Gorilla Encoding for High Cardinality Metrics in Pinot

Author: Qiaochu Liu TING CHEN | Uber
Status: In Review (internal)
Issue: https://github.com/apache/pinot/issues/15913

Justification

The Gorilla algorithm uses techniques like delta-of-delta encoding and variable-length encoding to compress timestamp and numeric values efficiently, reducing the storage space and improving query performance for time-series data. Supporting the Gorilla algorithm in Pinot will help Pinot enhance the functionality and expand the roadmap to high cardinality metrics and time series use cases.

Proposal

A common use case for Pinot is handling time-series data, especially for monitoring, logging, or event tracking. Efficient storage and fast querying of time-series data are essential for metric based observability use cases.

The Gorilla algorithm was introduced by Facebook in their "[Gorilla: A Fast, Scalable, In-Memory Time Series Database]()" paper. It compresses time-series data using:

- Delta-of-delta algorithm for timestamps
- XOR-based compression for floating-point values

These methods provide impressive compression ratios and speed advantages, enhanced Apache Pinot's time-series data ingestion. Based on Pinot's architecture, it's possible to add the delta of delta encoding, XOR based compression to Pinot.

Design

Compression for Timestamps

This approach works by storing differences between consecutive timestamp deltas, allowing for very compact storage, especially in cases where timestamps are evenly spaced.

Previously, we stored the numbers as follows, each integer takes 64 bits to store.

10101	10103	10104	10106	10105	...

Now we can store the delta between timestamp values as follows,

10101 Base value	2	3	5	4	...

Furthermore the delta of delta will be

10101 Base value	2 Base delta	-1	2	-1	...

After we adjust the numbers for each timestamp, we store the values using variable length integer encoding, instead of fixed bytes. With the above changes, we can compress the timestamp values. In many cases, time intervals between data points are consistent, so delta-of-delta values are often zero or small, which makes them highly compressible.

Variable Length Encoding

Variable length encoding compresses data by representing values with the minimum number of bits required. Instead of using a fixed number of bits for every value, it assigns fewer bits to smaller or more common values and more bits to larger or less common values.

E.g. we have a number 300 which can be represented as follows

1	0	0	1	0	1	1	0	0

It used 8 bytes to store the integer.

With Variable length encoding, it can be represented by two bytes.

0	0	0	0	0	1	0

And

1	1	0	1	1	0	0

Compression for Floating-Point Data

Floating-point numbers representation can lead to small changes in value resulting in similar bit patterns. Similar to timestamp, we can do the following steps to compress

1. Storing the First Value: The first floating-point number is stored in its entirety as a reference point.
2. XOR Operation: For each subsequent floating-point number, compute the XOR between the current value and the previous value. This works well because small changes in value will lead to fewer differing bits, resulting in a smaller binary representation.
3. Delta Encoding: The result of the XOR operation gives a "delta" that represents the change from the previous value. This delta can often be stored more efficiently than the full value.
4. Bit-Packing: The resulting deltas can be packed tightly, taking advantage of the fact that many deltas will have leading zeros or will fall within a small range.

An example is as follows,
Consider the following sequence of floating-point numbers: 10000.0, 10000.3, 10000.9, 10000.65

1. Store 10000.0 as the first value.
2. Compute XOR:
   - 10000.3 XOR 10000.0
   - 10000.9 XOR 10000.3
   - 10000.65 XOR 10000.9
3. Store the resulting deltas instead of full values.

Integration with Pinot Interface

We will introduce two new CompressionCodec

None
```
public enum CompressionCodec {
    LZ4,
    ZSTANDARD,
    DELTA_OF_DELTA,
    XOR;
}
```

Taking delta of delta as an example, we can integrate the following properties into Pinot table Config.

None
```
{
  "fieldConfigList": [
    {
      "name": "timestamp",
      "compressionCodec": "DELTA_OF_DELTA"
    },
    {
      "name": "value",
      "compressionCodec": "XOR"
    },
...
  ]
}
```