CASE STUDY

Energy meets Big Data

Background

Our customer - a company specializing in the analysis of energy usage data - was encountering performance and data integrity issues with their data processing platform. When we joined the project, the processing capacity was becoming insufficient and scaling the platform was not an option.

Since the client expected significantly more data within a short period, they asked SoftwareMill to build a new system with scalability in mind. Apart from pulling data from external systems, the new solution needed to be capable of consuming data pushed from a number of external devices, with an expected throughput of 2000 messages per second.

Performance issues

There were two areas where the performance needed to be improved.

The first one was an external API from which the energy consumption data was fetched. The API endpoint was rather a slow one, with response times around tens of seconds. Combined with a single-threaded data fetching process, it resulted in a situation where it was impossible to fetch the consumptions from a single day within 24 hours, which led to an accumulating delay.

The other area was the data cleaning part, whose implementation was far from optimal. Although the algorithms themselves were fairly simple, there was a significant number of unnecessary SQL queries that slowed down the entire process. Moreover, the legacy implementation was in PHP, which resulted in single-threaded execution, although most of the calculations could well have been executed in parallel.

hello@softwaremill.com

Our solution

The general architecture of the solution we created is presented in Figure 1. Following are more detailed descriptions of some specific areas of the system.

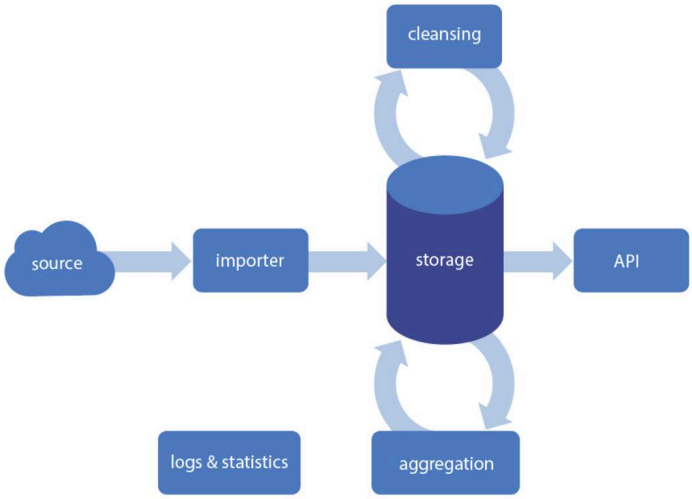

Figure 1. Architecture overview

Performance and data integrity

In order to achieve maximum efficiency when fetching data from the external API, we introduced a streaming approach, which let us parallelize the fetching process in a number of places. Firstly, we were able to make a controlled number of concurrent connections to the external server. Secondly, we could perform some intermediate computations and prepare batches of data to be inserted into the database in an asynchronous manner.

By using the Akka Streams library to implement the processing pipeline, we could focus on developing the actual processing logic in the form of small, independent building blocks. Meanwhile, the responsibility for planning and optimizing the actual execution of the processing graph was handled by Akka Streams automatically. To ensure that the imported data is both complete and not duplicated, we implemented a number of integrity checks to preserve uniqueness and re-fetch any missing records.

Scalability

Having in mind the constantly growing volume of data and the high throughput expected for the pushed data, we decided to choose Apache Cassandra as the database for two main reasons:

- high performance of inserts
- out-of-the-box scalability on commodity hardware

Post-processing large datasets

The collected data was exposed through a REST API so that the client could access it from their own systems. While exposing the raw data was a no-brainer, it was a challenge to efficiently compute some less trivial aggregations so that the could be exposed as well. For such resource-heavy computations we have successfully used Apache Spark - a platform that abstracts away the distributed processing of large datasets.

Event-based metrics

The client wanted to be able to track some areas of the data import process and use the tracking data to compute a number of metrics. Since we didn't want the tracking mechanism to affect the performance of the critical data processing components, we chose an event-based approach with the data processor only asynchronously emitting events (at almost no cost) and a separate infrastructure handling those events and computing the metrics.

The monitoring infrastructure was build using Riemann as the event aggregation layer and InfluxDB as the time series datastore. The metrics stored in the datastore could then be easily visualized in Grafana, which seamlessly connects to InfluxDB.

Summary

By introducing Akka Streams to implement a streaming-based approach to fetching data from external systems, and using Cassandra as a high-performant and easily-scalable database, we were able to rebuild our client's data processing platform to perform in a timely manner and be ready to handle the constantly growing volumes of data.

Contact

We take your mind off software development. Just drop us a line: hello@softwaremill.com - we'll get back to you!