Agg Cache – Caching Time-Series data Aggregates to improve performance

Team: Mingyi Lim and Eugene Chang

Problem

Time series data is huge, with Twitter's Observability stack collecting 170 million metrics every minute and serving 200 million queries per day[i]. Time series databases are typically used for metrics and log storage for large systems and applications, and are typically. One common use of Timeseries data is real-time analytics and dashboarding, where users are able to create visualizations to track specific metrics over time.

However, many clients achieve this by simply re-querying data repeatedly. This repeated queries can put a strain on the time series database by repeatedly querying and aggregating the same data over and over again, which is not required in most cases. We want to investigate if the use of an intermediate caching layer can improve performance of these queries and reduce redundant queries.

Proposal

We propose a cache layer in-between the Time Series database which reduces the amount of data the database needs to handle. The cache layer should provide the following

1. Caching - given an existing query, we can retrieve data from the cache where available.
2. Query simplification – We can modify queries to the TSDB to only the subset of data not present in cache
3. Intermediate Aggregated queries
    a. Due to the density of timeseries data, most queries downsample queries by aggregating data over fixed time ranges with some granularity.
    b. Aggregations are then commonly performed over the range of data captured in these granularities
    c. We want to cache these aggregations to avoid performing repeated aggregations - something that is not done in existing time series caches[ii]

Measure/Metrics

- We will compare the performance of raw queries over our caching system to measure performance gains in terms of
- Total Query latency
- Bandwidth utilization (how much data is served from the DB to the cache)

Tangible Steps for execution

1. Setup connection to Influx DB/Timescale DB (preferably Influx due to it's more traditional TSDB structure)
2. Generate dummy test data which emulates real-world system data (either from a real application or using a generator)
3. Create a set of queries for measurement
4. Create cache (using C/Rust/Golang/Scala) to serve queries
    - Alternatively, we can create client which does caching (just as a prototype, since it is simpler to write, and we don't have to build a cache from scratch). We can ballpark the network latency using similar requests.
    - Cache/Client should hold data in memory, modify query based on existing data and query DB for additional datapoints required.
5. Run comparison using 2 clients
    - Standard Client directly connecting to Influx
    - Modified Client connecting to Influx via cache service

We might not have to use HPC, but we could probably use GCP (if available) to allocate resources to run our experiments.

---

[i] 2013. How Twitter monitors millions of time series. https://www.oreilly.com/ content/how-twitter-monitors-millions-of-time-series/.

[ii] TSCache: an efficient flash-based caching scheme for time-series data workloads https://doi.org/10.14778/3484224.3484225 Jian Liu Kefei Wang Feng Chen 2021