TDengine

Open Source, High-Performance, Scalable Time-Series Database with SQL Support

Jeff Tao

Founder and core developer of TDengine

What is time-series data?

Time-series data is a series of time-stamped data.

Typical scenarios:

- Data generated by IoT devices, sensors, meters, etc.
- Data generated by vehicles
- Data generated by the agents in DevOps (hosts, VMs, containers …)
- Financial market data

Popular time-series databases on market

- InfluxDB
- TimeScaleDB
- TDengine
- QuestDB
- Prometheus
- OpenTSDB
- VictoriaMetric
- Graphite
- RRDTool
- ………

Why do we need a new Time-Series Database?

1. Developer friendly
   1. Flexible Data ingestion
   2. Query language
   3. Open system
   4. Management efforts

2. Scalability
   1. Vertical scalable
   2. Horizontal scalable

3. Performance
   1. Data ingestion rate
   2. Query latency

4. Beyond database, other components are needed for a modern data processing system
   1. Stream processing
   2. Caching
   3. Data subscription

TDengine

Characteristics of time-series data

1 Data always has time stamp

2 Structured data

3 Data source is like a stream

4 Data is rarely deleted or updated

5 Retention policy is always applied

6 More write than read operations

7 Data rate is pretty stable

8 Real-time data computing is desired

9 Query is always in time and space range

10 Massive data volume

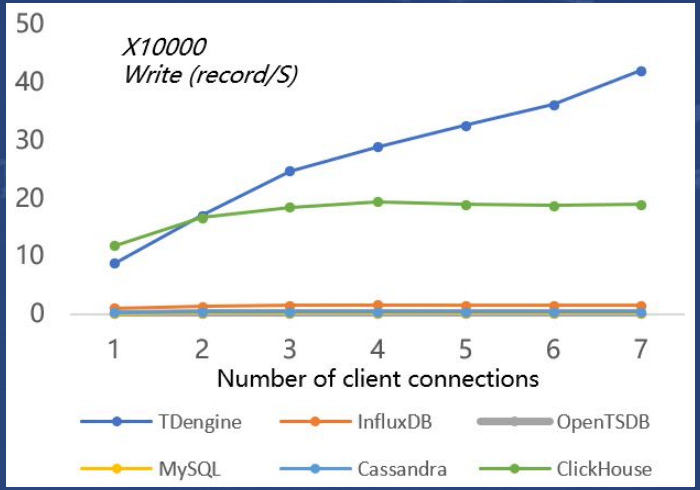
TDengine takes full advantages of the above characteristics

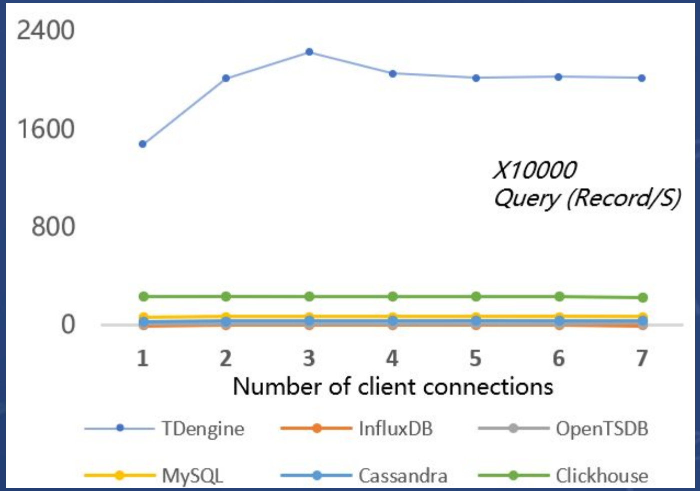
TDengine

TDengine Highlights – High Performance

- Due to purpose built time-series data storage engine, TDengine improves read/write performance **10 times** over generic RDBMS.
- TDengine can process 20k writes or read 10s of millions records in a second by one single core
- The data compression ratio is 5x higher than RDBMS.
- Compared with InfluxDB, data ingestion rate is 2-5 times higher, For query scenarios, a few have the same speed as InfluxDB, while the majority are several times faster, and some are up to 40 times faster. Data compression ratio is 70% higher.

Testing reports are available at: www.tdengine.com

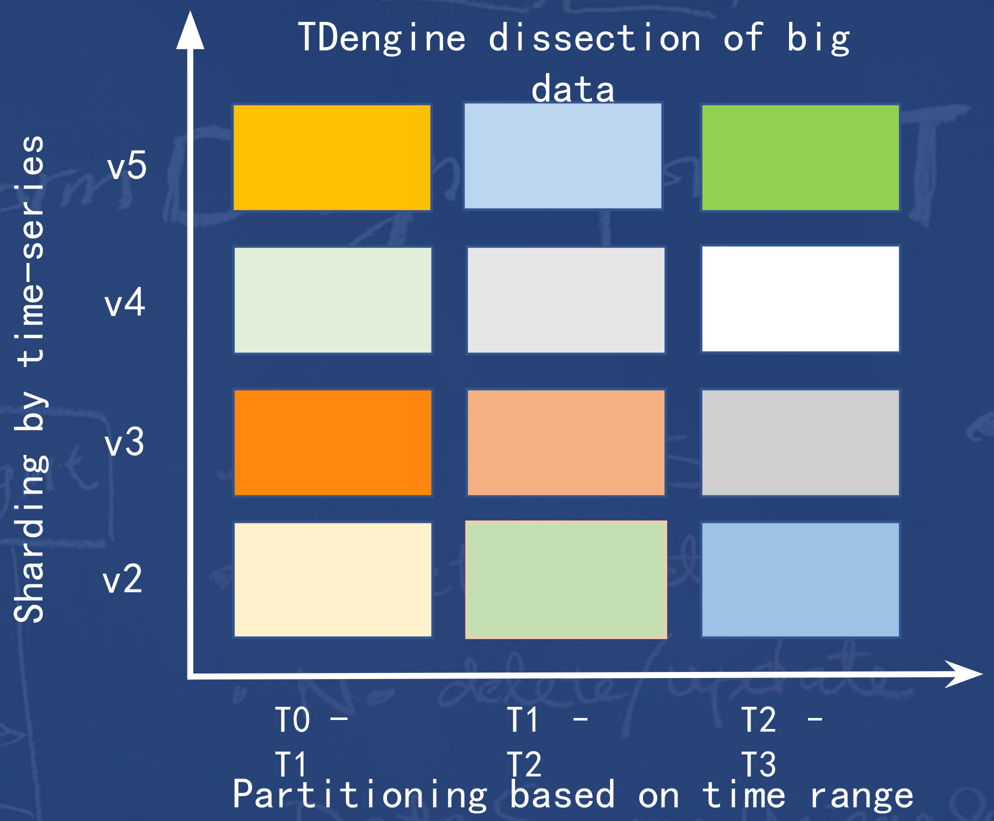




TDengine

TDengine Highlights - Scalable

- Native distributed design
- Support both sharding and partitioning
- A shard contains data for multiple time-series
- A partition contains data for a time range
- A node may contain multiple shards



TDengine dissection of big data

Sharding by time-series

v5

v4

v3

v2

T0 - T1

T1 - T2

T2 - T3

Partitioning based on time range

TDengine – SQL Support

- TDengine supports SQL-like Syntax, and support C/C++, JAVA, GO, Rust, Python, Node JS, RESTful APIs.
- Ad-hoc query through TDengine Shell
- No learning curve, small migration cost

```
create database demo;
use demo;
create table t1(ts timestamp, degree
float);
insert into t1 values(now, 28.5);
insert into t1 values(now, 29.0);
select * from t1;
select avg(degree), count(*) from t1;
```

TDengine - All in One

MSG Queue

Built-in,
No need for Kafka or other message queue

Caching

Provides Real-time Latest Record
No need for Redis

Database

Combine Real-time and Historical DB,
transparent operations

Stream Comp.

Stream Computing over single or multiple devices

Subscription

Latest data pushed to applications automatically

TDengine: All in One Time-Series Data Platform

Full Stack for IoT, No more Kafka, Redis, Spark, HBase, Zookeeper, etc.

Complexity is highly reduced, efficiency and performance are highly improved

TDengine – Resource Requirements

- Installation package is only 3.4M, no any other dependency.
- The minimum memory requirements is less than 16M
- It can be run even on ARM 32, Raspberry Pi

A perfect solution for Edge Computing

TDengine Innovation #1

Data Model

one table for one data collection point (one sensor)

TDengine

Typical scenario: smart power meter

Device ID	Time Stamp	Data Collected			Static Label	
Device ID	Time Stamp	Current	Voltage	Phase	Location	Type
d1001	1538548685000	3.13	220	0.31	CA.SanJose	1
d1002	1538548685100	8.21	219	0.82	CA.PaloAlto	2
d1001	1538548686000	3.11	219	0.35	CA.SanJose	1
d1003	1538548683000	5.41	110	0.53	CA.Cupertino	1
d1002	1538548686100	8.11	223	0.81	CA.PaloAlto	2
d1002	1538548687130	8.15	215	0.85	CA.PaloAlto	2
d1001	1538548687000	3.15	223	0.32	CA.SanJose	1
d1003	1538548684000	5.51	112	0.54	CA.Cupertino	1
d1003	1538548685000	5.60	109	0.53	CA.Cupertino	1
d1002	1538548688100	8.19	218	0.87	CA.PaloAlto	2
d1003	1538548686000	5.62	108	0.56	CA.Cupertino	1
d1001	1538548688500	3.19	221	0.31	CA.SanJose	1

Data Characteristics

- Each record has a time stamp
- Structured, mainly numerical numbers
- Consistent structure unless the firmware is updated
- Each device has a static attribute label
- the time when the data of each device arrives at the server is uncontrollable, but the relative order of the collected data points from each device to the server is basically guaranteed

TDengine

One table for one data collection point (sensor)

Device ID: D1001, label loc: CA.SanJose type: 1

Time stamp	Current	Voltage	Phase
1538548685000	3.13	220	0.31
1538548686000	3.11	223	0.35
1538548687000	3.15	219	0.32
1538548688500	3.19	221	0.33

Device ID : D1002, label loc: CA.PaloAlto type: 2

Time stamp	Current	Voltage	Phase
1538548685100	8.21	219	0.82
1538548686100	8.11	223	0.81
1538548687130	8.15	215	0.85
1538548688100	8.19	218	0.87

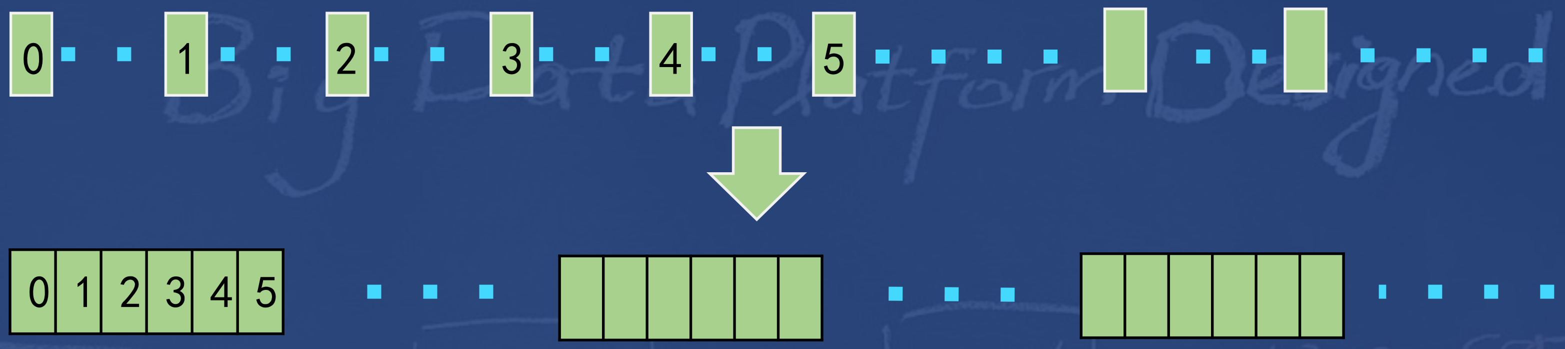
Device ID: D1003, label loc: CA.Cupertino type: 2

Time stamp	Current	Voltage	Phase
1538548683000	5.41	100	0.53
1538548684000	5.51	109	0.54
1538548685000	5.60	112	0.53
1538548686000	5.62	108	0.56

Benefit of the design:

- Records are automatically sorted by time
- Simple append operation for writing new data
- Less fluctuation in column values
- Device ID and label will not be stored repeatedly

Table storage: block by block, each contains a number of records

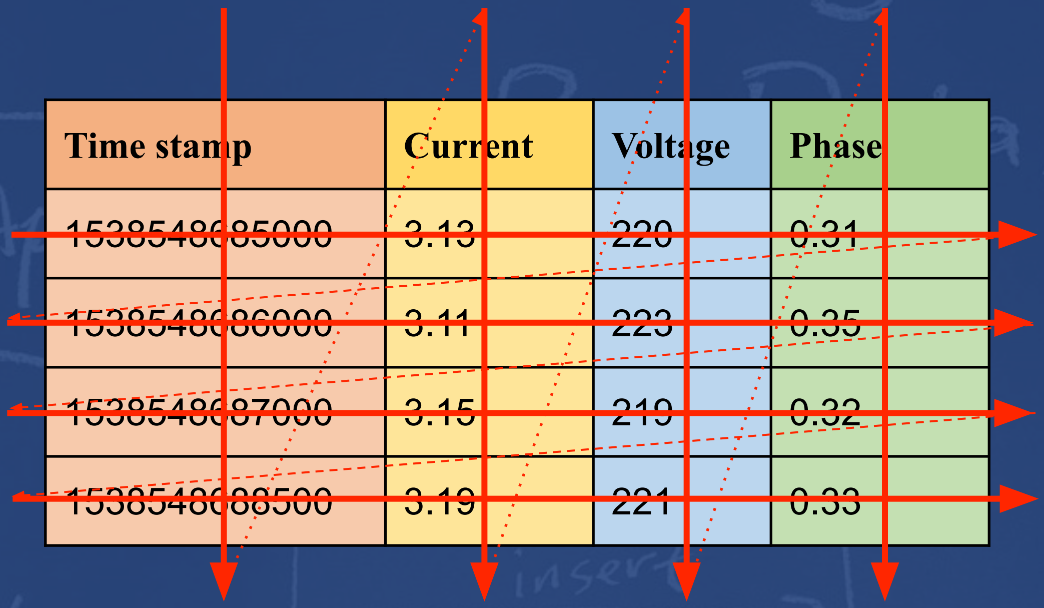


- The records of each table are stored in blocks
- A data block contains a number of records
- Each data block comes with pre-calculation
- Each data block has a schema
- A table often has multiple data blocks
- The system has a block index, based on the start/end time, locate the data block right away

TDengine

Data in the block using column-based store

Column-based Store ✓



Time stamp	Current	Voltage	Phase
1538548685000	3.13	220	0.31
1538548686000	3.11	223	0.35
1538548687000	3.15	219	0.32
1538548688500	3.19	221	0.33

Row-based Store

1538548685000	3.13	220	0.31	538548	1538548686000	3.11	223	0.35	3.11	1538548687000	3.15	219	0.32	9		1538548688500	3.19	221	0.33

- **Compression rate significantly improved:**
  - Similar data in a column to facilitate compression;
  - different data types can use different compression algorithms
- **Analysis performance significantly improved:**
  - The analysis of time series data is often carried out for a collection of data in a certain time period. A lot of unused data will be read using row-based storage

TDengine

TDengine Strategies:

✔ One table for one data collection point (sensor)

✔ Records are stored block by block, and continuously inside a block

✔ Column-based storage for each block

Ensure read/write efficiency of a single data collection point is the best

Other characteristics of time-series big data

- Many data collection points for same type of data, e.g. there may be 30 million smart meters in a province
- The same type of data collection points often need to be aggregated
- Multi-dimensional analysis of collected data required, and the dimensions of analysis may be uncertain during modeling

"One table for one data collection point" brings technical challenges:

The number of tables is huge, difficult to manage, and difficult to aggregate

TDengine Innovation #2

Super Table

efficient aggregation of multiple data collection points

Super Table: a certain type of data collection point

- To describe a super table, i.e. a data collection point type,

  we need to define two schemas:
  - Data schema of the collected data (time-series)
  - Data schema of the static label (attributes)

- Create a table for a specific data collection point:
  - Taking the super table as a template,

    the schema of the table is the collected data schema of the super table
  - Assign specific values to static labels

TDengine

A Super Table Example

Create a super table for the smart power meter, the collected data includes

current,

voltage and phase, and the label includes location and type

```
create table smeter (ts timestamp, current float, voltage int, phase float)
                tags (loc binary(20), type int);
```

Use smeter as a template to create 6 tables for 6 smart meters, and the location

labels are CA.SanJose, CA.PaloAlto, TX.Austin, etc.

```
create table t1 using smeter tags(‘CA.SanJose’, 1);
create table t2 using smeter tags(‘CA.PaloAlto’, 2);
create table t3 using smeter tags(‘CA.Cupertino’, 1);
create table t4 using smeter tags(‘CA.SanJose’, 2);
create table t5 using smeter tags(‘TX.Austin’,1);
create table t6 using smeter tags(‘TX.Dallas’, 1);
```

TDengine

Aggregation and multi-dimensional analysis by super table

Query the average voltage and maximum current of all smart meters in CA SanJose

```
Select avg(voltage), max(current) from smeters where loc = “CA.SanJose”
```

Query the average voltage of the type 1 smart meter in CA

```
Select avg(voltage) from smeters where type = 1 and loc like “CA%”
```

- Super table can be queried like ordinary table, but can specify the filter of labels
- There can be as many as 128 labels, and each label represents an analysis dimension
- Labels can be added, deleted, and modified after data modeling
- Each label can be a tree structure, which narrows the search scope

Label storage

Strategies:

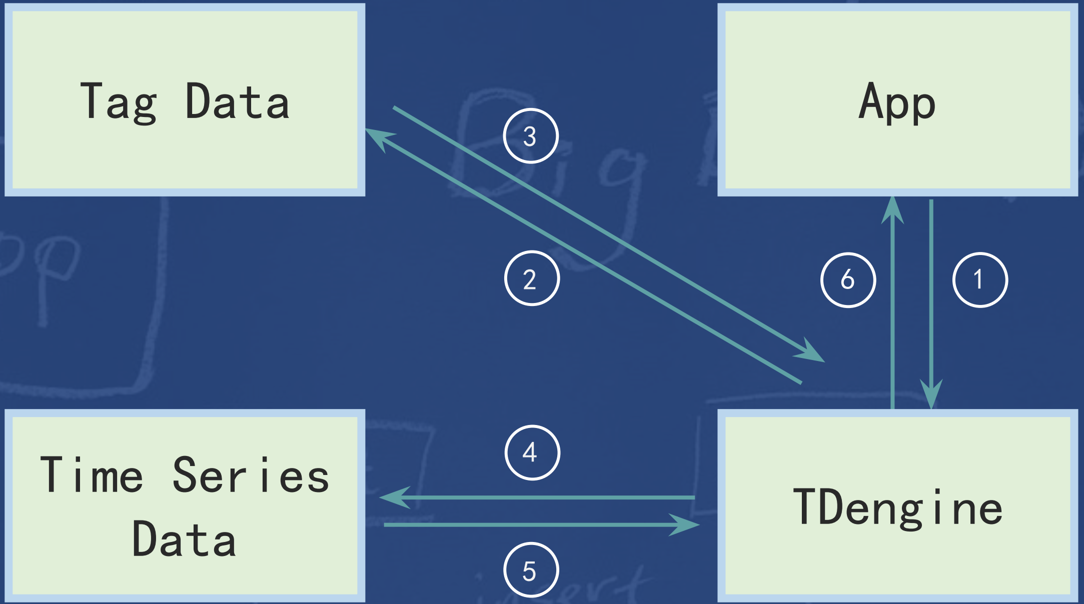
- Label data is stored completely separated from time series data
- Use Key-Value storage to facilitate addition, deletion and modification operations
- One label record for each data collection point
- Label records are stored together and indexed

Advantages:

- Compared to a typical NoSQL database, label values are not stored repeatedly, which reduce storage space significantly
- In multi-dimensional aggregation, first filter the label to find the collection points that need to be aggregated, reduce the aggregated data set significantly
- The total number of label records is equal to the number of collection points, the amount is small enough to be stored in memory to improve query efficiency

TDengine

TDengine aggregation process



Tag Data

App

Time Series Data

TDengine

Main process:

1. App initiates a query request to TDengine
2. TDengine sends the tag filter conditions to the tag data processing module
3. The tag query module returns a list of collection points that meet the filter conditions
4. TDengine notifies the time series data processing module to perform aggregation on the selected data collection points in the specified time period
5. TDengine receives the aggregated result
6. TDengine returns the result to the App

- In fact, Tag Data is a Dimension Table, and TS Data is a Facts Table.
- From another perspective, the design of the super table is a two-level index structure, and the first-level index is a label used to filter data collection points. The second index is a timestamp, used to filter the collected time series data

Advantages of TDengine Super Table:

- ✔ Significantly reduce label storage space
- ✔ Significantly improve the aggregation efficiency of data collection points
- ✔ Enable convenient and efficient multi-dimensional analysis

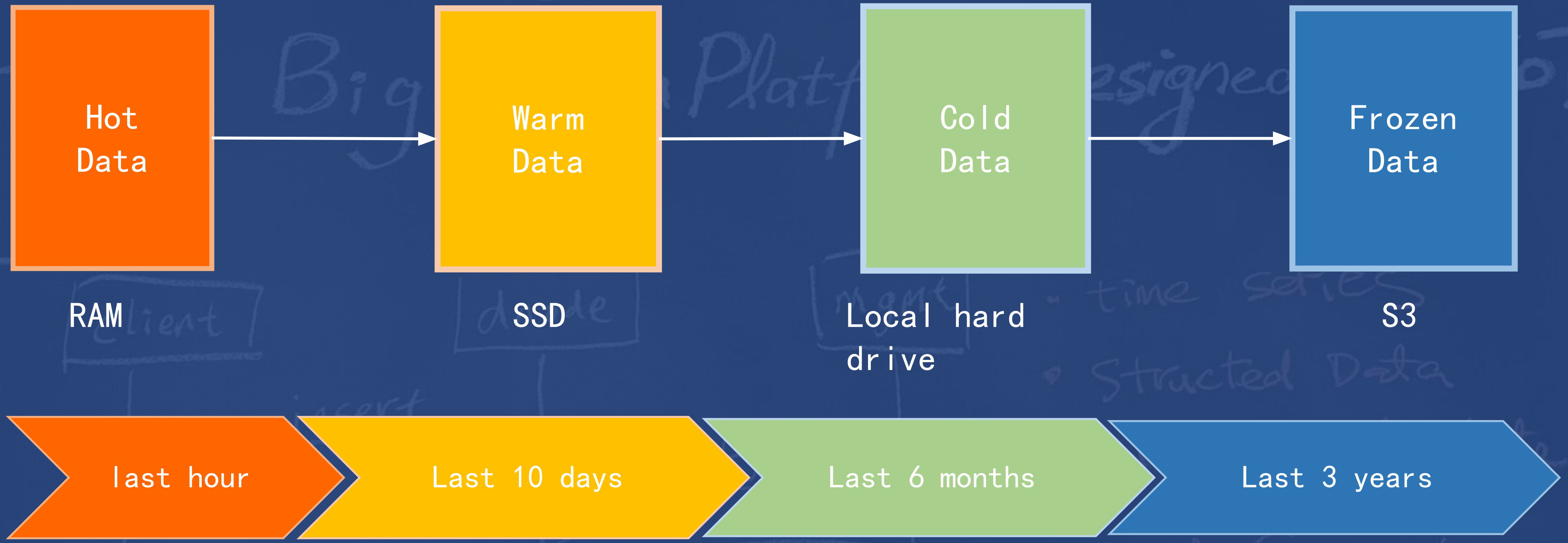
TDengine is a powerful multi-dimensional analysis tool

High Reliability of TDengine

The strategy of high reliability design :

- Guaranteed by WAL (Write Ahead Log)
- Inbound data is always written to WAL first, then to memory, and then reply to the application for confirmation
- If system is crashed, data can be restored from WAL when restarting

Multi-level storage of data in TDengine

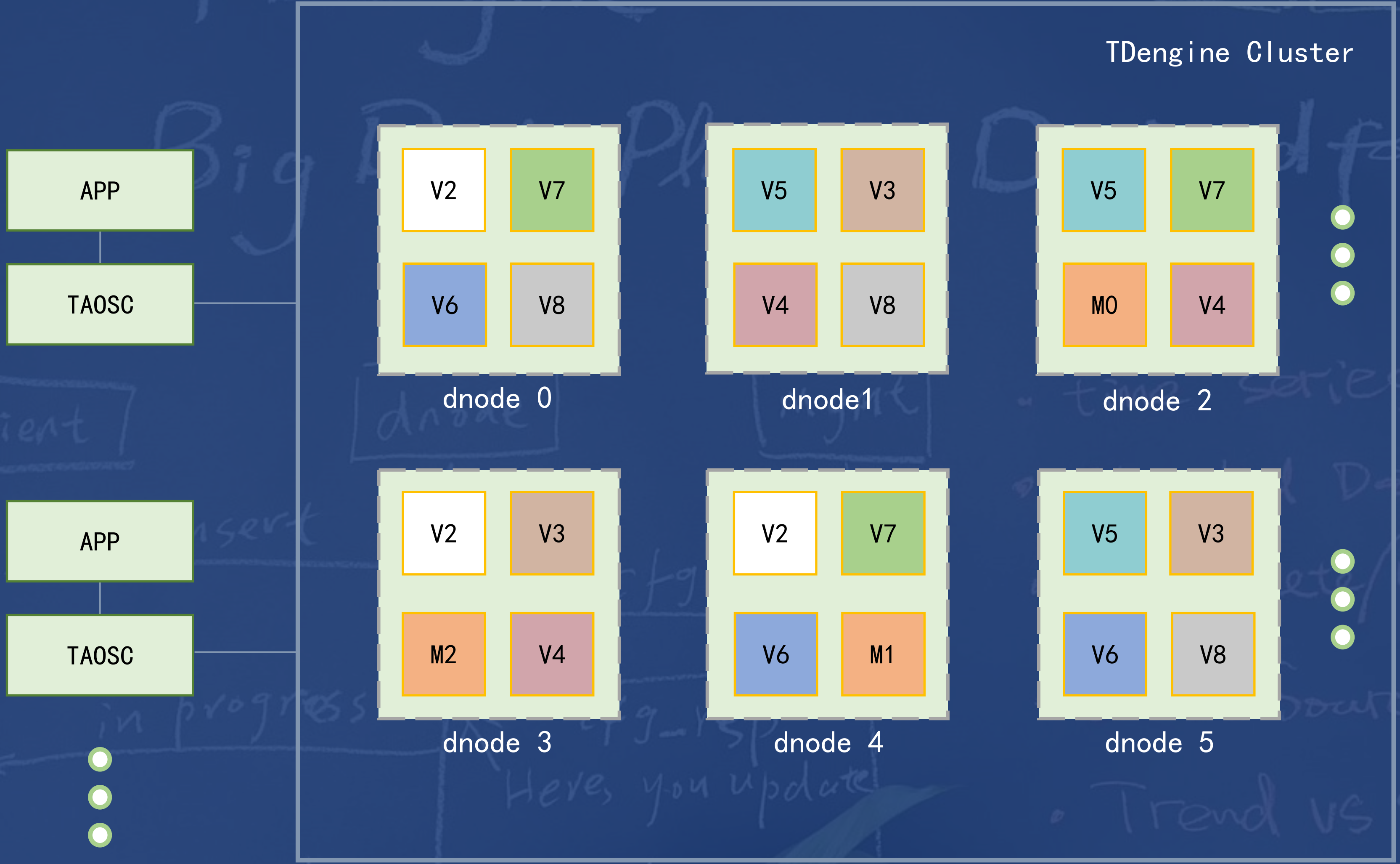


Simply configure the time range and corresponding storage path

,

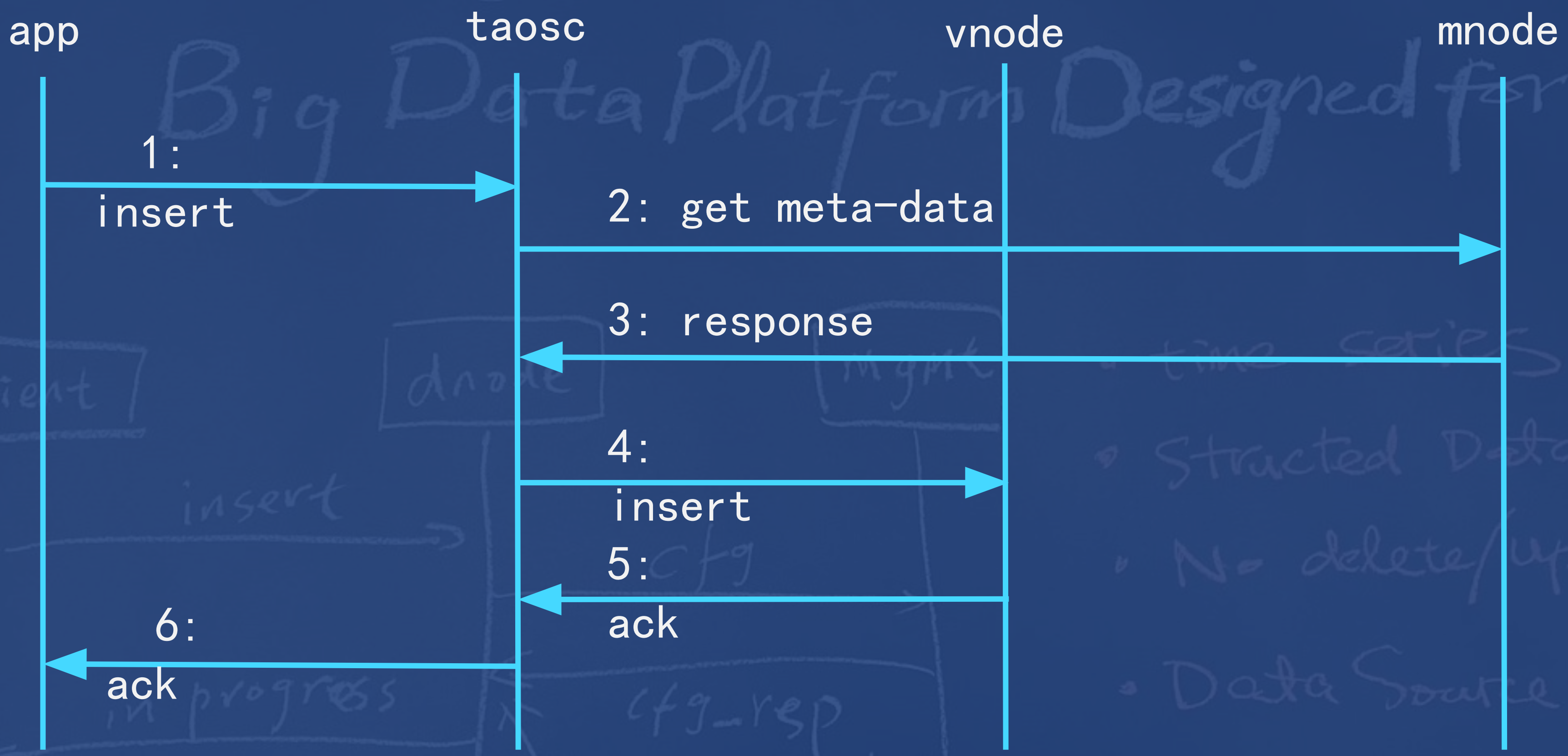
data will be automatically migrated to different storage media

TDengine Cluster Architecture



TDengine

TDengine's typical process



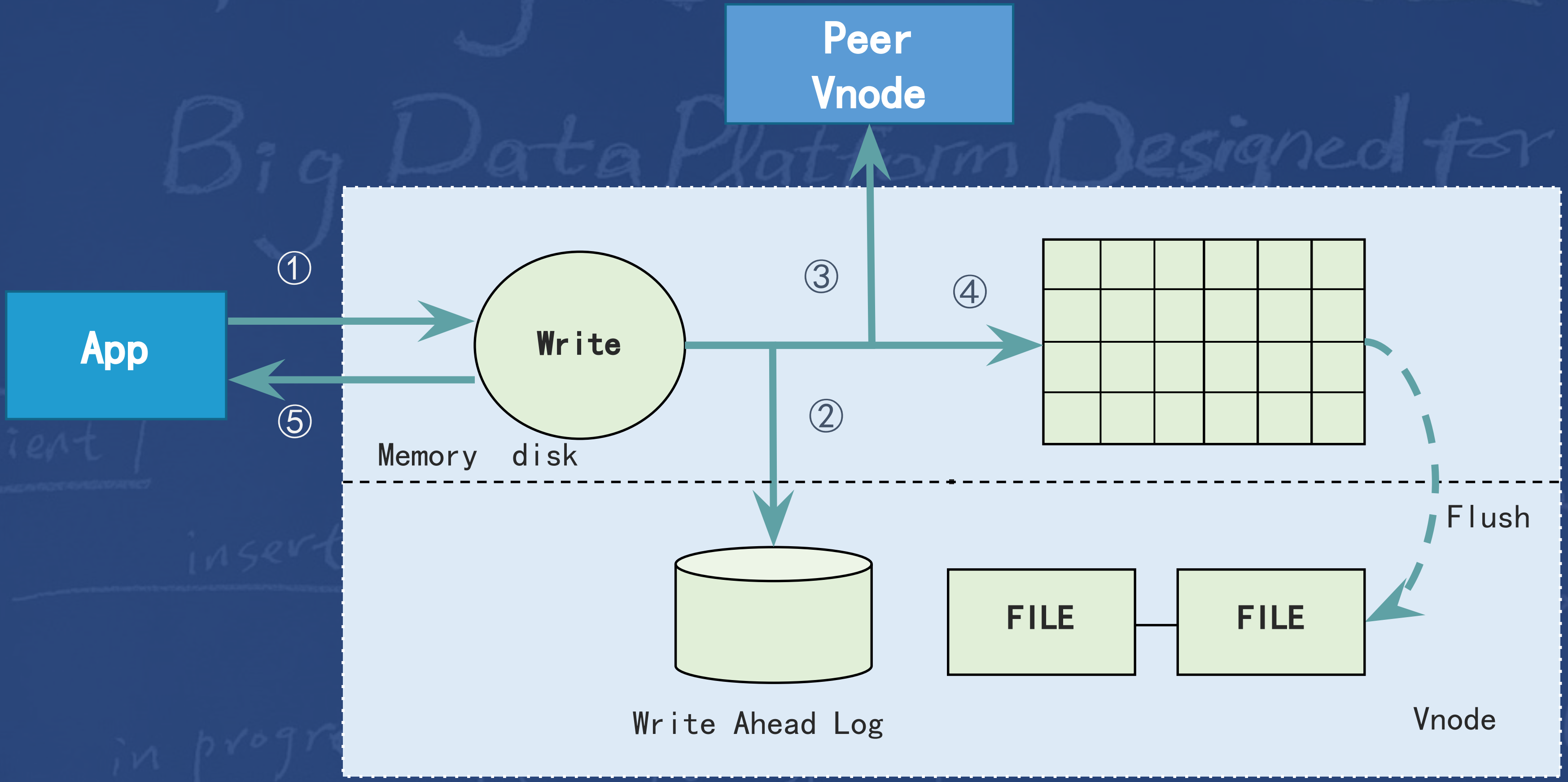
TDengine

High availability of TDengine

Strategies for High availability:

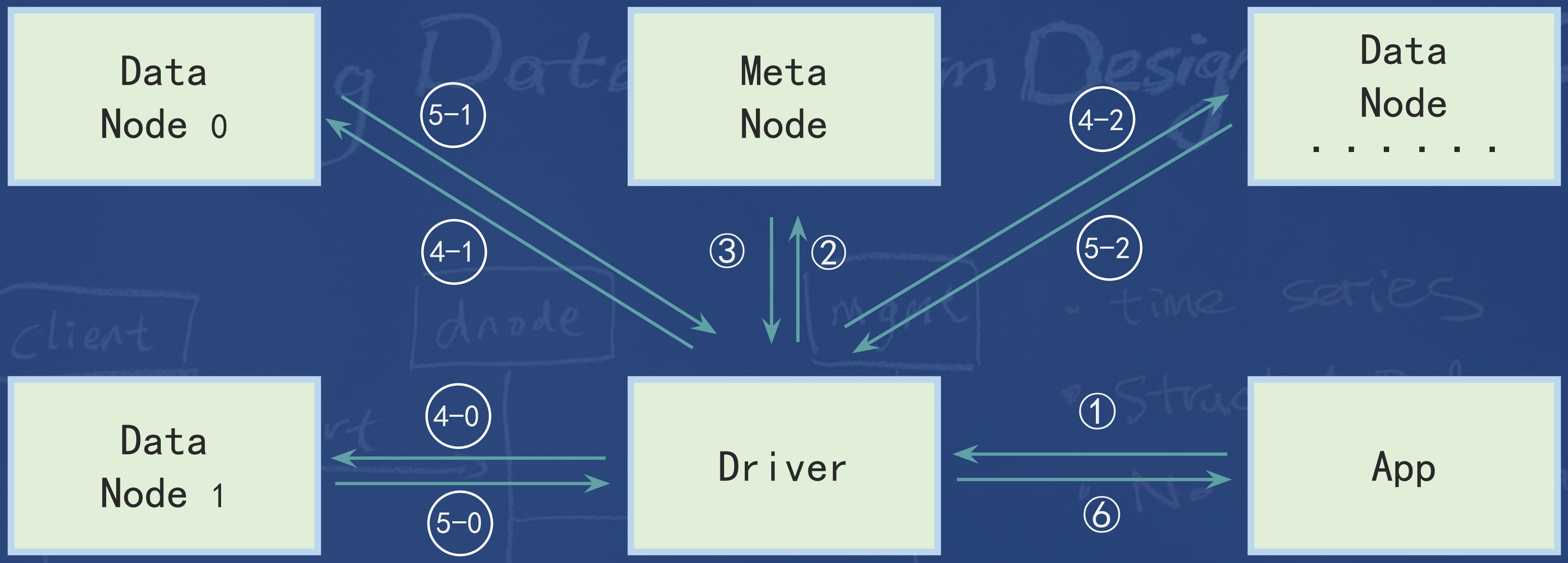
- Multiple copies & Master-Slave mechanism
- Writes can only be performed on the Master, but queries can be performed on the Slave node
- Supports both synchronous and asynchronous data replication between nodes
- When even number of copies, arbitrator is needed for solving the Split Brain problem
- For Mnode: Using synchronous replication, the entire system allows up to 3 Mnodes
- For Vnode: vnodes on different data nodes form a vnode group. The master-slave mechanism is adopted in the vgroup. The number of vnodes in the vgroup is the number of copies. Using asynchronous data to replicate by default

TDengine data writing process



TDengine

TDengine multi-node aggregation calculation process



TDengine

Other highlights of TDengine

Data aggregation on timeline (roll up)

In real-life scenarios, it is often necessary to aggregate data over a period of time. Such as down sampling, the sampling frequency is one per second, but in the end only the average value of one minute is needed. TDengine introduces the keyword interval to perform aggregation on the time axis. The aggregation of the timeline can be performed on a single table or on a group of tables that meet the label filter conditions.

Query the average value of the temperature recorded by the sensor t1 every five minutes

```
select avg(degree) from t1 interval(5m);
```

Query the average value of the temperature recorded by all sensors in California every five minutes

```
select avg(degree) from thermometer where loc like ‘CA%’ interval(5m);
```

TDengine

Stream Processing – Continuous Query

Currently supports Avg, Max, Min, Percentile, Sum, Count, Dev, First, Last, Diff, Scale, WAvg, Spread and other operations. Calculation can be made for a time period, and can be aggregated for a table or a group of tables that meet the label conditions.

Derived data calculated in real time can be written into a new table in real time to facilitate subsequent query operations. Derived data can also perform various aggregation calculations with other original data or other derived data to generate new data.

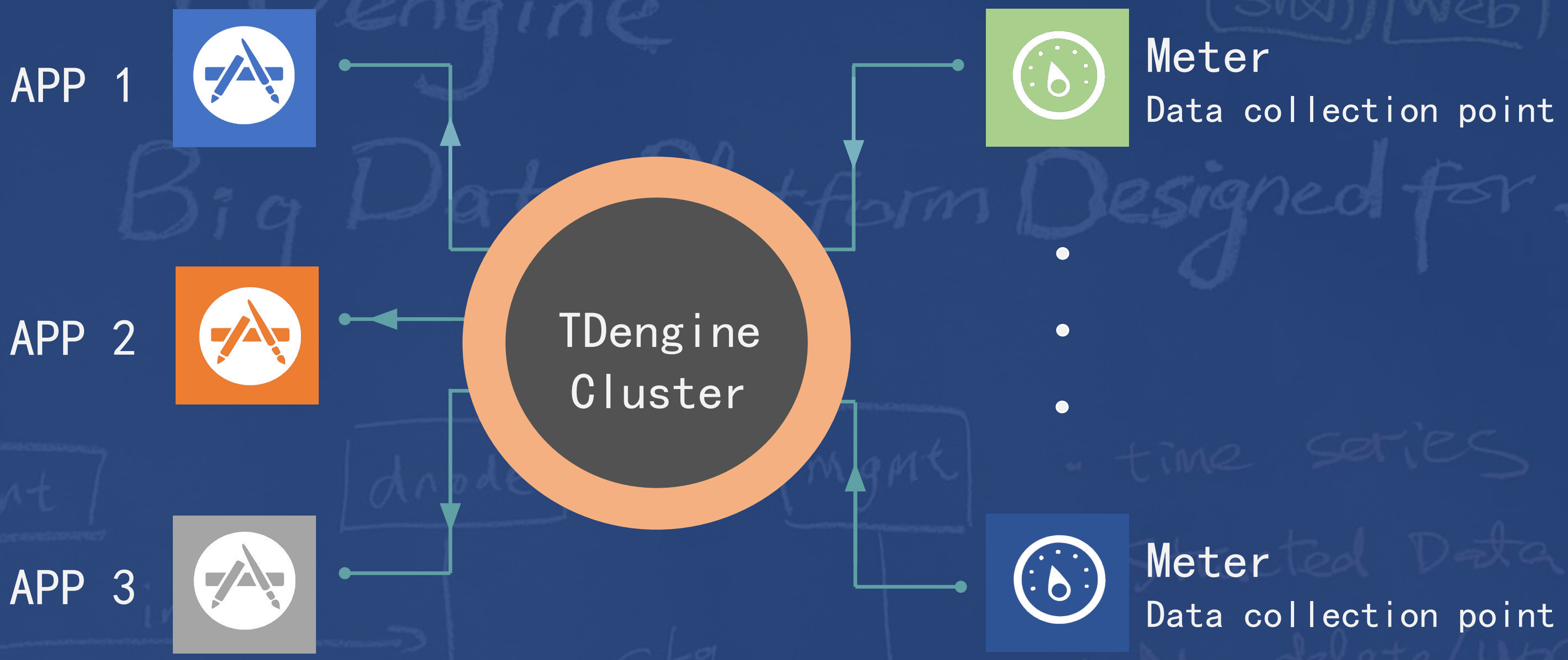
Calculate the average temperature of California for the past five minutes for every minute

```
select avg(degree) from thermometer where loc like ‘CA%’ interval(5m) sliding(1m);
```

Calculate the average temperature of the last 5 minutes in California every minute and write it into the new table d1

```
create table d1 as
select avg(degree) from thermometer where loc like‘CA%’ interval(5m) sliding(1m);
```

Data Subscription



- Similar to Kafka, the application can subscribe to the data stream, as long as the data is updated, the application will be notified in time
- When subscribing, the application only needs to specify the table name (it can be a super table) and start time, and can also specify filter conditions

TDengine

Caching

- The latest data point for a time-series is always saved in memory, so it can be retrieved quickly.
- SQL function last_row is provided.
- In some cases, redis is not needed anymore, the complexity and cost can be reduced.

www.github.com/taosdata/TDengine

TDengine Core are Fully Open Sourced

Single Node Ed.

2019.07.12

Clustering Ed.

2020.08.03

18,000+ Stars

4,300+ Forks

9,000+ Issues

Project	Open Sourced	Stars	Forks
InfluxDB	6 years ago	23,100	2,900
OpenTSDB	6 years ago	4,600	1,200
TimeScale	4years ago	12,700	665

TDengine

Jeff
Tao

- Email: jhtao@tdengine.com
- Twitter: twitter.com/jhtao
- LinkedIn: linkedin.com/in/jhtao/

- www.tdengine.com
- Discord: tinyurl.com/2p8j37aw