The Introduction Of openTSDB

1. Introduction

OpenTSDB is a distributed Time Series Database (TSDB) based on HBase. OpenTSDB was written by Benoit Sigoure to collect, store and display metrics of various computer systems (network gears, operating systems, applications), and generate readable data graphs easily. The developer claims that OpenTSDB is the first open-source monitoring system built on an open-source distributed database.

OpenTSDB is written in Java.

2. History

OpenTSDB was originally written by Benoit Sigoure in 2010 to monitor metrics of the StumbleUpon search engine which requires storing over 1 billion data points per day. StumbleUpon was in charge of the initial development and its open-source release. Yahoo! is currently maintaining OpenTSDB along with the open-source community.

3. System Architecture

OpenTSDB consists of three components: tCollector, Time Series Daemon (TSD), and HBase. One instance of tCollector is deployed on each server. It is responsible to periodically pull metrics data from processes running on the server and the operating system. TSDs receive data from the tCollectors and push data to the HBase backend storage system. Upon receiving queries, TSD scans HBase and retrieves relevant data.

OpenTSDB supports Telnet and HTTP protocols for data writing. All communication takes place on the same port. TSD determines the client protocol by looking at the first few bytes it receives. In addition, OpenTSDB provides a built-in simple WebUI user interface for visually displaying the data of one or more selected indicators and labels. Alternatively, OpenTSDB can be bound to external systems,

such as monitoring frameworks, dashboards (Grafana), and so on, using the HTTP API.

All communications are done via TSD RPC and Hadoop RPC, therefore all components are stateless. There can be as many TSDs as needed to handle the workload as the system scales.

4. Data Model

OpenTSDB's data model is a tag-based single-valued model, that is, there is only one index value per row of records (data points) written, as shown below.

metric	timestamp	value	tags

Timestamp is in seconds or milliseconds. Value supports integer, floating point, and EVENTS or histograms in JSON format. Tags are mainly used to distinguish different data sources. Generally, tags are used as dimension values to support aggregation operations on different dimensions (such as sum, count, AVG, etc.).

5. Storage Mechanism

OpenTSDB is a time series database built on HBase. HBase is a KV wide table storage system. Therefore, the storage mechanism of OpenTSDB is based on the KV wide table model of HBase. You can run the SRC /create_table.sh script in the OpenTSDB source directory to create the following four tables on HBase.

- **UID_TABLE** stores the mapping between names and ids. Dictionaries metric and tags to reduce the storage space of repeated strings.
- **TSDB TABLE** stores the original data points.
- **TREE_TABLE** is used to organize the time series into a file system-like tree structure.
- **META TABLE** is used to record the total number of timelines and other records.

5.1 UID_TABLE design

UID_TABLE is divided into name and ID column clusters. Under each column cluster, metrics, TAGk, and TAGV have three fixed column qualifiers. The name column cluster stores the mapping between ID and name, and the ID column cluster stores the mapping between name and ID, as shown below. In the default configuration of OpenTSDB, the id mapped by name is 3 bytes wide.

Row Key	Column Family : name			Column Family : id		
	metrics	tagk	tagv	metrics	tagk	tagv

The TSD node of OpenTSDB is stateless. How can ids in the OpenTSDB dictionary table be generated to ensure that multiple TSD nodes concurrently create the same ID with the same name? This is mainly achieved by means of atomicIncrement and compareAndSet of HBase. In the ID column family, there is a row with a single-byte key \x00 and three columns, metrics, tagk, and tagv. Each column maintains an 8-byte signed integer that represents the maximum UID currently allocated for each column.

Therefore, UID allocation is mainly divided into the following two steps:

① OpenTSDB invokes atomicIncrement of HBase on the corresponding column to obtain the new UID.

② Set rowKey=UID, value=name on the name column cluster using CAS with expected empty data by calling compareAndSet. If the data rowKey=UID and value=name already exists, the CAS fails, indicating that the dictionary mapping of another node is successful. The node only needs to call GET to obtain the mapping relationship.

Therefore, it can be seen that the way OpenTSDB allocates uids is relatively simple. However, if step 2 fails, the Uids allocated in step 1 will be wasted, so it is possible that the Uids allocated in step 1 are not continuous. In addition, because UID allocation is implemented by atomicIncrement and compareAndSet of HBase, the allocation speed of UID becomes a bottleneck when a large number of new timelines

are written simultaneously.

5.2 TSDB TABLE design

OpenTSDB data points are stored in the TSDB_TABLE by default, including second/millisecond value data points, comments, and histograms. The data table has only one column cluster named "t", as shown below.

Row Key			Column Family : t						
			+0	+60	+120	...	+300	...	+3600

5.2.1 Row Key

Row Key is a byte array of optional salt, metric id, base_timestamp, and tagk/v pair, [salt][...] . By default, UID is encoded in 3 bytes. Starting with OpenTSDB 2.2, an optional salt is added to the row key to better avoid hot issues with data writes.

Base_timestamp is encoded in 4 bytes in seconds. If the timestamp of the ata point is timestamp, base_timestamp = timestamp / 3600. The advantages of this method are that it avoids filling too many data points in a row and supports scanning of data points by metric and time range during query, improving the efficiency of data query. Before generating the row key, the UID of the tagk/v pair needs to be sorted lexicographically, that is, sorted according to the lexicographic order of tagK from smallest to largest, to ensure that the row key generated by the same metric and tags with the same basic timestamp is unique.

5.2.2 The Data Points Column

OpenTSDB's design for the data point column qualifier is clever enough to store numeric data points, annotations, and histograms in the same table with the number of bytes of the column qualifier. The Qualifier holds the timestamp, data type, data length, and other information in one or more datapoints.

5.3 TREE_TABLE design

Tree_table, which organizes time series into a layered file system-like structure

for use with tools such as Graphite or other dashboards. The tree is defined by a set of rules that process TSMeta objects to determine where in the hierarchy, if any, a time series should occur.

5.4 META TABLE design

Meta_table stores indexes of different time series in OpenTSDB, which can contain metadata of each sequence and data points stored in each sequence. Note that data will only be written to this table if OpenTSDB is configured to track metadata or if a user creates a TSMeta object through the API. There are only one column family (name) is used, with two types of columns (ts_meta and ts_counter).

6. Advantage

① Hbase storage is used, and no single point of failure exists.

② Hbase provides almost unlimited storage space. Supports permanent storage and capacity planning

③ Easy to customize graphics

④ Data points can be expanded to 10 billion levels.

⑤ Extend metrics to K (i.e. CPU usage, count as a metric, i.e. a metric is one monitoring item)

⑥ Supports data at the second level.

For Operation & Maintenance Engineer, OpenTSDB can obtain real-time status information about infrastructure and services, and display hardware and software errors, performance changes, and performance bottlenecks of clusters.

For managers, OpenTSDB can measure system SLAs, understand interactions between complex systems, and show resource consumption. The overall performance of the cluster can be used to assist in budgeting and cluster resource coordination.

For developers, OpenTSDB can show the cluster's major performance bottlenecks, the bugs that often occur, so they can focus on solving the important problems.

7. Disadvantage

On the one hand, the operation and maintenance challenges are great. The TSD node needs to be maintained as well as hbase and HDFS. On the other hand, OpenTSDB does not build an inverted index for the timeline, making multidimensional retrieval difficult.

- OpenTSDB relies on the HBase CAS to allocate UIDs. If a large number of new timelines need to be written at the same time, The UID allocation becomes a bottleneck.
- Moreover, the TSD node needs to read the UID table data of the HBase and maintain a UID cache on the TSD node. Therefore, the TSD node can only be restarted in a rolling manner. Otherwise, the HBase cluster may be suspended due to high READ UID throughput.
- If the HBase node jitter or hangs, the write throughput of OpenTSDB drops off a cliff. Generally, you need to restart the TSD node to recover. The main reason is that the Hbase client AsynchBase used by OpenTSDB has a Bug that cannot detect changes of the underlying Hbase RegionServer.

7. Development Trend

According to the source code submissions of OpenTSDB on Github, the most active event segments of OpenTSDB were from 2013 to 2016, and there were very few submissions after 2019, indicating that the activity of OpenTSDB is decreasing.

Based on the ranking of the time serial database on DB-Engines, OpenTSDB's position in the rankings of the sequential databases on DB-Engines continues to decline. Meanwhile, OpenTSDB's overall score remained unchanged from 2016 to 2021, indicating that OpenTSDB's growth has stalled.