Why do we need time-series database

陈云星
学号： 2019141460506

Abstract: This report firstly introduces the background of the popularity of time-series database. And then give a general introduction about what is time-series data and why we need time-series database.

Keywords: time-series databaset; IO; B+ Tree; LSM Tree;

I. INTRODUCTION

In an era, the emergence of explosive products is often inseparable from the background of the era. And one of the biggest technological backgrounds today is the booming development of big data intelligence. With the joint advancement of algorithm innovation, computing resource improvement, and data accumulation, many breakthroughs have been made in the fields of intelligent computing, such as Computer Vision and Natural Language Processing. Thanks to these breakthroughs, data-based smart applications and services have become closely integrated into human life.

A significant driver of today's intelligent applications is large amounts of data. Previously, the most traditional data storage sources were usually located in giant cloud data centers. These huge data centers generate and store big data information, including online shopping records, social media access records and so on. However, this traditional system is being changed with the production mode and characteristics of data. With the rapid popularization of mobile computing and the Internet of things, application devices have generated billions of bytes of data in intelligent terminals. This change will push the resources required for processing computing from the center to the edge, that is, to make processing computing closer to the end terminals rather than the centralized cloud servers.

As mentioned above, new changes have taken place in the production of data and characteristics of data. At IoT terminals, data with structured and temporal characteristics is often generated continuously and in large quantities in a very stable stream. In the context of such requirement, products with the storage and computing capabilities needed to deal with the Internet of Things, that is, time series databases, came into being.

II. WHY TIME-SERIES DATABASE

Figure 1 shows the trend of popularity of different types of databases in the past two years. It is obvious from the figure that time series DBMS shows the highest growth trend. Driven by the above requirements, the popularity of time-series databases must mean that traditional relational databases cannot cope with the above situation well. Then why can't time series data be directly stored in relational database?

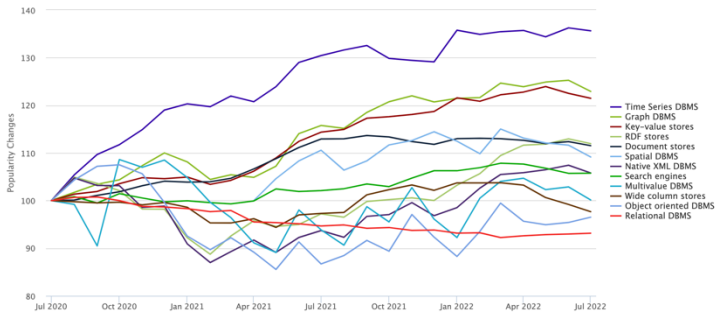

Figure 1. Popularity Changes of Different DBMS

We can push the situation to the extreme, taking the largest amount of data generated - Equipment sensor data in the industrial field as an example. The continuous working time of equipment in these industrial fields is long, and the frequency of working condition data collection is high. The collection and processing of many working condition data will bring great pressure to storage and calculation. Take a very typical specific case - Goldwind power generation data acquisition as an example. Goldwind has more than 2W fan devices, and one fan has 120-510 sensor devices. The acquisition frequency of each sensor is up to 50Hz. This means that each sensor will collect 50 data points to collect peaks in one second. Through calculation, we can know that all wind turbine equipment of Goldwind will generate a total of 500million time series index points per second. Although the amount of data generated is extreme, it can also fully show that the amount of data generated by current devices makes data collection, storage, and calculation face great challenges. Of course, some common acquisition / query requirements in modern business can also reach 200000 / s. Although it is much smaller than the throughput of 500million / s, it is still very difficult for a stand-alone relational database to cope with such a demand.

In common relational databases, two data storage structures are commonly used - B tree and b+ tree. The write speed of both is $\log_{Bn}$. This involves not only the time complexity of the algorithm itself, but also the IO of the disk. Usually, it takes about 10ms to read the disk IO of a data block. Therefore, we should try to reduce the number of disk accesses when designing the database. In the process of building a tree, because it involves the splitting of nodes, the disk blocks used for storage will be split. This will further generate a large amount of random disk I/O. Corresponding to random disk I/O is sequential disk I/O. The differences between the two are as follows:

- Sequential I/O: the initial sector address given by this I/O and the end sector address of the previous i/o are completely continuous or not much separated. In sequential I/O access, the disk requires less track search time, and the read / write head can access the next block with minimal movement.
- Random I/O refers to continuous read and write operations, but discontinuous access sector addresses, which are randomly distributed in the address space of the disk Lun.

Random I/O may be caused by disk fragmentation causing disk space discontinuity, or the current block space is smaller than the file size. The reason why continuous I/O is more efficient than random IO is that when doing continuous I/O, the magnetic head hardly needs to change lanes, or the time of changing lanes is very short; For random I/O, if I/O is very frequent, it will cause the magnetic head to constantly change lanes, resulting in a great reduction in efficiency. As shown in the figure, we can see that the performance difference between sequential write and random write of different disks is almost an order of magnitude. Therefore, sequential disk I/O is very important when high write performance is required.

So, we can see that the algorithm complexity and random disk I/O brought by the b+ tree of relational database will reduce the data writing performance. Time series databases usually use algorithms with high write performance, such as LSM tree (log structured merge tree).

LSM tree gives up part of the ability to read data in exchange for maximizing the writing ability. The core idea of this algorithm is not difficult. Assuming that the memory is large enough, the latest data should be stored in memory first, instead of having to be written to disk every time there is data update. Wait until the data has accumulated to a certain amount, and then use the merge sort method to merge and append the data in memory to the end of the disk queue. At the same time, sequential IO is achieved by optimizing the disk dropping operation.

The second reason is massive data storage. When relational databases store time series, there will be a lot of redundant data.

First of all, we can analyze several attributes of time series data:

- Metric: the indicator of data collection, corresponding to the table in the relational database. Like the wind.
- Tag: dimension column, used to record the ownership of data, which generally does not change over time. For example, which city and which equipment produced it.
- Field: indicator column, which represents the test value of data, and generally changes over time. Such as wind force value, speed value, etc.
- Timestamp: represents the time node of data generation.
- Data point: a specific time interval. It contains a series of continuous timestamp data, similar to a row in a relational database.

If these data are recorded in the form of relational database, a large amount of redundant data will be generated on tag, and data points cannot be represented efficiently. For time series, poor data compression will lead to the occupation of many machine resources, which will lead to a sharp increase in costs.

The third reason is that in terms of reading, for time series data, it is often necessary to query hundreds of millions of data on the second level, such as grouping aggregation or down-sampling.

Although LSM Tree solves the B+ tree's writing problem at the expense of query, it will use some auxiliary means, such as index, BloomFilter cardinality estimation and other means to speed up the query. At the same time, various databases will have different cache mechanisms to speed up the query. So the Key-Value storage with LSM tree structure still have good read-write performance.

III. CONCLUSION

Starting from the general background of time series data, this paper first describes the current edge computing around intelligent hardware, which leads to the protagonist of the report - time series database Next, the report explains why time series data should be stored in time series database instead of traditional relational database from three aspects. The main disadvantages of traditional relational databases are:

- High storage cost: the compression effect of time series data is not good, resulting in the occupation of a large number of storage devices and increased costs.
- High maintenance cost: data tables need to be maintained manually.
- Low write performance: the single machine has low write performance, which is difficult to meet the data write requirements of tens of millions of levels per second.
- Low write performance: the single machine has low write performance, which is difficult to meet the data write requirements of tens of millions of levels per second.
- Poor query performance: poor ability to aggregate massive data.

The advantages of time series database are:

- Support tens of millions of data point writes per second

- Support packet aggregation and downsampling queries of tens of millions of data per second.
- Solve the data redundancy problem of relational database.