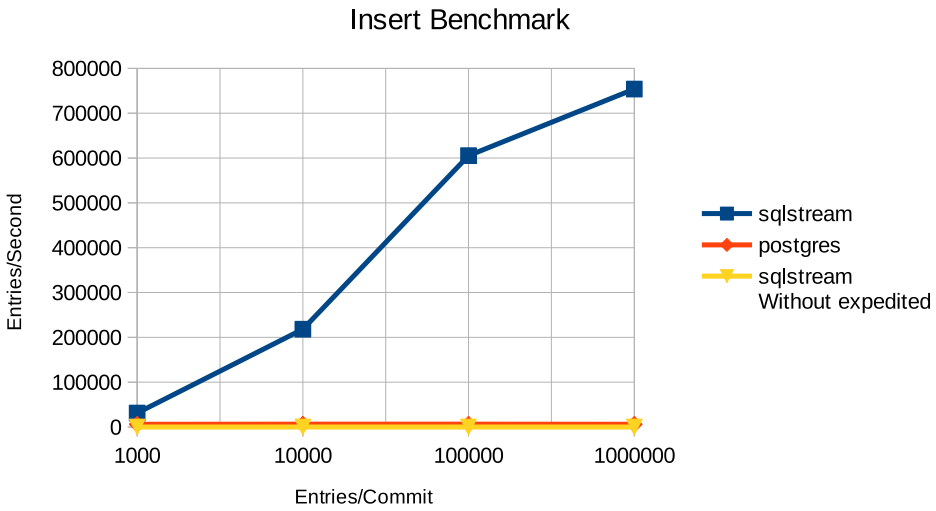
The application features two separate databases. One PostgreSQL database which stores the static information used by the application, such as weather station information and countries. The second part is an SQLstream streaming database which handles the flow of continuous weather data updates.

Table 1: Database Structure

Static	Static	Stream
Stations	**Countries**	**WeatherStream**
StationID	*CountryID*	*StationID*
Latitude	Name	DataType
Longitude		Data
Elevation		
Name		
CountryID		

5 Results

In this section a benchmark showing the performance difference between SQLstream and Postgres for inserting a large amount of records is presented. Furthermore, a link to a video demonstrating the application working in real time is available in the references[5].



The weather website was successfully created and works by sending a request for a query to the server, which responds by continuously sending back the requested data from the stream. The data includes the current value, as well as an average of the last 2 weeks.

6 Conclusions

The results show that it is possible to create an effective system for weather data using a streaming database and that a DSMS outperforms a DBMS in this context. For our application to be of real use it would need to receive a huge amount of data continuously from the weather stations. In practice the API supplying the weather data is more likely to be a bottleneck than the database.

Regarding the DSMS used, SQLstream, there are some mixed feelings. While the system had good performance, there was a lot of issues with getting the system up and running. The s-Studio platform crashed frequently which made progress slow and tedious. Furthermore, SQLstream did not support joins between streams and static tables, which caused a lot of issues as we required joins between the weather stream and the static tables containing station and country data. This forced us to work around it by putting the static data into a normal PostgreSQL database and performing the join through an obscure extension between the external databases. What should have been a simple operation required a huge amount of work. This gave the impression that SQLstream is more suited to work in conjunction with other systems as a part of a larger network rather than as a single database solution.

Another problem was that the data used in this project was historical data, which was used to simulate continuous updates, rather than true real-time data. This caused some format and inconsistancy issues both when trying to parse the data and when accounting for missing data.

References

All links worked as of 2015-05-27.

1. Stonebraker, M.,Cetintemel, U., Zdonik, S.
   The 8 Requirements of Real-Time Stream Processing,
   http://www.sigmod.org/publications/sigmod-record/0512/p42-article-stonebraker.pdf
2. Abadi, D., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, M., Zdonik, S.
   Aurora: a new model and architecture for data stream management,
   http://www.cs.brandeis.edu/~mfc/papers/vldb095.pdf
3. SQLstream documentation,
   Streaming Big Data Performance Benchmark for Real-time Log Analytics in an Industry Environment,
   http://www.sqlstream.com/wp-content/uploads/2013/03/SQLstream_Big_Data_Performance.pdf
4. NOAA - National Centers for Environmental Information,
   GHCN (Global Historical Climatology Network) - Daily Documentation,
   http://www1.ncdc.noaa.gov/pub/data/cdo/documentation/GHCND_documentation.pdf
5. Lewenhaupt, A.
   WeatherStream Demo,
   https://www.youtube.com/watch?v=9Ovp3zj0dn4