1. Why we need TDengine?

The traditional database is unable to cope with the huge amount of data, high speed of data input and abundant data sources.

MySQL has the following problems in the scenario of massive time series data:

* High storage cost: poor compression of time series data requires a lot of machine resources;
* High maintenance cost: for single machine system, it is necessary to manually divide the warehouse and table on the upper layer, so the maintenance cost is high;
* Low write throughput: the single machine has low write throughput, which is difficult to meet the writing pressure of tens of millions of sequential data;
* Poor query performance: it is suitable for transaction processing, and the aggregation and analysis performance of massive data is poor.

In addition, using Hadoop ecosystem (Hadoop, spark, etc.) to store timing data will have the following problems:

* High data delay: offline batch processing system, which takes hours or even days from data generation to analysis;
* Poor query performance: unable to make good use of indexes, relying on MapReduce tasks, and the query time is generally at the level of minutes.

However, the time series database needs to solve the following problems:

* Writing timing data: how to support the writing of tens of millions of data points per second.
* Reading of time series data: how to support the grouping and aggregation of hundreds of millions of data at the second level.
* Cost sensitive: massive data storage brings cost problems. How to store these data at a lower cost will become the top priority of time series database.

As a member of time series database, TDengine solves the above problems well, and also has the following advantages:

* TDengine has a wide range of characteristics and requirements for data sources that can be processed. TDengine provides efficient horizontal expansion function, and has a storage structure that matches high compression ratio to achieve the best storage efficiency, so it is suitable for data sources with a large amount of data.
* TDengine can continuously process a large amount of input data in the same hardware environment and provide a performance evaluation tool that can easily run in the user environment, so it can process data sources with extremely high data input speed.
* TDengine is specifically optimized for a large number of data sources. It is especially suitable for efficiently importing, writing and querying data from billions of data sources, so it can handle a large number of data sources.

1. How was TDengine designed?
2. Data model

* Typical scenarios of Internet of things

In typical Internet of things, Internet of vehicles, operation and maintenance monitoring scenarios, there are often many different types of data acquisition devices that collect one or more different physical quantities. For the same type of acquisition equipment, there are often multiple specific acquisition equipment distributed in different places. Big data processing system is to summarize all kinds of collected data, and then calculate and analyze. For the same kind of equipment, the data collected are very regular.

* Data characteristics

In addition to timing characteristics, careful study shows that Internet of things, Internet of vehicles, and operation and maintenance monitoring data also have many other obvious characteristics:

* Highly structured data;
* Data is rarely updated or deleted;
* Transaction processing without traditional database;
* Compared with Internet applications, write more and read less;
* The flow is stable, which can be predicted according to the number of equipment and acquisition frequency;
* Users focus on the trend of a period of time, rather than the value of a specific point in time;
* Data has retention period;
* The query and analysis of data must be based on time period and spatial area;
* In addition to storage and query operations, various statistical and real-time calculation operations are also required;
* The amount of data is huge, and more than 10billion pieces of data can be collected in a day.

Taking full advantage of the above features, tdengine adopts a specially optimized storage and computing design to process timing data, which significantly improves the system processing capacity and greatly reduces the complexity of system operation and maintenance.

* Relational database model

Because the collected data is generally structured data, and in order to reduce the learning threshold, tdengine uses the traditional relational database model to manage the data. Therefore, users need to create libraries first, and then create tables before inserting or querying data. Tdengine uses structured storage rather than key value storage of NoSQL.

* One data collection point and one table

In order to make full use of the timing and other data characteristics of its data, tdengine requires that a separate table be created for each data collection point (for example, if there are 10 million smart meters, 10 million tables need to be created, and d1001, d1002, d1003, d1004 in the above tables need to be created separately) to store the timing data collected by this collection point. This design has several advantages:

* It can ensure that the data of a collection point is continuously stored in blocks on the storage medium. If you read data in a period of time, it can greatly reduce random reading operations and improve the reading and query speed by the order of magnitude.
* Because the data generation process of different collection devices is completely independent, the data source of each device is unique, and there is only one writer for a table, so it can be written in an unlocked way, and the writing speed can be greatly improved.
* For a data acquisition point, the data generated is sequential, so the write operation can be realized by adding, which further greatly improves the data write speed.

If the traditional method is used to write the data of multiple devices into a table, due to the uncontrollable network delay, the timing of data from different devices arriving at the server cannot be guaranteed. The writing operation should be protected by a lock, and it is difficult to ensure the continuous storage of data from one device. The method of one data collection point and one table can ensure the best performance of the insertion and query of a single data collection point to the greatest extent.

* Super Table: collection of data collection points of the same type

Because one data collection point has one table, the number of tables increases greatly, which is difficult to manage. Moreover, applications often need to do aggregation operations between collection points, and the aggregation operations also become complex. To solve this problem, tdengine introduces the concept of Super Table (called table for short).

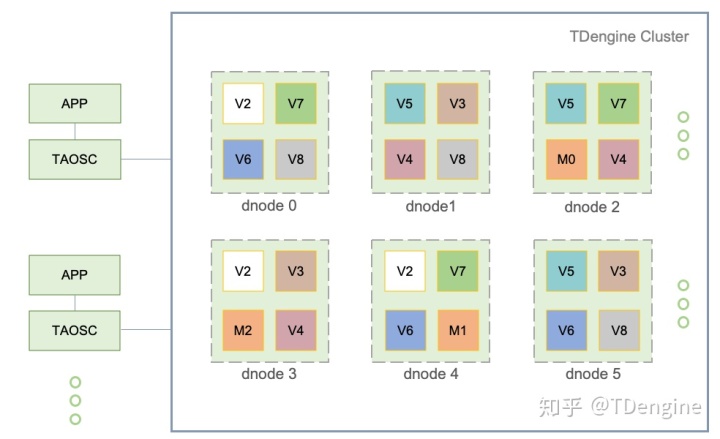
A super table is a collection of data collection points of a specific type. The table structure of the same type of data collection point is exactly the same, but the static attributes (labels) of each table (data collection point) are different. Describe a super table (the combination of a specific type of data collection points). In addition to defining the table structure of the collection quantity, you also need to define the schema of its label. The data type of the label can be integer, floating point number, string, and there can be multiple labels, which can be added, deleted, or modified afterwards. If the whole system has n different types of data collection points, n super tables need to be established.

In the design of tdengine, the table is used to represent a specific data collection point, and the super table is used to represent a set of data collection points of the same type. When creating a table for a specific data collection point, the user uses the definition of the super table as a template and specifies the label value of the specific collection point (table). Compared with traditional relational databases, tables (a data collection point) have static labels, and these labels can be added, deleted and modified afterwards. A super table contains multiple tables with the same temporal data schema but different tag values.

When aggregating multiple data collection points with the same data type, tdengine will first find the tables that meet the label filtering conditions from the super table, and then scan the time series data of these tables for aggregation operation, which can greatly reduce the data sets that need to be scanned, so as to greatly improve the performance of aggregation computing.

1. Clusters and basic logical units

The logic structure diagram of tdengine distributed architecture is as follows:

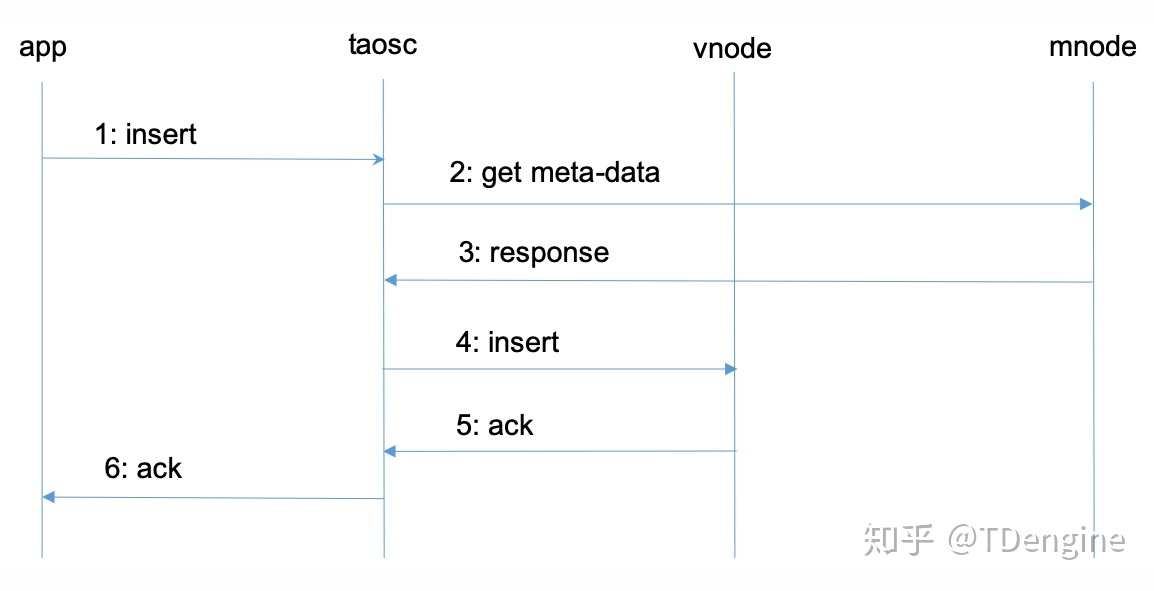


A complete tdengine system runs on one or more physical nodes. Logically, it includes data nodes (dnode), tdengine clients (TAOSC) and Applications (APPs). There are one or more data nodes in the system, which form a cluster. The application interacts with tdengine cluster through TAOSC API.

Follows are its components:

* Physical node (pnode): pnode is a computer that runs independently and has its own computing, storage and network capabilities. It can be a physical machine, virtual machine or container with an OS installed. Physical nodes are identified by their configured FQDN (fully qualified domain name).
* Data node (dnode): dnode is a running instance of the tdengine server side execution code taosd on the physical node. A working system must have at least one data node. Dnode contains zero to more logical virtual nodes (vnodes) and zero or more logical management nodes (mnodes). The unique identification of dnode in the system is determined by the end point (EP) of the instance. EP is the combination of FQDN (fully qualified domain name) of the physical node where dnode is located and the network port number (port) configured by the system. By configuring different ports, a physical node (a physical machine, virtual machine or container) can run multiple instances or have multiple data nodes.
* Virtual node: in order to better support data fragmentation and load balancing, and prevent data overheating or tilting, data nodes are virtualized into multiple virtual nodes (vnode, V2, V3, V4, etc. in the figure). Each vnode is a relatively independent work unit and the basic unit of sequential data storage, with independent running threads, memory space and persistent storage paths. A vnode contains a certain number of tables (data collection points). When creating a new table, the system will check whether a new vnode needs to be created. The number of vnodes that can be created on a data node depends on the hardware resources of the physical node where the data node is located. A vnode belongs to only one dB, but a DB can have multiple vnodes. In addition to the stored timing data, a vnode also saves the schema and tag values of the included tables. A virtual node is uniquely identified in the system by the EP of its data node and its vgroup ID, and is created and managed by the management node.
* Management node (mnode): a virtual logical unit, which is responsible for the monitoring and maintenance of the operation status of all data nodes, as well as the load balancing between nodes (m in the figure). At the same time, the management node is also responsible for the storage and management of metadata (including users, databases, tables, static tags, etc.), so it is also called meta node. Multiple (no more than 5) mnodes can be configured in the tdengine cluster, and they are automatically built into a virtual management node group (M0, M1, M2 in the figure). Mnodes are managed by the master/slave mechanism, and data synchronization is carried out in a strong consistent way. Any data update operation can only be carried out on the master. The creation of mnode cluster is automatically completed by the system without human intervention. There is at most one mnode on each dnode, which is uniquely identified by the EP of the data node to which it belongs. Each dnode automatically obtains the EP of all the dnodes in the whole cluster through internal message interaction.
* Virtual node group: vnodes on different data nodes can form a virtual node group to ensure the high reliability of the system. The virtual node group is managed by master/slave. Write operations can only be performed on the master vnode. The system uses asynchronous replication to synchronize data to the slave vnode, which ensures that a copy of data can be copied on multiple physical nodes. The number of virtual nodes in a vgroup is the number of copies of data. If the number of replicas of a DB is n, the system must have at least N data nodes. The number of replicas can be specified through the parameter replica when creating a DB. The default is 1. Using the multi copy feature of tdengine, you can get the same high reliability of data without expensive disk arrays and other storage devices. The virtual node group is created and managed by the management node, and the management node assigns a system unique ID, vggroup ID. If the vnode group IDs of two virtual nodes are the same, they belong to the same group, and the data is backed up to each other. The number of virtual nodes in the virtual node group can be dynamically changed, allowing only one, that is, there is no data replication. Vgroup ID will never change. Even if a virtual node group is deleted, its ID will not be recovered and reused.
* Taosc: TAOSC is the driver provided by tdengine to applications, which is responsible for handling the interface interaction between applications and clusters, and is embedded in JDBC, ODBC driver, or C, python, go language connection library. Applications interact with the whole cluster through TAOSC rather than directly connecting the data nodes in the cluster. This module is responsible for obtaining and caching metadata; Forward the insertion, query and other requests to the correct data node; When returning the results to the application, you also need to be responsible for the final level of aggregation, sorting, filtering and other operations. For JDBC, ODBC and c/c++ interfaces, this module runs on the physical node where the application is located, but the consumption of resources is very small. At the same time, to support the fully distributed restful interface, TAOSC has a running instance on each dnode of the tdengine cluster.

Their working process is as follows:



1. The application initiates the request of inserting data through JDBC, ODBC or other API interfaces.
2. TAOSC will check the cache to see whether meta data of the table is saved. If so, go directly to step 4. If not, TAOSC will send a get meta data request to mnode.
3. Mnode returns the meta data of this table to TAOSC. Meta data contains the schema of the table, as well as the vgroup information to which the table belongs (vnode ID and the end point of the dnode. If the number of copies is n, there are n groups of end points). If TAOSC fails to get a response from mnode for a long time, and there are multiple mnodes, TAOSC will send a request to the next mnode.
4. TAOSC sends an insert request to the master vnode.
5. After vnode inserts data, it sends a reply to TAOSC, indicating that the insertion is successful. If TAOSC fails to get a response from vnode for a long time, TAOSC will think that the node is offline. In this case, if the inserted database has multiple copies, TAOSC will send an insert request to the next vnode in the vgroup.
6. TAOSC notifies app that the writing is successful.
7. Storage model and data partition and fragmentation

* Data slicing

For massive data management, in order to achieve horizontal expansion, it is generally necessary to adopt sharding partitioning strategy. Tdengine realizes data partition through vnode, and realizes temporal data partition through a data file in a time period.

Vnode (virtual data node) is responsible for providing write, query and calculation functions for the collected timing data. In order to facilitate load balancing, data recovery and support heterogeneous environments, tdengine divides a data node into multiple vnodes according to its computing and storage resources. The management of these vnodes is automatically completed by tdengine and is completely transparent to applications.

For a single data collection point, regardless of the amount of data, a vnode (or vnode group, if the number of copies is greater than 1) has enough computing and storage resources to process (if a 16 byte record is generated per second, the original data generated in a year is less than 0.5g), so tdengine stores all the data of a table (a data collection point) in a vnode, The data of the same collection point will not be distributed to two or more dnodes. Moreover, a vnode can store data from multiple data collection points (tables), and the upper limit of the number of tables that a vnode can accommodate is onemillion. In design, all tables in a vnode belong to the same dB. On a data node, unless specially configured, the number of vnodes owned by a DB will not exceed the number of system cores.

When creating a DB, the system will not allocate resources immediately. However, when creating a table, the system will check whether there is an allocated vnode and whether there is a free table space in the vnode. If so, create a table in the vnode with an empty space immediately. If not, the system will create a new vnode on a dnode from the cluster according to the current load, and then create a table. If the DB has multiple replicas, the system will not only create a vnode, but a vgroup (virtual data node group). The system has no limit on the number of vnodes, only limited by the computing and storage resources of the physical node itself.

MEDA data (including schema, labels, etc.) of each table is also stored in vnode, rather than in mnode. In fact, this is the segmentation of meta data, which is convenient for label filtering operations in an efficient and parallel manner.

* Data partition

In addition to vnode fragmentation, tdengine also partitions time series data according to time periods. Each data file only contains timing data of one time period, and the length of the time period is determined by the DB configuration parameter days. This method of partitioning by time period also facilitates the efficient implementation of data retention policies. As long as the data file exceeds the specified number of days (system configuration parameter keep), it will be automatically deleted. Moreover, different time periods can be stored in different paths and storage media, so as to facilitate the hot and cold management of big data and realize multi-level storage.

In general, tdengine divides big data through vnode and time, which is convenient for parallel and efficient management and horizontal expansion.

* load balancing

Each dnode regularly reports its status (including hard disk space, memory size, CPU, network, number of virtual nodes, etc.) to mnode (virtual management node), so mnode knows the status of the whole cluster. Based on the overall state, when mnode finds that a dnode is overloaded, it will move one or more vnodes on the dnode to other dnodes. In the process of moving, external services continue, and data insertion, query and calculation operations are not affected.

If mnode does not receive the status report of dnode for a period of time, mnode will think that this dnode has been offline. If the offline time exceeds a certain length (the length is determined by the configuration parameter offlinethreshold), the dnode will be forcibly removed from the cluster by mnode. If the number of copies of vnodes on this dnode is greater than one, the system will automatically create new copies on other dnodes to ensure the number of copies of data. If there are mnodes on this dnode, and the number of copies of mnodes is greater than one, the system will automatically create new mnodes on other dnodes to ensure the number of copies of mnodes.

When new data nodes are added to the cluster, because new computing and storage are added, the system will also automatically start the load balancing process.

The load balancing process does not require any manual intervention, and the application does not need to restart. It will automatically connect new nodes, which is completely transparent.

1. Data writing and replication process

* Data writing

If a database has n replicas, there are n virtual nodes in a virtual node group, but only one is a master, and the others are slaves. When the application writes a new record to the system, only the master vnode can accept the write request. If the slave vnode receives a write request, the system will notify TAOSC that it needs to be redirected.

* synchronous copy

For scenarios with higher data consistency requirements, asynchronous data replication cannot meet the requirements because there is a very small probability of data loss. Therefore, tdengine provides a synchronous replication mechanism for users to choose. When creating a database, in addition to specifying the number of replicas, users also need to specify a new parameter quorum. If the quorum is greater than one, it means that every time the master forwards to the replica, it needs to wait for the quorum-1 reply confirmation before notifying the application that the data has been written successfully in the slave. If no quorum-1 reply confirmation is received within a certain period of time, the master vnode will return an error to the application.

With synchronous replication, the performance of the system will decline and latency will increase. Because the metadata must be consistent, the data synchronization between mnodes is synchronous replication by default.

1. Caching and persistence

* cache

Tdengine adopts time driven cache management strategy (first in first out, FIFO), also known as write driven cache management mechanism. This strategy is different from the read driven data cache mode (least recent used, LRU), which directly saves the recently written data in the system cache. When the cache reaches the critical value, the oldest data is written to disk in batches. Generally speaking, for the use of IOT data, users are most concerned about the newly generated data, that is, the current state. Tdengine makes full use of this feature to store the recently arrived (current state) data in the cache.

Tdengine provides users with millisecond data acquisition capability through query functions. Directly storing the recently arrived data in the cache can respond more quickly to the query analysis of the user for the latest piece or batch of data, and provide faster database query response capability as a whole. In this sense, tdengine can be used as data cache by setting appropriate configuration parameters, without the need to deploy redis or other additional cache systems, which can effectively simplify the system architecture and reduce the cost of operation and maintenance. It should be noted that after tdengine restarts, the system cache will be emptied, and the previously cached data will be written to disk in batches. The cached data will not be reloaded into the cache like the special key value cache system.

Each vnode has its own independent memory, and is composed of multiple fixed size memory blocks. Different vnodes are completely isolated. When writing data, it is similar to the writing method of logs. The data is added to the memory in sequence, but each vnode maintains its own skip list, which is easy to find quickly. When more than half of the memory blocks are full, the disk dropping operation is started, and the subsequent write operation is carried out in the new memory block. In this way, half of the memory blocks in a vnode are reserved with the latest data, so as to achieve the purpose of caching and fast search. The number of memory blocks of a vnode is determined by the configuration parameter blocks, and the size of memory blocks is determined by the configuration parameter cache.

* Persistent storage

Tdengine uses a data-driven method to write the data in the cache to the hard disk for persistent storage. When the cached data in vnode reaches a certain scale, tdengine will also pull the disk dropping thread to write the cached data to persistent storage in order not to block the subsequent data writing. Tdengine will open a new database log file when the data is dropped. After the data is dropped successfully, it will delete the old database log file to avoid unlimited growth of the log file.

In order to make full use of the characteristics of time series data, tdengine divides the data stored in the persistent storage of a vnode into multiple files. Each file only stores data for a fixed number of days, which is determined by the system configuration parameter days. After cutting into multiple files, given the start and end date of the query, you can immediately locate which data files need to be opened without any index, which greatly speeds up the reading speed.

For the collected data, there is usually a retention time, which is determined by the system configuration parameter keep. Data files that exceed this set number of days will be automatically deleted by the system to free up storage space.

Given the two parameters of days and keep, the total number of data files of a vnode is: keep/days. The total number of data files should not be too large or too small. Suitable within 10 to 100. Based on this principle, reasonable days can be set. In the current version, the parameter keep can be modified, but the parameter days cannot be modified once it is set.

In each data file, the data of a table is stored piece by piece. A table can have one or more data file blocks. In a file block, data is stored in columns, occupying a continuous storage space, which greatly improves the reading speed. The size of the file block is determined by the system parameter maxRows (maximum number of records per block), and the default value is 4096. This value should not be too large or too small. If it is too large, the search time for locating the data in a specific time period will become longer, affecting the reading speed; Too small, the index of the data block is too large, and the compression efficiency is low, which also affects the reading speed.

Each data file (.Data end) has a corresponding index file (.Head end). The index file has summary information of a data block for each table, recording the offset of each data block in the data file, the start and end time of the data and other information, so as to help the system quickly locate the data that needs to be searched. Each data file also has a corresponding last file (.End of last), which is designed to prevent the fragmentation of data blocks when the disk is dropped. If the number of records in the disk of a table does not reach the system configuration parameter minrows (minimum number of records per block), it will be stored in the last file first. When the disk is next loaded, the records of the new disk will be merged with the records of the last file, and then written to the data file.

When writing data to disk, decide whether to compress data according to the system configuration parameter comp. Tdengine provides three compression options: no compression, one-stage compression and two-stage compression, corresponding to comp values of 0, 1 and 2 respectively. One stage compression carries out the corresponding compression according to the type of data. The compression algorithms include delta delta coding, simple 8b method, zig zag coding, lz4 and other algorithms. Two stage compression uses general compression algorithm on the basis of one-stage compression, and the compression rate is higher.

* Multilevel storage

Under the default configuration, tdengine will save all data in /var/lib/taos directory, and the data files of each vnode are saved in different directories under this directory. In order to expand the storage space, minimize the bottleneck of file reading and improve the data throughput, tdengine can configure the system parameter dataDir to make multiple attached hard disks used by the system at the same time. In addition, tdengine also provides the function of hierarchical data storage, that is, data files are stored on different storage media according to the new and old degree of data files. For example, the latest data is stored on SSD, more than one week of data is stored on local hard disk, and more than four weeks of data is stored on network storage devices, so as to reduce storage costs and ensure efficient data access. The movement of data on different storage media is automatically completed by the system, which is completely transparent to the application. Hierarchical storage of data is also configured through the system parameter dataDir.

The configuration format of dataDir is as follows:

dataDir data\_ path [tier\_level]

Where data\_ Path is the folder path of the mount point, tier\_ Level refers to the storage level of the media. The higher the level of media storage, the older the data file. Multiple hard disks can be attached to the same storage level, and the data files on the same storage level are distributed on all hard disks of the storage level. Tdengine supports up to three levels of storage, so tier\_ The values of level are 0, 1 and 2. When configuring dataDir, there must be only one mounting path without specifying tier\_ Level, called special mount disk (path). By default, the mounting path is a level 0 storage medium and contains special file links, which cannot be removed, otherwise it will have a devastating impact on the written data.

Suppose a physical node has six mountable hard disks /mnt/disk1, /mnt/disk2,..., /mnt/disk6, where disk1 and Disk2 need to be designated as level 0 storage media, Disk3 and Disk4 are level 1 storage media, and disk5 and disk6 are level 2 storage media. Disk1 is a special mount disk, which can be found in /etc/taos/taos CFG is configured as follows:

dataDir /mnt/disk1/taosdataDir /mnt/disk2/taos 0dataDir /mnt/disk3/taos 1dataDir /mnt/disk4/taos 1dataDir /mnt/disk5/taos 2dataDir /mnt/disk6/taos 2

The attached disk can also be a non local network disk, as long as the system can access it.

Note: the multi-level storage function is only supported in the enterprise version

1. Data query

Tdengine provides a variety of query processing functions for tables and super tables. In addition to conventional aggregation queries, it also provides window queries and statistical aggregation functions for temporal data. The query processing of tdengine needs the cooperation of the client, vnode and mnode nodes.

* Single table query

The parsing and verification of SQL statements are completed on the client. Parse the SQL statement and generate an abstract syntax tree (AST), and then verify and check it. And request the management node (mnode) for the metadata information (table metadata) of the specified table in the query.

According to the end point information in the metadata information, the query request is serialized and sent to the data node (dnode) where the table is located. After receiving the query request, dnode identifies the virtual node (vnode) that the query request points to and forwards the message to the query execution queue of vnode. The query execution thread of vnode establishes the basic query execution environment, returns the query request immediately, and starts executing the query at the same time.

When the client gets the query results, the working thread in the query execution queue of dnode will wait for the execution of the vnode execution thread to complete before returning the query results to the requesting client.

* Aggregation, downsampling, interpolation by timeline

The remarkable feature of time series data different from ordinary data is that each record has a timestamp. Therefore, aggregation of time stamped data on the time axis is an important function different from ordinary databases. From this point of view, it is similar to the window query of stream computing engine.

The keyword interval is introduced into tdengine to segment the fixed length time window on the time axis, aggregate the data according to the time window, and aggregate the data within the window as needed. For example:

select count(\*) from d1001 interval(1h)；

For the data collected by d1001 device, the number of records stored per hour is returned according to the time window of 1 hour.

In the application scenario where query results need to be obtained continuously, if there is data loss in a given time interval, the data results in that interval will also be lost. Tdengine provides strategies to interpolate the results of time axis aggregation calculation. By using the keyword fill, you can interpolate the results of time axis aggregation. For example:

select count(\*) from d1001 interval(1h) fill(prev)；

Collect data for d1001 device and count the number of records per hour. If there is no data in a certain hour, the statistical data of the previous hour will be returned. Tdengine provides forward interpolation (prev), linear interpolation (linear), null value filling (null), and specific value filling (value).

* Multi table aggregate query

The overall process of multi table aggregate query is the same as that of single table query, but there are the following differences:

• Because multiple tables may be distributed in different nodes (dnodes), the aggregation query of multiple tables needs to first obtain the information of all data nodes where the table is located, and send query requests to the relevant dnodes at the same time.

• The partial results obtained by the calculation of each vnode need to be aggregated in the second stage to form the final results. The aggregation process in the second stage is completed on the client.

• Since the table label information is stored in vnode, the query of label information also needs to be completed by vnode. The client encapsulates the filter expression of the tag in the query request structure and sends it to vnode. The query execution thread of vnode extracts the tag query conditions from it, and then executes the query. Tag query and filtering are completed before the query on the table. After the tag query is completed, the qualified tables are included in the next query processing process.

* Pre calculation

In order to effectively improve the performance of query processing, according to the unchangeable characteristics of IOT data, the statistical information of the data stored in the data block is recorded in the header of the data block: including the maximum value, minimum value, and. We call it precomputation unit. If the query processing involves all the data of the entire data block, the pre calculation results are directly used, and there is no need to read the content of the data block. Because the amount of pre calculated data is far smaller than the size of data block stored on the disk, for the query processing with disk IO as the bottleneck, the use of pre calculated results can greatly reduce the pressure of reading IO and accelerate the process of query processing. The precomputer system is similar to postgre SQL's brin (block range index).

1. What are the advantages of TDengine?

TDengine is a high-performance, scalable time-series database with SQL support. Its code, including its cluster feature is open source under GNU AGPL v3.0. Besides the database engine, it provides caching, stream processing, data subscription and other functionalities to reduce the complexity and cost of development and operation. TDengine differentiates itself from other TSDBs with the following advantages.

• High Performance: TDengine outperforms other time series databases in data ingestion and querying while significantly reducing storage cost and compute costs, with an innovatively designed and purpose-built storage engine.

• Scalable: TDengine provides out-of-box scalability and high-availability through its native distributed design. Nodes can be added through simple configuration to achieve greater data processing power. In addition, this feature is open source.

• SQL Support: TDengine uses SQL as the query language, thereby reducing learning and migration costs, while adding SQL extensions to handle time-series data better, and supporting convenient and flexible schemaless data ingestion.

• All in One: TDengine has built-in caching, stream processing and data subscription functions. It is no longer necessary to integrate Kafka/Redis/HBase/Spark or other software in some scenarios. It makes the system architecture much simpler, cost-effective and easier to maintain.

• Seamless Integration: Without a single line of code, TDengine provide seamless, configurable integration with third-party tools such as Telegraf, Grafana, EMQX, Prometheus, StatsD, collectd, etc. More third-party tools are being integrated.

• Zero Management: Installation and cluster setup can be done in seconds. Data partitioning and sharding are executed automatically. TDengine’s running status can be monitored via Grafana or other DevOps tools.

• Zero Learning Cost: With SQL as the query language, support for ubiquitous tools like Python, Java, C/C++, Go, Rust, Node.js connectors, there is zero learning cost.

• Interactive Console: TDengine provides convenient console access to the database to run ad hoc queries, maintain the database, or manage the cluster without any programming.