Grafana-InfluxDB Research Report

Written by: Yitan Ze, Jiahao Xu

Faculty Advisor: Professor Chen Li

Abstract:

The goal of this project was to find an optimization method for data loading on Grafana, a time series data visualization tool. We first analyze the performance of Grafana when it encounters a large amount of data. After finding the bottleneck of the data limit, we try to find a solution that helps accelerate the visualization speed.

Problem:

When we load a large amount of InfluxDB data (i.e. 2.5 million, 165MB), Grafana will be stuck and cannot show data at all. After several tests, the slow but manageable amount of data that Grafana could handle is about 40MB (650,000 data points). Therefore, we want to do some optimization so that Grafana can smoothly display datasets of any size in an acceptable time.

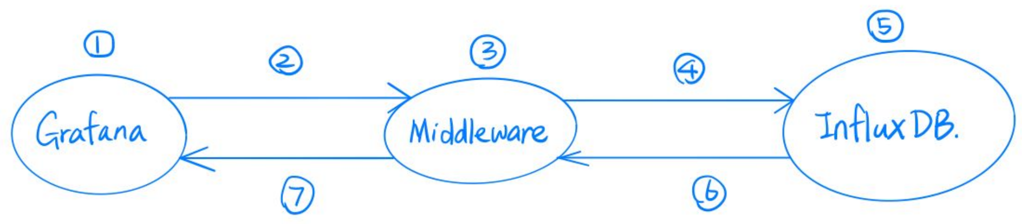
Getting to know the system:

- Grafana interacts with InfluxDB using RESTFul API.
- InfluxDB sends a JSON style, gzip encoded file back to Grafana
- The time spent on the internet transfer is around 3s (a query for 600,000), while rendering took almost 10s.

Solution:

Implement a middleware:

We want to implement a middleware between Grafana and InfluxDB that can optimize the query, so that the frontend can display a large query result with intelligently sampled data.



For testing purpose, we implemented four versions of middleware:

1. Simple Forwarding:
    a. Get query from Grafana, forward it to DB.
    b. Retrieve the result from DB, forward it to Grafana.
2. Same query, but only send 2000 points to frontend.
    a. Get query from Grafana, forward it to DB.
    b. Retrieve the result from DB, forward only 2000 points to Grafana.
3. Optimized query using random sampling.
    a. Get query from Grafana, optimize the query so that it will only grab 2000 points from DB using random sampling. Send the optimized query to DB.
    b. Retrieve the result from DB, forward it to Grafana.

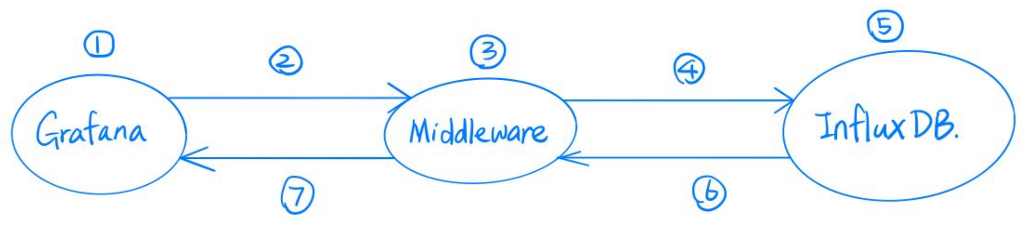
Note： We also tried optimizing query using GROUP BY, i.e. split the data into 2000 intervals and get the mean of each interval. Then we realized that taking the average may omit the true pattern of the data(i.e. when data is uniformly sampled in a range), so we chose random sampling, since it uses reservoir sampling and thus gives a more precise data pattern.

Basic of middleware:

The middleware acts like InfluxDB, working on a certain port. It can handle all the query Grafana made for Influxdb. Besides, the middleware can also handle functions that Influxdb does not support. For example, Grafana sent a *query_with_special_request.* The middleware can parse, take out the special request, which Influxdb does not support, and solve the special request. This feature makes the middleware extendable.

Testing our middlewares:

In order to know the performance of three middlewares and which part slows down the Grafana visualization process, we try to visualize 2.6 million data on Grafana using our three middlewares, and inspected the time it takes in each part.



Note: To avoid caching, we restart our database in every query experiment. We use Chrome to inspect the frontend rendering time.

DATA SIZE: 2,595,496	***4+5+6*** **InfluxDB Query time(sec)**	Count query time	***6*** Approximate Transfer time	***2+3+4+5+6+7*** **Total query time in the middleware(sec)**	***1*** **Grafana time consumption (sec)**
Simple forwarding	9.68	0.57	9.11	27.31	CRASHED
Same query, send 2000 pts to Grafana	9.55	0.539999995	8.78	12.06	0.29
Optimized query:(SAMPLE)	2.42	1.26	0.5	4.00	.245
Optimized query(GROUP BY)	1.31	1.28		2.81	.362

Observation:
The central bottleneck of the Grafana' slow visualization is the frontend rendering limit. Frontend Grafana has a limitation on the amount of data it can display. When the data size is very large (i.e 2.5 million), the frontend would crash. Query time and transfer time may also get slow when data is large.

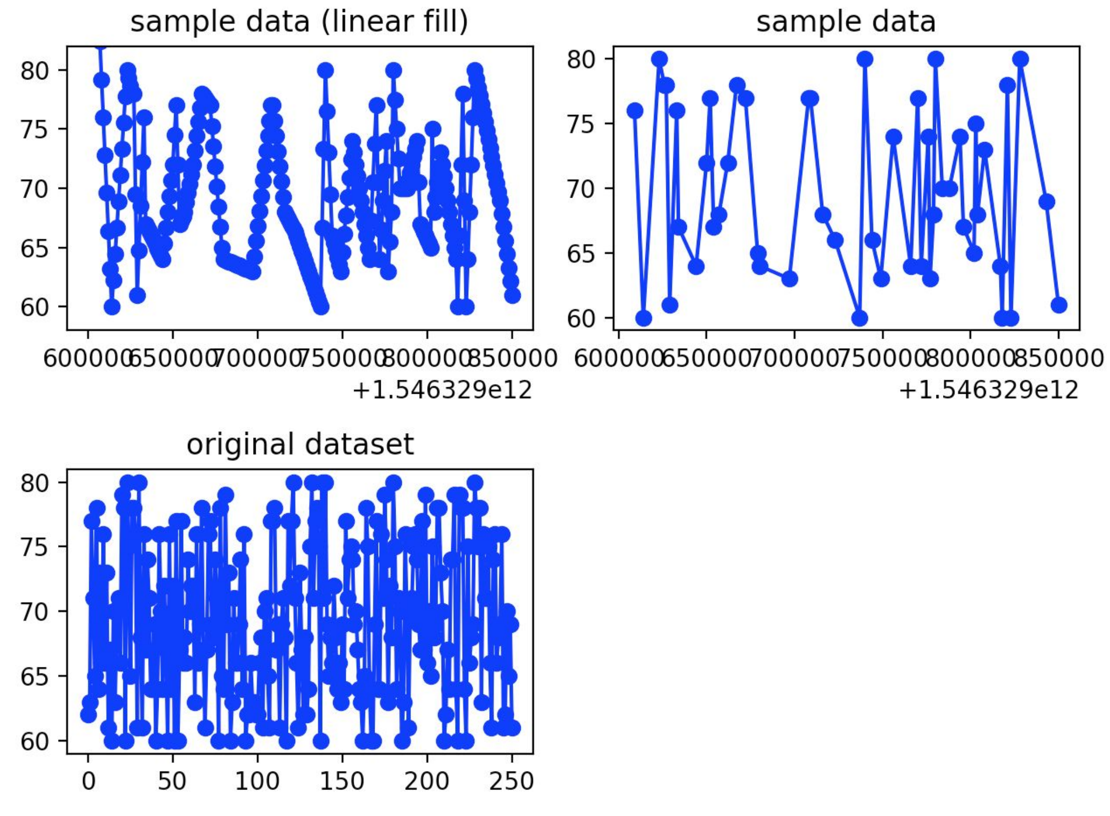
Disadvantage:
Although using random sampling in the middleware will improve the frontend visualization speed, the sampled pattern may not be close to the original data pattern given the fixed sampling

ratio. For instance, if we fix the number of sample size(i.e. 2000), for dataset with no obvious pattern, this sampled data may not show the original data pattern very well, thus, an optimization is needed.

Intelligently decide a sample ratio:

Therefore, we want to make our middleware be able to intelligently decide a sample ratio. Below is our rough idea of implementation:

- Try different sampling ratios, for each ratio:
    Measure the similarity for each of the sampled dataset.
- Decide a min-similarity requirement.
- Find the lowest sampling ratio that meets the similarity requirement



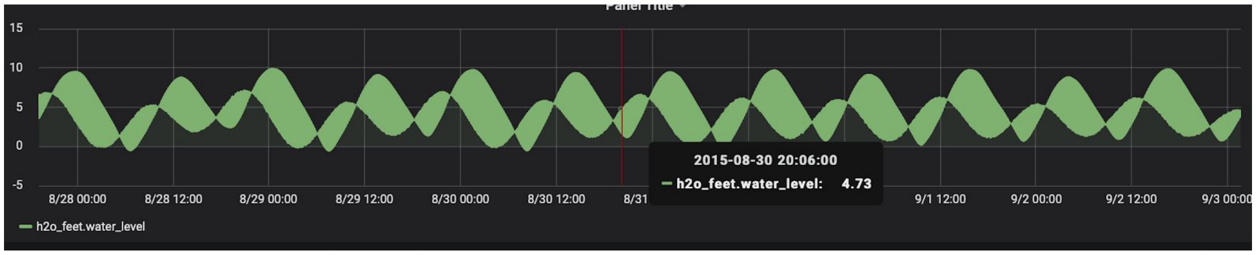
To measure the similarity between sampled data and the original data, we first fit the sampled data using a linear model (since it's how Grafana displays the data and we want to make the two datasets have similar amount of data), then compute the distance of two datasets using the distance function provided by the SAX model. In the example above, the top left diagram shows the linear fitted data of the top right sampled data, and we want to compute the distance of the sample data(after linear fit) with the original data.

Symbolic Aggregate Approximation(SAX) is a Novel Symbolic Representation of Time Series. It allows a time series of arbitrary length n to be reduced to a string of arbitrary length w, and it provides a convenient distance measure function. For a detailed explanation, please refer to section 3.3 in *https://cs.gmu.edu/~jessica/SAX_DAMI_preprint.pdf*.

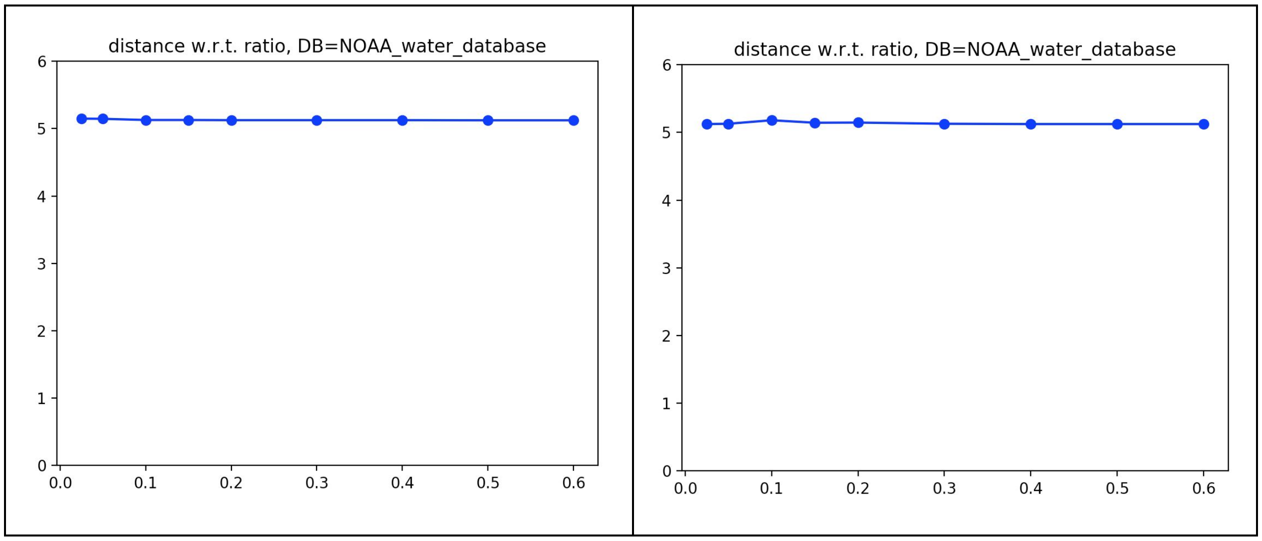
Experiments:

After finishing the implementation, we did experiments on three different kinds of datasets/tables.

1. On NOAA: h2o_feet table: obvious pattern



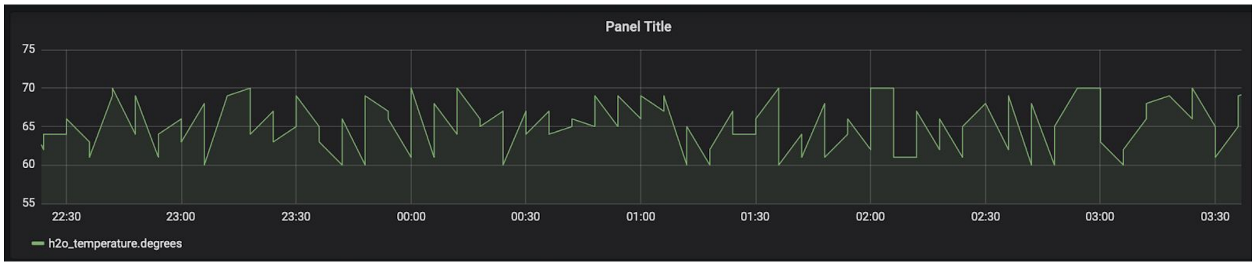
original data size: 4316
alphabet size avg: 11

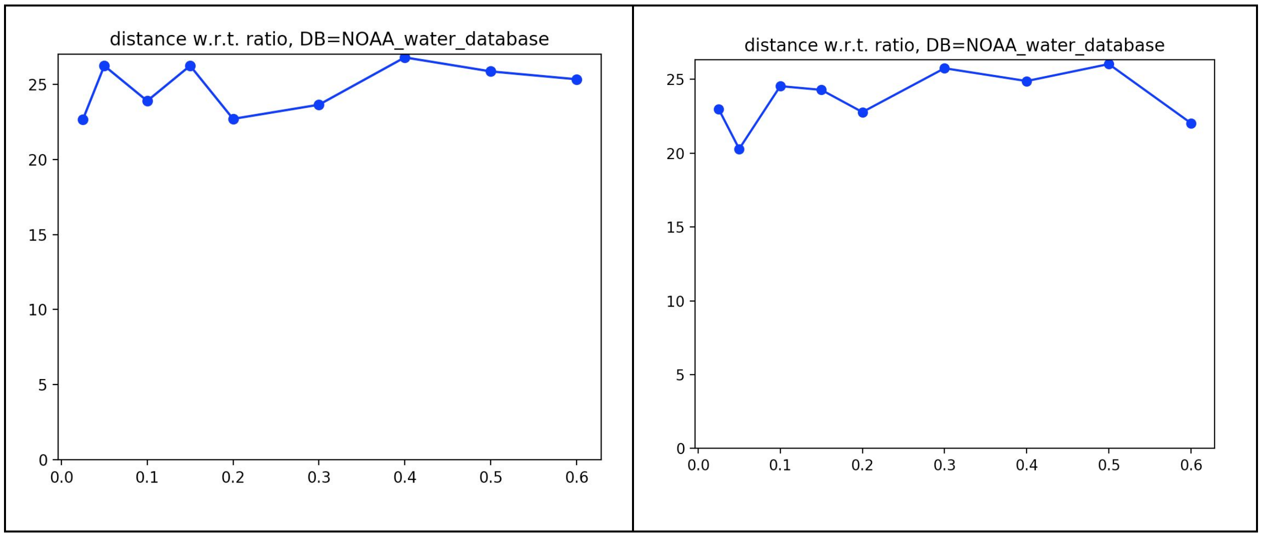


2. NOAA: H2o_temperature table: no obvious pattern

original data size: 380

alphabet size avg: 10

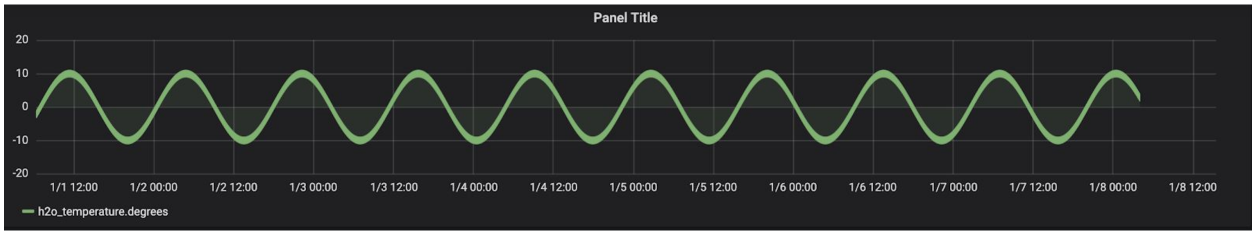


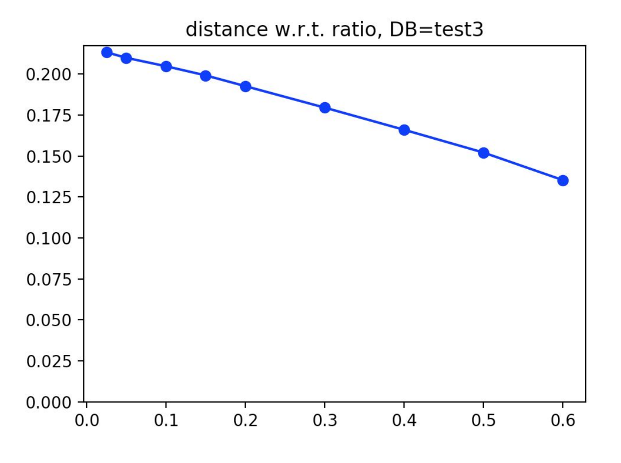


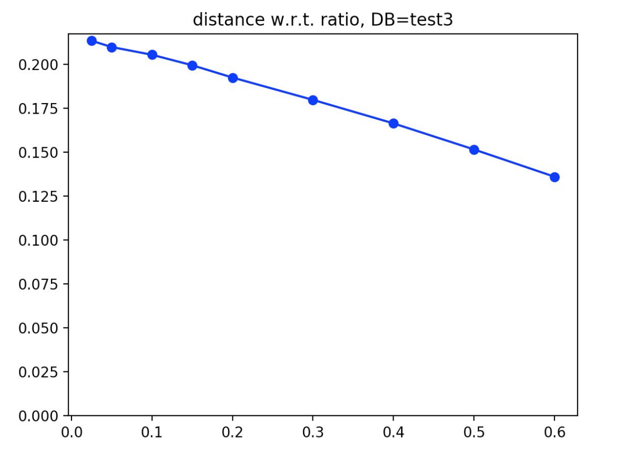
3.SINE_DB: obvious pattern

original data size: 593094

alphabet size avg: 22







Observation:

Data set with more obvious patterns have a smaller distance, so it's hard to set a distance lower bound that's appropriate for ALL sizes of data sets. Therefore, a distance lower bound needs to be chosen for each data set. Below is our solution: (given that the max data size frontend can show without obvious delay is 12000)

- For data set with an obvious pattern: choose the sample ratio with the smallest distance when the sample size <= 12000.
- For data set with an unobvious pattern: simply choose the largest sample size frontend can show, that is 12000.

Future Work:

Although currently, our middleware is able to intelligently determine the sample size based on different data sets and data patterns, the determining process is largely dependent on the query input and the process needs to be done in advance. Our program works well if we know what query to run in advance, however, if Grafana wants to send a large number of different queries, it’s not realistic to run the pre-analyze program on each query. Therefore, a more “intelligent” sample size determines an algorithm is needed.

Note: a more comprehensive test data, and our research notes can be found in https://docs.google.com/document/d/1n134Bv2ykH8haWJdncO-He2slL8xXL_xn9JqGNiL7ZU/edit?usp=sharing