Projectdocumentation for Open Environemental Data Cloud Prototyping Project

Nico Tasche, ...

Technische Universitaet Berlin, Strae des 17. Juni 135, 10623 Berlin, Germany,
nico.tasche@campus.tu-berlin.de

Abstract. Keywords: Open Data, Environemental Data, Big Data

1 Introduction

Very usefull text!

2 Database

2.1 Overview

2.2 Requirements

Requirements

- scale-able in the range of petabyte in size
- hundred of thousands of requests per minute
- high availablility
- partition tolerance
- fast to handle timeseries
- fast to handle geolocation data
- immediate consistency is NOT necessary

Requirements

- Open source or at the very least an open license is required
- must be well documented
- must be managable regarding administration and learning afford

2.3 Survey of Existing Solutions

2.4 Evaluation Criteria and Decision-making Process

The process of deciding what database architecture to use we started with our requirements.

Espacialy the last point of our secondary requiremnts had to be taken into account, because we had no real database expert in our team, so we first considert database-system we allready knew. Our approach was to check whether those databases fulfill our requirements first.

Any relational database as main datastorage has been quickly disreagarded, cause of the bad fairly scaling behavir with the amount of data we have to handly.

2.5 Implementation Details

Intro to Elasticsearch Elasticsearch is an opensource Lucene based search engine. It is under active development, with an extensive documentation.

Architecture Each index can be sharded and each shard can have multiple indieces.

TODO: Picture of architecture

To better distribute search requests, the workload is divided among all shards belonging to an index. Because that would not scale very well and would have no partition tolerance, each shard has a configurable number of replicas. A new search request is send to on replica of each shard.

Data model We decided to have an data model which is data-source-centric with the extra posibility to partition the data over time. That means, each data source gets it own index with its own timeframe and its own adjusted datastructure. All our data sources save a few basic data point with each element stored in the database, in particular are those:

- timestamp: when has the datapoint been recorded
- location: where has the datapoint been recorded

Those are acutally the only information we need to store, besides the individual measurements. We do acutally store some more information, but regarding the common usecases for searches those two datapoints are enough for environemental data. Please refer to the full data-model in the appedix for more information.

This data-model has multiple advantages:

- it keeps the data provenance
- it allows us to adjust the server infrastructe based on the data source
- it scales indefinitely
- index size is deterministic, cause of time based partitioning

So why does it scale so good? When importing data from one source, I process and store the data points in one index. This index is not just limited to the data

source, it is also limited to the time, e.g. 2016. That means, when 2016 is finished with importing data, the index is done and can be closed up, no one needs to care about it anymore. After the index is done, it might even be transfered to another Elasticsearch node with different hardware. That would be usefull, for example, when the average density of the smurf population is beeing stored. The index can be transferd to a less powerfull hardware with fewer CPU cores and spinning harddrives and even fewer replicas, because this information is probably hardly requested.

Query optimization A search request in send to all indieces, that means:

Why do we need query optimization? For that I'm going to give a small small example to consider:

1. we import multiple sources, with multiple messurements: source1(airtemperature, watertemperatur) 1980-2017, source2(airtemperature) 1983-1990, source3(uv-index) 2009-2017
2. each source is partitioned by year and source 1 is partitioned by month for all data after 2015.
3. every index is naivly sharded over 3 nodes

Let's make a simple search request: give me all uv values data from 2015 till 2017 and aggregate an everage over the month.
Because the user does not now anything about the internal database architecture (at least he should not) he requests the temperature and the timeframe.

Worst case:

$$source1 = 35years + (2yearsx12month)x3shards \tag{1}$$

$$source1 = 177shards \tag{2}$$

$$source2 = 17yearsx3shars \tag{3}$$

$$source2 = 51shards \tag{4}$$

$$source3 = 8yearsx3shars \tag{5}$$

$$source3 = 8yearsx3shars \tag{6}$$

$$source1 + source2 + source3 = 252shards \tag{7}$$

So in worst case each shard has its own node(very unlikely), the search request has to be send to 252 nodes/computers.

First optimization, Limit the time

With a naive approach by checking the common time part of the request 2016-2017 and limit the indieces search with the following pattern:

$$indexsearch : *-201* \tag{8}$$

$$source1 = 5years + (2yearsx12month)x3shards \tag{9}$$

$$source1 = 87shards \tag{10}$$

$$source2 = 0yearsx3shars \tag{11}$$

$$source2 = 0shards \tag{12}$$

$$source3 = 3yearsx3shars \tag{13}$$

$$source3 = 9shards \tag{14}$$

$$source1 + source2 + source3 = 96shards \tag{15}$$

We allready reduced the number of shards we need to address by 61%.

TODO:!!!
With a little more sophisticated timelimitation algorithm, we could acctually do more and search just those two years: If we store in a seperate database, which data source and therefore indiece actually holds the requested data we can do even much more:

Now we are at 68% reduction.

Second optimization, Limit to indieces which contain the right data

If we store in a seperate database, which data source and therefore indiece actually holds the requested data we can do even much more:

TODO

By using those two optimitzations, we were able to reduce the number of requeseted shard to 6, which means a total reduction of 96.2%.

This was just a naive example. In reality the reduction should even be much higher, with a growing number of data sources. Let's say we have allready 100 data sources and we can limit a request to just two of those for example because the requested messuremnt it provided just by those two, the saving of network traffic and workload would be immense.

2.6 Critical Analysis/Limitations

Joins One mayor drawback of elasticsearch is the missing possibility of server side join, the way they are known by SQL based database-system. This means, any kind of join operation has to be done either on a seperate server, like our api instance, or on the application side. This is actually something we were not really aware of for a long time.

References

[1982] Example: Nonlinear oscillations and boundary-value problems for Hamiltonian systems. Arch. Rat. Mech. Anal. 78, 315–333 (1982)

[1982] Clarke, F., Ekeland, I.: Nonlinear oscillations and boundary-value problems for Hamiltonian systems. Arch. Rat. Mech. Anal. 78, 315–333 (1982)