**Introduction to TDengine**

TDengine is an efficient platform for storing, querying, and analyzing time series big data. It is specially designed for optimization of the Internet of Things, Internet of Vehicles, Industrial Internet, and operation and maintenance monitoring. TDengine does not rely on any open source or third-party software, has completely independent intellectual property rights, and has high performance, high reliability, scalability, and zero management. Easy to learn and other technical features. Provides functions such as caching, data subscription, and streaming computing to minimize the complexity of R&D and O&M.

The author of TDengine is the Taosi data team led by Tao Jianhui(Jeff Tao), the founder of Background Taosi Data Technology Co., Ltd.

**TDengine summarizes the twelve characteristics of IoT data:**

1. The data is time series and must have a timestamp;

2. The data is structured;

3. The data is rarely updated or deleted;

4. The data source is unique;

5. Relative to Internet applications, write more and read less;

6. Users pay attention to the trend over a period of time, not the value of a certain point in time;

7. The data has a retention period;

8. The query and analysis of data must be based on time period and geographical area;

9. In addition to storage queries, various statistical and real-time computing operations are often required;

10. The flow is stable and predictable;

11. Some special calculations such as interpolation are often required;

12. The amount of data is huge, and the data collected in one day can exceed 10 billion.

After summarizing these characteristics, it will be found that the IoT data is like log data, and there is almost no possibility of update operations, so the implementation of transaction processing in the database is completely redundant; the data is time series, and the timestamp can naturally be used as the primary key. index structure;

IoT data is structured. Like HBase and Cassandra, it is stored with Key-Value. The computing efficiency and storage efficiency are greatly reduced. Structured storage should be used.

The temperature of IoT data is determined by time, and the data just collected is the hottest, not determined by user clicks. Therefore, a simple first-in, first-out memory management can be used to achieve efficient caching, and Redis is not needed at all;

From the perspective of a device, IoT data is a data stream. It is not the most natural thing to realize the stream computing of sliding windows. How can it be used with such a complex engine as Spark;

For data partitioning, it is easy to solve the problem by simply partitioning by device and time period, and no complex partitioning mechanism is needed at all;

The IoT data flow is relatively stable, and the IoT device itself must have the ability to cache. It is possible to completely abandon the Kafka suites and implement a simple message queue and data subscription to meet the needs.

Then I found out that there is a time series database, I immediately looked at their documents and codes, and found that they used some features of time series data, but they still did not fully utilize it, and they were only positioned as a database.

Later, I learned that there are real-time databases in the industry, and found that these real-time databases are all antique products. They are basically developed on Windows. They are expensive, and there is no standard SQL. There is almost no horizontal expansion, and there is almost no big data analysis capability. The ability to cope with the growing demand for big data and big data analysis will sooner or later become obsolete.

A distributed message queue with high reliability and persistent storage. The messages that each mobile phone needs to push are placed in a queue. Is there a difference between message queues and time series data for IoT? Essentially none.

One is unstructured and one is structured;

One is simple in and out, but the other requires analysis and calculation;

There is no major difference in the architectural design of the system between the two.

Therefore, TDengine quickly positioned its own product, which is the IoT big data platform, which integrates a series of functions such as time series database, cache, message subscription, and streaming computing to solve the IoT big data problem in one stop. In this way, the complexity and cost of system development and maintenance can be greatly reduced.

After studying the characteristics of IoT data, TDengine made two technological innovations:

1. The data model of "one table for one device" greatly improves the data insertion and query efficiency of a single device.
2. Give each table a static label, and store the static label data and the collected dynamic data completely separately to solve the problem of multi-table aggregation query.

**Open source protocol of TDengine**

TDengine is currently open sourced on GitHub. Adoption is AGPL licensed. The full name of AGPL is: GNU Affero General Public License, which is a supplement to GPL and adds some restrictions on the basis of GPL.

Open source project address: https://github.com/taosdata/TDengine

**The development history of TDengine**

* In 2017, Taosi Data began to develop TDengine
* In August 2018, the first product of TDengine was really handed over to users. It took about a year and a half from the initial research and development to the launch of the available products.
* On July 12, 2019, Taosi Data officially announced that the kernel (storage and computing engine) and community version of TDengine will be 100% open source
* In January 2020, Taosi Data received nearly 10 million US dollars of Pre-A round investment from GGV and other companies to develop an open source IoT big data platform TDengine
* In April 2020, Taosi Data announced the completion of the A round of financing of more than 10 million US dollars, and completed two rounds of 10 million US dollars of financing within three months

**Advantages of TDengine**

More than 10 times performance improvement: defines an innovative data storage structure, a single core can process at least 20,000 requests per second, insert millions of data points, and read more than 10 million data points, which is faster than existing general-purpose databases more than ten times.

Hardware or cloud service costs are reduced to 1/5: due to super performance, computing resources are less than 1/5 of general big data solutions; through columnar storage and advanced compression algorithms, storage takes up less than 1/10 of general-purpose databases.

Full-stack time series data processing engine: Integrate database, message queue, cache, stream computing and other functions, and applications do not need to integrate Kafka/Redis/HBase/Spark/HDFS and other software, greatly reducing the complexity and cost of application development and maintenance. Seamless integration with third-party tools: Integrate with Telegraf, Grafana, Matlab, R without a single line of code. In the future, MQTT, OPC, Hadoop, Spark, etc. will be supported, and BI tools will also be seamlessly connected.

Powerful analysis functions: Whether it is data ten years ago or one second ago, you can query it by specifying a time range. Data can be aggregated on a timeline or across multiple devices. Ad hoc queries are available anytime via Shell, Python, R, MATLAB.

High Availability and Horizontal Scaling: Through distributed architecture and consensus algorithms, through multiple replication and clustering features, TDengine ensures high availability and horizontal scalability to support mission-critical applications.

Zero operation and maintenance cost and zero learning cost: The cluster installation is simple and fast, no need for sub-database and sub-table, and real-time backup. Similar to standard SQL, supports RESTful, supports Python/Java/C/C++/C#/Go/Node.js, similar to MySQL, zero learning cost.

Core open source: Except for some auxiliary functions, the core of TDengine is open source. Businesses will no longer be database bound. This makes the ecosystem stronger, the products more stable, and the developer community more active.

**Applicable scenarios of TDengine**

As a basic software, TDengine has a wide range of applications. In principle, it can be used in all places where machines, equipment, and sensors are used to collect data.

Some typical scenarios are listed below:

* Public safety: Internet records, call records, individual tracking, interval screening
* Power industry: centralized monitoring of smart meters, power grids, and power generation equipment
* Communication industry: detailed bill of calls, user behavior, base station/communication equipment monitoring
* Financial industry: transaction records, access records, ATM, POS machine monitoring
* Travel tools: real-time monitoring of trains/cars/taxi/planes/bicycles
* Transportation industry: real-time road conditions, intersection flow monitoring, bayonet data;
* Petrochemical: real-time monitoring of oil wells, transportation pipelines, and transportation fleets
* Internet: Server/application monitoring, user access logs, ad click logs
* Logistics industry: tracking and monitoring of vehicles and containers
* Environmental monitoring: monitoring of weather, air, hydrology, geological environment, etc.;
* Internet of things: elevators, boilers, machinery, water meters, gas meters and other networking equipment
* Military industry: data collection and storage of various military equipment
* Manufacturing: Production process control, process data, supply chain data collection and analysis

**TDengine's write storage strategy**

1. Single point write

Although the amount of time series data is huge, the data source of each device is unique because the process of generating data from different acquisition devices is completely independent. And when a table has only one writer, naturally there is no need to waste resources on the locking mechanism. You must know that the write operation of a traditional relational database must be protected by a lock.

Tdengine uses lock-free writing, which saves a lot of resources and speeds up writing.

2. Continuous storage

Secondly, for a data collection point, since the data it generates is time series, it is a naturally sorted data structure. Therefore, the way of subsequent writing is realized by orderly append (append), which can give full play to the performance of the hard disk and further improve the data writing speed.

3. Super watch

Each device in TDengine corresponds to a table. However, information such as the device address number is not necessary to be written to the disk. So they put a lot of tables together and turned it into a super table, and then just use the address number and other information to filter it when querying.

4. Columnar Compression

TDengine uses column storage. Because the content format of each column in the column store is similar, it is beneficial to compress and save space. Moreover, different compression algorithms are adopted for different types of data. After targeted compression, another regular compression is performed. Therefore, the space occupied by the data finally written to the hard disk will be very small.

**Features of TDengine**

It is specially designed for the data of the Internet of Things, and uses the characteristics of the time series of the data of the Internet of Things to realize the function that each collection point corresponds to a table. But it is not suitable for processing general internet data.

Columnar storage + compression is used to save hardware costs. (High compression efficiency: Using the characteristics of IoT data with little fluctuation, compression after dif interpolation, and then second-order compression, the efficiency is very high.)

High availability is supported, and each physical node is divided into multiple virtual data nodes and virtual management nodes. Virtual data nodes store data, and virtual management nodes manage MetaData. The virtual data nodes and virtual management nodes are distributed on different physical nodes to achieve high availability of data set applications.

In terms of storage structure, a separate table is created for each collection point for storage. In this way, the data of each collection point is continuously stored and the reading efficiency is improved. Since there is only one data source for each table, lock-free writing can be achieved and the writing rate can be improved.

For variable aggregation, the concept of super table is introduced. A collection device of the same type can create a hypertable. When creating a super table, you can specify tags for this type of table, and filter the tables in the database through tags when querying, so that even if there are a lot of tables in the database, you can achieve fast multi-table aggregation.

The installation package is very small and easy to install and use. SQL is supported, and the syntax is similar to MySQL.