Seminar Databasesystems

By
Samuel Kurath

University of Applied Science, HSR

Autumn 2016

Advisor:
Prof. Keller Stefan

Seminar Databasesystems

Abstract

This paper is splitted in three parts, it begins with an overview of streams and their difficulties. Followed by a part about Apache Storm a distributed stream processing framework. And finally a concrete implementation based on a given problem with Storm. The goal of the implementation is to do some queries and analysis on the minutely updated Augmented Diffs of OpenStreetMap.

Contents

0 Introduction

Durring the MSE master degree the students have to absolvate two seminars. The goal of these is to elaborate a theme on your own, discuse the result in group and write a paper about the topic.

The Databasesystems Seminar does a focus on streams and their processing. Stream processing is a strong growing subject in reference to the huge amount of data we are exposed and produce nowadays.

A big force in generating this data is the rapidly increasing amount of Internet of Things IoT sensors, the willingness of the people to populate a lot of personal information on social media platforms and also the expanding interest in data collection of companies.

With this amount of data new problems in collecting, processing, storing etc. appear and thus new solutions and technical tools to solve them appear too.

The man who is swimming against the stream knows the strength of it.

Woodrow Wilson

1 Stream Processing

Steams are older then computers so it is not a big surprice that streams and the processing of them isn't a absolutly new topic in the computer science world. Historically there are techniques like **logging** or in the domain driven development area there is **event sourcing**, which are very similar to streaming processing. But they have not always been such an important deal like they is today with the with the unbelievable amount of data. Antecedent to handle this streams was an event driven way and do analytics after storing the data.

Frozen yogurt

Let me explain the *old* event driven way with a small example.

Imagine a factory which produces frozen yogurt in different flavors. They weigh and register every cup of yogurt at the end of the assembly line.

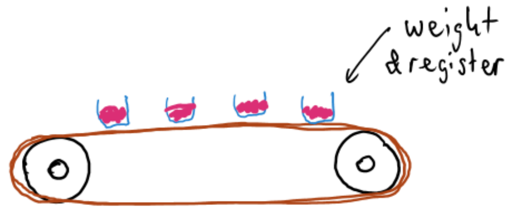

Figure 1.1: frozen yogurt assembly line

Common architecture

The factory has sensors which weight the cups and this weights are sent to a server. The server handles the request and stores the frozen yogurt with his weight in the database. for analytic tasks there is a web application. If you are now interested in the total amount of produced cups. You can simply open the browser go to the analytic page. This starts a request to the server and the server will call the database with a quey like "select count(*) from frozen_yogurt". After the quey is executed you will get the result from the server and have the aggregated nummer on your screen.

(This is more or less a default example of a Three-tier architecture)

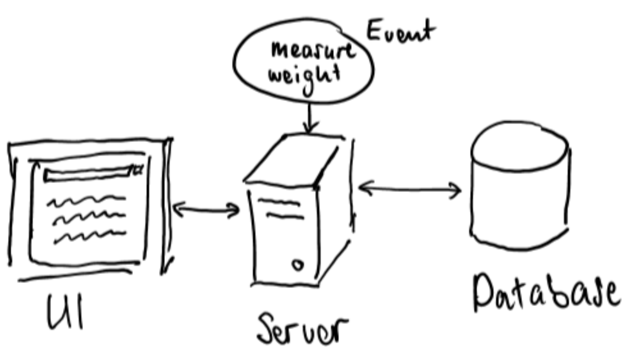

Figure 1.2: Three-tier architecture

Now the owner of the factory does a very good business and they are able to expand the production. And with this expansion they also add a lot more of sensors to the assembly line like, a temperature sensor, optical recognition to check if the cups are always full and a lot more.

The requirements of the system are also updated the owner want to have statistic about the production all the time and want's immediately notifications if for example the temperature is to high.

These new requirements leads to new challenges in the architecutre of the system and are hard to implement with the current state and the enormus data produced by the all sensors.

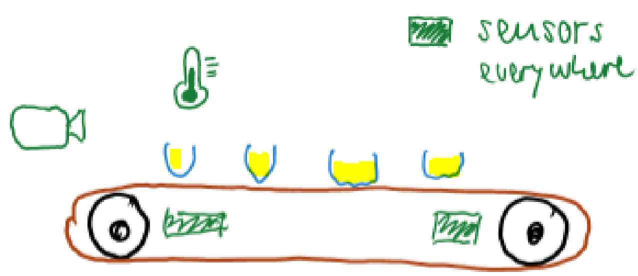

Figure 1.3: sensors everywhere

Streaming architecture

A good way to handle this new requirements is to continuous aggregate and filter the stream of data before it is stored in the database. For this purpose there has grown up a lot of new techniques and frameworks during the last few years.

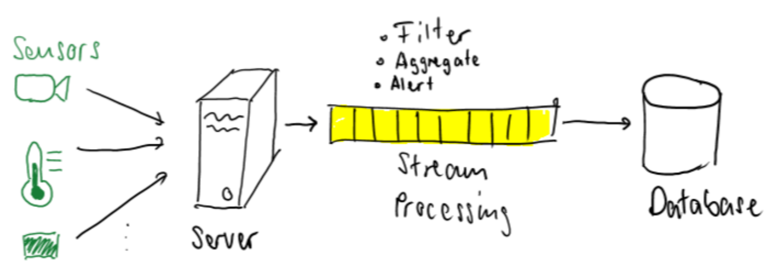

Figure 1.4: streaming architecture

Conclusion

The new streaming architecture of the frozen yogurt factory has got a lot of advantages but also has to handle some big difficulties. Now it is possible to get very fast access to production statistics, deal with the huge amount of data and throw alerts in real time. It's also helpfull in the developers perspective, because you can add several consumer to the stream. Normaly this stream is immutable if you use a technologie like kafka, which has the advantage that there aren't concurrency problems anymore. Thus the requirements in such real time stream processing systems are ambitious and we will face them in the next section.

Eight Rules For Stream Procesing

In the paper "The 8 Requirements of Real-Time Stream Processing" Michael Stonebraker, Uur etintemel and Stan Zdonik bring their thoughts about real-time stream processing together and write it down. It faces the fundamental ideas of stream processing and sums them up in eight rules which every allows us to measure and compare different technologies. Thus leads me to take a closer look at these rules and add my reflection to them.

Rule 1: Keep the Data Moving

The first requirement for a real-time stream processing system is to process messages in-stream, without any requirement to store them to perform any operation or sequence of operations. Ideally the system should also use an active (i.e., non-polling) processing model.

Rule 2: Query using SQL on Streams (StreamSQL)

The second requirement is to support a high-level StreamSQL language with built-in extensible stream- oriented primitives and operators.

Rule 3: Handle Stream Imperfections (Delayed, Missing and Out-of-Order Data)

The third requirement is to have built-in mechanisms to provide resiliency against stream imperfections, including missing and out-of-order data, which are commonly present in real-world data streams.

Rule 4: Generate Predictable Outcomes

The fourth requirement is that a stream processing engine must guarantee predictable and repeatable outcomes.

Rule 5: Integrate Stored and Streaming Data

The fifth requirement is to have the capability to efficiently store, access, and modify state information, and combine it with live streaming data. For seamless integration, *the system should use a uniform language when dealing with either type of data.*

Rule 6: Guarantee Data Safety and Availability

The sixth requirement is to ensure that the applications are up and available, and the integrity of the data maintained at all times, despite failures.

Rule 7: Partition and Scale Applications Automatically

The seventh requirement is to have the capability to distribute processing across multiple processors and machines to achieve incremental scalability. Ideally, the distribution should be automatic and transparent.

Rule 8: Process and Respond Instantaneously

The eighth requirement is that a stream processing system must have a highly-optimized, minimal-overhead execution engine to deliver real-time response for high-volume applications.

There are some things you learn best in calm,
and some in storm.

Willa Cather

2
Apache Storm

Apache Storm is a reliable, distributed and fault-tolerant system for stream processing. The beginnings of the project were at Backtype (later bought by Twitter) and created by Nathan Marz. He open sourced Storm on September the 19th in 2011. The project rapidly got a big development coummunity and on September the 18, 2013 Nathan moved Storme to Apache Incubator.

Storm works with different types of components which are responsible for clear defined task. This components are bundled and managed in a so called **Topology**. The entrypoint and the stream input is handled by a **Spout**, the spout passes to **Bolts**. Bolts are responsible for the main data processing and persists the data. They can be chained or parallelised in a way that fits best for your current problem.

Figure 2.1: Storm

Topology

In generell Storm passes tuples between the different components. To organize this tuples there are topologies.

Grouping

The Topology defines the grouping, this means how streams are consumed by the bolts and how they consume them. There are four kinds of grouping *Shuffle Grouping*, *Fields Grouping*, *All Grouping* and *Custom Grouping*.

Shuffle Grouping

Shuffle Grouping takes a single entry from the source and sends each tuple to a randomly choosen bolt, which is listening to this kind of tuples. It also garanties that each consumer get's the same number of tuples.

Fields Grouping

With Fields Grouping it is possible to send toples to spezified bolts based on the fields of the tuple. It takes care that the same combination of fields is always sent to the same bolt.

All Grouping

All Grouping sends every tuple to all the bolts. This is helpful for tasks like signals or use different filters for alerting systems.

Custom Grouping

It is possible to build your on grouping based on the stream. This is very helpful if you have to make fine granular desicions about which bolt has to handle which tuple.

Spout

The entry point of each topology are the so called spouts. The spouts are responsible for their reliability, thus you have to take care to implement them in a fault-tolerant way. This means that a spout must have the ability to deal with unprocessable messages from the stream.

To handle the reliability at the spout you can add an ID to every message. If a message is correctly processed by all the target bolts the *ack* method of the spout is called. But if there were troubles or the timeout is reached the *fail* method will be triggered.

A good idea is about ten percent and implementation and hard work, and luck is 90 percent.

Guy Kawasaki

3 Implementation

This chapter is about the implementation with Apache Storm to solve the defined queries. The idea of the test scenario is to use ”OSM Augmented Diffs” as a stream. This diffs extend the ordinary minutely diffs of OSM with more information. The result contains all the nodes, ways and relations changed durring the time periode.

Queries

The following list is about the queries which has to be done on the Augmented Diff stream.

A) Leader board of top 10 OSM active users

B) Leader board of top 10 OSM objects added

C) Node objects with suspicious keys and values

D) Way objects with only user tag ”area=yes” without other user tags

A) Leader board of top 10 OSM active users

It's about users who created or updated any OSM nodes all around the world. The users are identified with by the tags "uid" and "user".

B) Leader board of top 10 OSM objects added

Count all the created or updated nodes grouped by a combination of key and value. An example for this is "amenity=bench".

C) Node objects with suspicious keys and values

Detection of vandalism and unwanted bad actions in OpenStreetMap is a very difficult topic. Thus we focus on "Quality Assurance Monitoring" by filtering for suspicious tags of node or way elements. As an example for this issues are ";" in Tags like "crossing=island;uncontrolled".

D) Way objects with only user tag "area=yes" without other user tags

The point D) also deals with the vandalism problems.

4
Conclusion