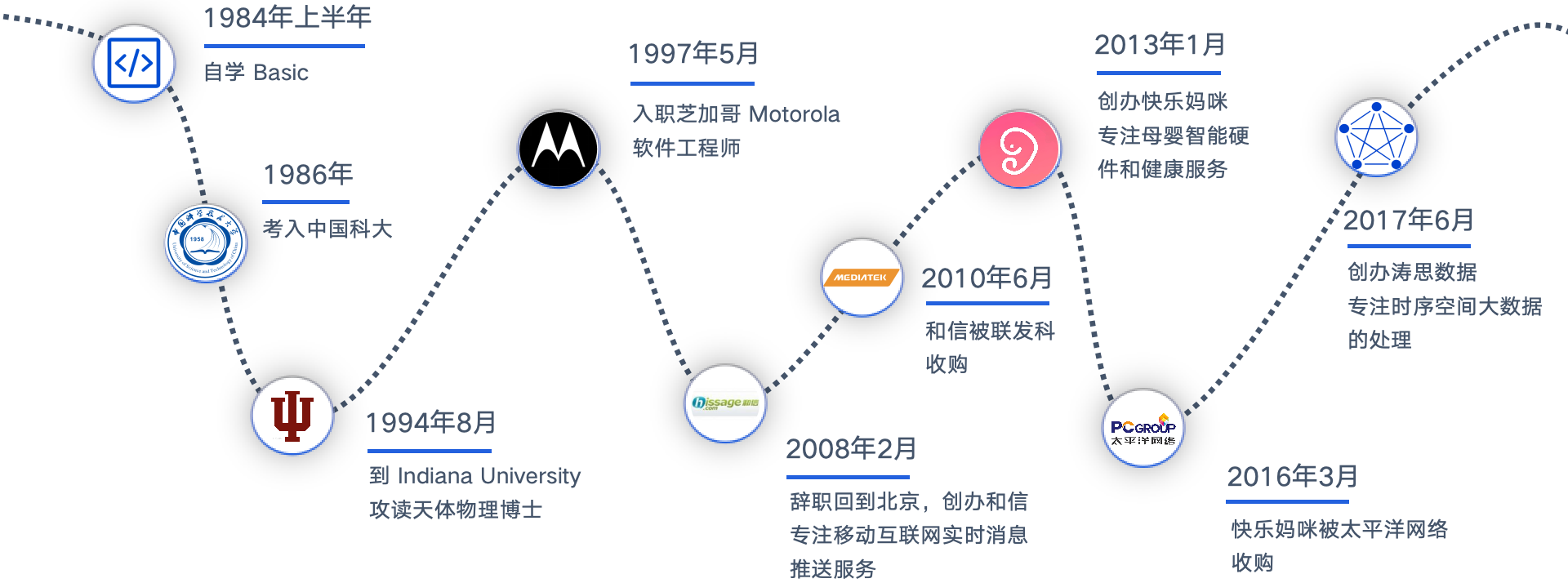
An Innovative Approach to Time Series Database Design

Jeff Tao

Founder & CEO，涛思数据/TDengine

November 2025

倒腾的历史


1984年上半年
自学 Basic
1986年
考入中国科大
1994年8月
到 Indiana University
攻读天体物理博士
1997年5月
入职芝加哥 Motorola
软件工程师
2008年2月
辞职回到北京，创办和信
专注移动互联网实时消息
推送服务
2010年6月
和信被联发科
收购
2013年1月
创办快乐妈咪
专注母婴智能硬
件和健康服务
2016年3月
快乐妈咪被太平洋网络
收购
2017年6月
创办涛思数据
专注时序空间大数据
的处理

What is Time-Series Data?



Energy



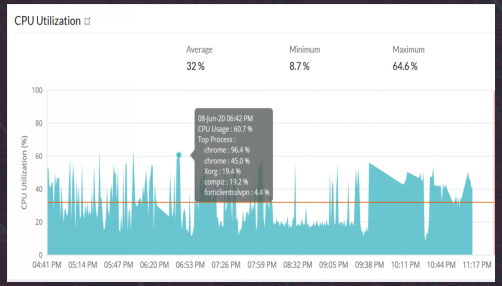
Transportation/Logistics



Manufacturing



Connected Vehicles



IT Monitoring



Financial

a sequence of data points indexed in time order

Characteristics of Time-Series Data

01 Timestamped data

02 Structured data

03 Data source is like a stream

04 Data rate is relatively stable

05 Transaction is not required

06 More write than read operations

07 Data is rarely deleted or updated

08 There is always a retention policy

09 Real-time data computing is desired

10 Query is always in the time and space range

Popular Time-Series Databases on the Market

InfluxDB	TimescaleDB	TDengine	QuestDB	Prometheus
OpenTSDB	VictoriaMetrics	Graphite	RRDtool	… …

TSDB Data Model: Tag Set Data Model

Every time series is uniquely identified by a metric name and a set of tags.

- For Prometheus or OpenTSDB, a time series for connected vehicles could be written like this:

```
gps_x { vin=”vin1”, brand=”tesla”, model=”s3”}
gps_y { vin=”vin1”, brand=”tesla”, model=”s3”}
gps_z { vin=”vin1”, brand=”tesla”, model=”s3”}
```

- For InfluxDB, a time series for connected vehicles could be written like this:

```
gps, vin=vin1, brand=tesla, model=s3
```

TSDB Data Model: Relational Data Model

Schema is always defined first, there is always a timestamp column, and each metric has a dedicated column and a data type.

TimescaleDB: one table to store the tags, one table to store the time series data.

```
• CREATE TABLE vehicles(vin SERIAL PRIMARY KEY, brand VARCHAR(20), model VARCHAR(20));
• CREATE TABLE vehicle_data(time TIMESTAMPTZ NOT NULL, vin SERIAL, x DOUBLE PRECISION, y
  DOUBLE PRECISION, z DOUBLE PRECISION, FOREIGN KEY (vin) REFERENCES vehicles (vin));
```

To query all data for the Tesla Model S3, the SQL statement would be:

```
• SELECT * FROM vehicle_data JOIN vehicles ON vehicle_data.vin=vehicles.vin WHERE
  brand=“tesla” AND model=”s3”;
```

TDengine Data Model

One Table for One Data Collection Point (Sensor)



Typical Scenario: Smart Power Meter

Data Characteristics:

- Each record has a timestamp.
- Data is structured, mainly numerical numbers.
- Structure is consistent 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.

Device ID	Timestamp	Metrics			Tags	
Device ID	Timestamp	Current	Voltage	Phase	Location	Type
d1001	1538548685000	3.13	220	0.31	CA.SanDiego	1
d1002	1538548685100	8.21	219	0.82	CA.SanFrancisco	2
d1001	1538548686000	3.11	219	0.35	CA.SanDiego	1
d1003	1538548683000	5.41	110	0.53	CA.LosAngeles	1
d1002	1538548686100	8.11	223	0.81	CA.SanFrancisco	2
d1002	1538548687130	8.15	215	0.85	CA.SanFrancisco	2
d1001	1538548687000	3.15	223	0.32	CA.SanDiego	1
d1003	1538548684000	5.51	112	0.54	CA.LosAngeles	1
d1003	1538548685000	5.60	109	0.53	CA.LosAngeles	1
d1002	1538548688100	8.19	218	0.87	CA.SanFrancisco	2
d1003	1538548686000	5.62	108	0.56	CA.LosAngeles	1
d1001	1538548688500	3.19	221	0.31	CA.SanDiego	1

One Table per Data Collection Point (Sensor)

Device ID d1001, location CA.SanDiego, type 1

Timestamp	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 d1003, location CA.SanFrancisco, type 2

Timestamp	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

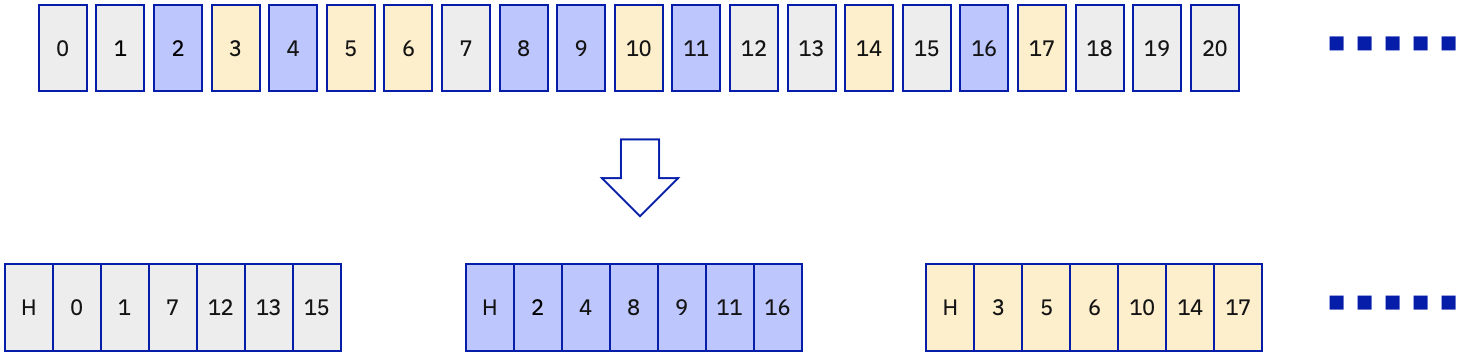
Device ID d1002, location CA.LosAngeles, type 2

Timestamp	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

Benefits of the design:

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

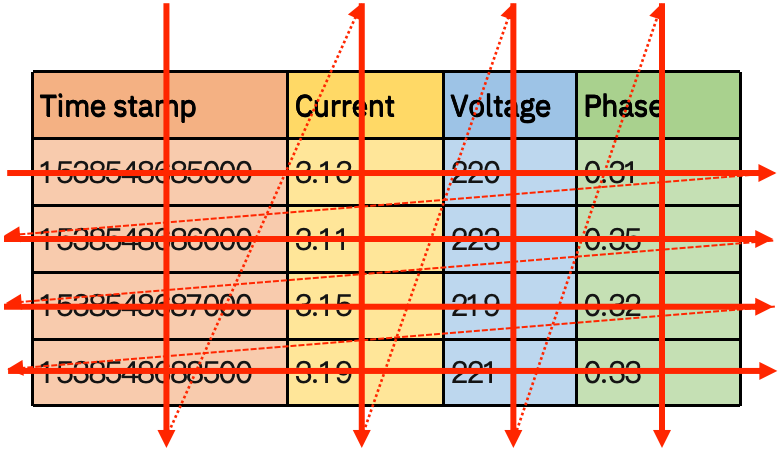
Table Storage: Block by Block, Each Contains Multiple Records



- 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 it can locate the data block right away.

Data in the Block Using Column-based Store

Column-based Store ✓



- **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

Row-based Store



TDengine Strategies:

01 One table for one data collection point (sensor)

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

03 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

01 Many data collection points for one type of measurement, e.g. there may be 30 million smart meters in a province.

02 The same type of data collection points often need to be aggregated.

03 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 Supertable:

Efficient Aggregation of Multiple Data Collection Points



Super Table: A Type of Data Collection Point

To describe a supertable, 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 supertable as a template, the schema of the table is the collected data schema of the supertable.
- Assign specific values to static labels.

A Supertable Example

Create a supertable for smart power meters: the collected data includes current, voltage and phase, and the label includes location and type.

```
CREATE STABLE 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);
```

Aggregation and Multi-dimensional Analysis by Supertable

Query the average voltage and maximum current of all smart meters in San Jose, CA:

```
SELECT AVG(voltage), MAX(current) FROM smeters WHERE loc = “CA.SanJose”;
```

Query the average voltage of the type 1 smart meters in CA:

```
SELECT AVG(voltage) FROM smeters WHERE type = 1 AND loc LIKE “CA%”;
```

- Supertables can be queried like ordinary tables, but you 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

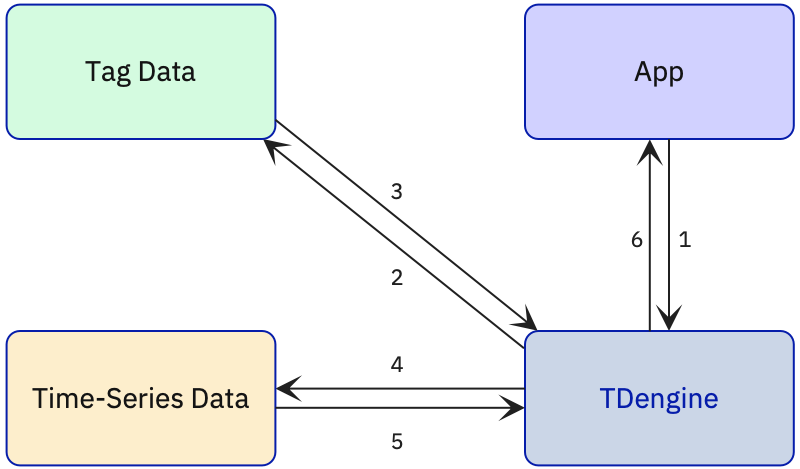
Strategies:

- 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 reduces storage space significantly.
- In multi-dimensional aggregation, first filtering the label to find the collection points that need to be aggregated can reduce the aggregated data set significantly.
- The total number of label records is equal to the number of collection points, so the amount is small enough to be stored in memory to improve query efficiency.

TDengine Aggregation Process



Workflow:

1. An application sends a query request to TDengine.
2. TDengine sends the tag filtering conditions to the tag processing module.
3. The tag query module returns the tables for the DCPs that meet the filtering conditions.
4. TDengine instructs the time-series processing module to aggregate the selected DCPs in the specified time range
5. TDengine receives the aggregated results.
6. TDengine returns the results to the application.

- The tag data is a dimension table and time-series data is a facts table.
- The supertable can be seen as a two-stage index: the first stage is the tag index, which filters DCPs, and the second stage is timestamp index, which filters time-series data by time range.

Advantages of TDengine Supertable:

- Significantly reduces label storage space
- Significantly improves the aggregation efficiency of data collection points
- Enables convenient and efficient multi-dimensional analysis

TDengine Virtual Table
Grouping DCPs into a Single Device



Virtual Table: Origins

Consider tables p1, p2, p3, and p4 with the following schema and data:

Timestamp	Current
0:00:01	3.22
0:00:04	4.22
0:00:05	3.19

Timestamp	Voltage
0:00:02	220
0:00:04	230
0:00:05	240

Timestamp	Temperature
0:00:04	32.5
0:00:05	34.3

Timestamp	Amplitude
0:00:04	1.0
0:00:05	0.9

The following result set is needed for a report. However, the multi-level JOIN query required to generate this is difficult to use and affects performance.

Timestamp	Current	Voltage	Temperature	Amplitude
0:00:01	3.22			
0:00:02		220		
0:00:03			32.5	
0:00:04	4.22	230		1.0
0:00:05	3.19	240	34.3	0.9

Virtual Table: Data Model

1. Example: In a power generation scenario collecting data over OPC, the operator uses this single-column model:

```
create table stb(ts timestamp, val double) tags(device varchar(20), type varchar(20), point int);
```

2. The device in this scenario generates four metrics – current, voltage, temperature, and amplitude – each of which corresponds to a subtable in TDengine:

```
create table p10 using stb tags (NULL, NULL, 10);
create table p11 using stb tags  (NULL, NULL, 11);
create table p12 using stb tags (NULL, NULL, 12);
create table p13 using stb tags (NULL, NULL, 13);
```

3. Once TDengine has started ingested this data, the DBA updates the data collection points' names and types by adding tags to the four subtables as follows:

```
create table p10 using stb tags  ('d1', 'voltage', 10);
create table p11 using stb tags  ('d1', 'current', 11);
create table p12 using stb tags ('d1', 'temperature', 12);
create table p13 using stb tags ('d1', amplitude, 13);
```

Virtual Table: Grouping DCPs into a Single Device

1. Create a virtual supertable for all generator devices:

```
create table devices (ts timestamp, current double, voltage double, ……) tags (device varchar(20)) virtual 1;
```

2. Create a virtual subtable for each generator:

```
create virtual table d1 (p10.val, p11.val, p12.val, p13.val) using devices tags('device1');
```

Or use following syntax to specify only certain columns:

```
create virtual table d1 (current from p10.val, voltage from p11.val) using devices tags('device1');
```

3. The query is then simplified into:

```
select ts, current, voltage, from d1
```

or

```
select ts, last(current), last(voltage), …… from d1 where ts between …… and ……interval(1h) fill(value, 0)
```

You can now easily query the current value of all metrics for all generators.

Timestamp	Device	Current	Voltage	Temperature	Phase	Amplitude
2024-03-0100:00:00	d1					
2024-03-0100:00:00	d2					
2024-03-0100:00:00	d3					

```
select ts,
       interp(current), interp(voltage) ……
from devices
partition by device
range('2024-03-01 00:00:00.000')
fill(linear)
```

Virtual Table: How Original Tables Are Combined

Consider tables t1, t2, and t2 with the following schema and data:

t1	
ts	value
0:00:01	3.22
0:00:04	4.22
0:00:05	3.19

t2	
ts	value
0:00:02	220
0:00:04	230
0:00:05	240

t3		
ts	value	value
0:00:03	32.5	
0:00:04		1.0
0:00:05	34.3	0.9

Projection query:

ts	c1	c2	c3	c4
0:00:01	3.22			
0:00:02		220		
0:00:03			32.5	
0:00:04	4.22	230		1.0
0:00:05	3.19	240	34.3	0.9

Query specific columns:

ts	c1	c2
0:00:01	3.22	
0:00:02		220
0:00:04	4.22	230
0:00:05	3.19	240

Interpolate the first column in PREV mode:

ts	c1	c2
0:00:01	3.22	
0:00:02	3.22	220
0:00:04	4.22	230
0:00:05	3.19	240

Virtual Table: One Table per Device

Features:

1. **Selecting and joining columns:** You can create virtual tables containing specified columns from multiple tables as needed and query them using the same syntax as normal tables.
2. **Timestamp alignment:** Data is aligned by timestamp, and if data exists in different tables at the same timestamp, it is combined into the same row. If some tables have no data at a particular timestamp, the corresponding columns contain a null value. (You can interpolate this data by using the padding function.)
3. **Dynamic updates:** Virtual tables are updated automatically and in real time when the data in the original tables changes.

Highlights:

1. **Simple:** One device equals one table, and complex joins and nested queries are no longer necessary.
2. **Fast:** Dynamic calculations reduce the need for duplicated data, and merge sort optimizes data alignment efficiency
3. **Real time:** Changes to original tables are reflected in virtual tables within milliseconds.
4. **Scalable:** You can add and remove columns at any time. Instead of designing a comprehensive schema at the beginning of a project, flexibly create the tables that your applications need when they need them.

Virtual tables make the “one table per device” design a reality

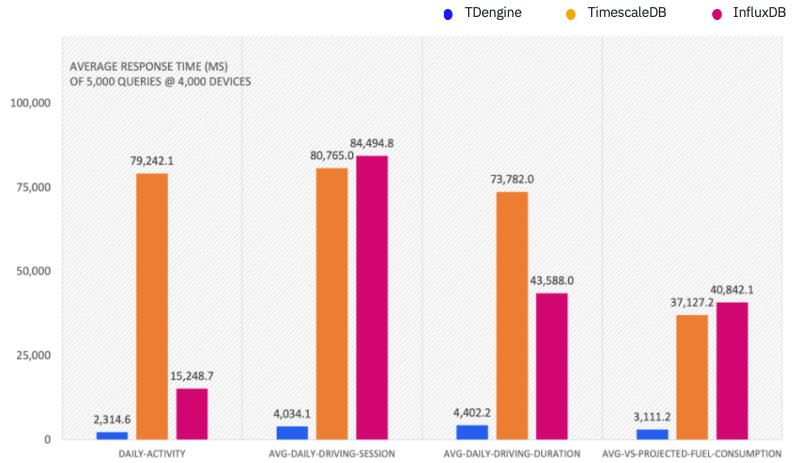
TSDB Benchmark Report



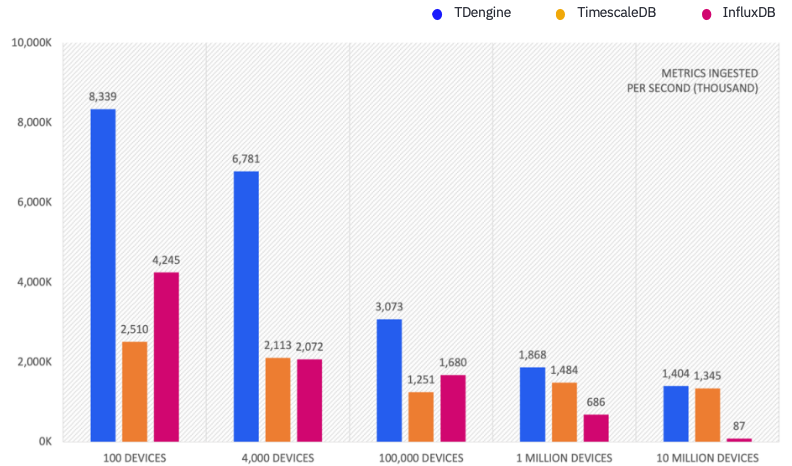
High Performance at Any Scale (1)

TDengine delivers 10x higher performance than general-purpose databases, and TSBS benchmark results show that it outperforms other time-series databases as well.

Query Performance



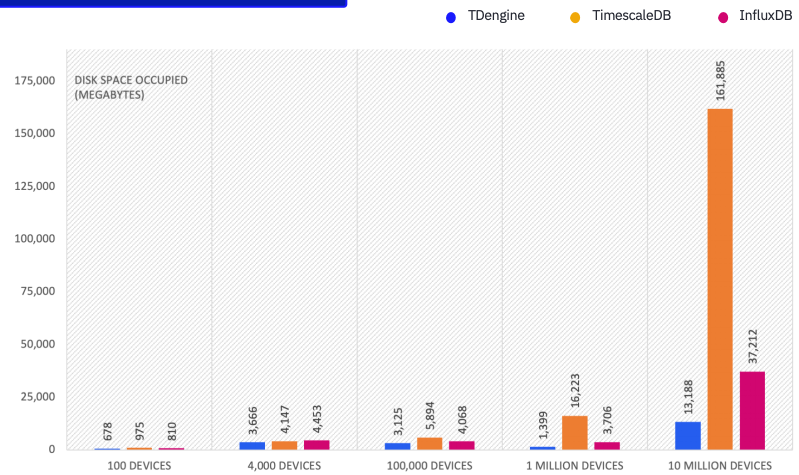
Data Ingestion Rate



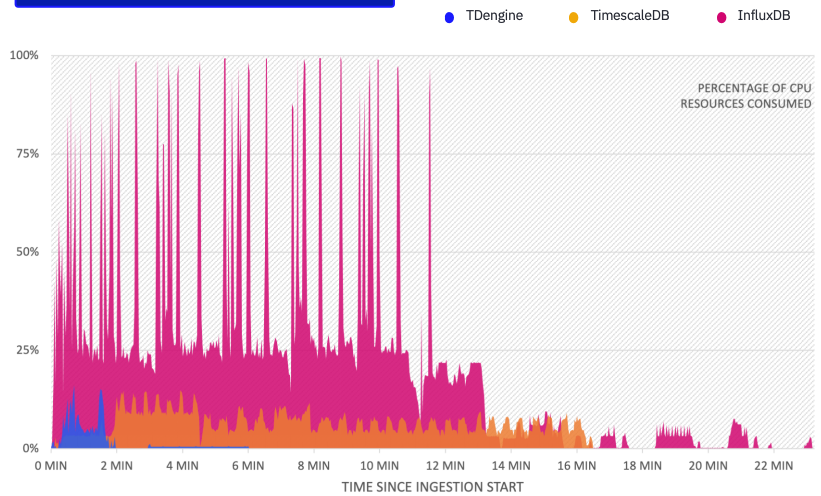
High Performance at Any Scale (2)

In addition to its superior ingestion and query performance, TDengine uses less disk space and compute resources than other time series database.

Disk Space Usage



Server CPU Usage



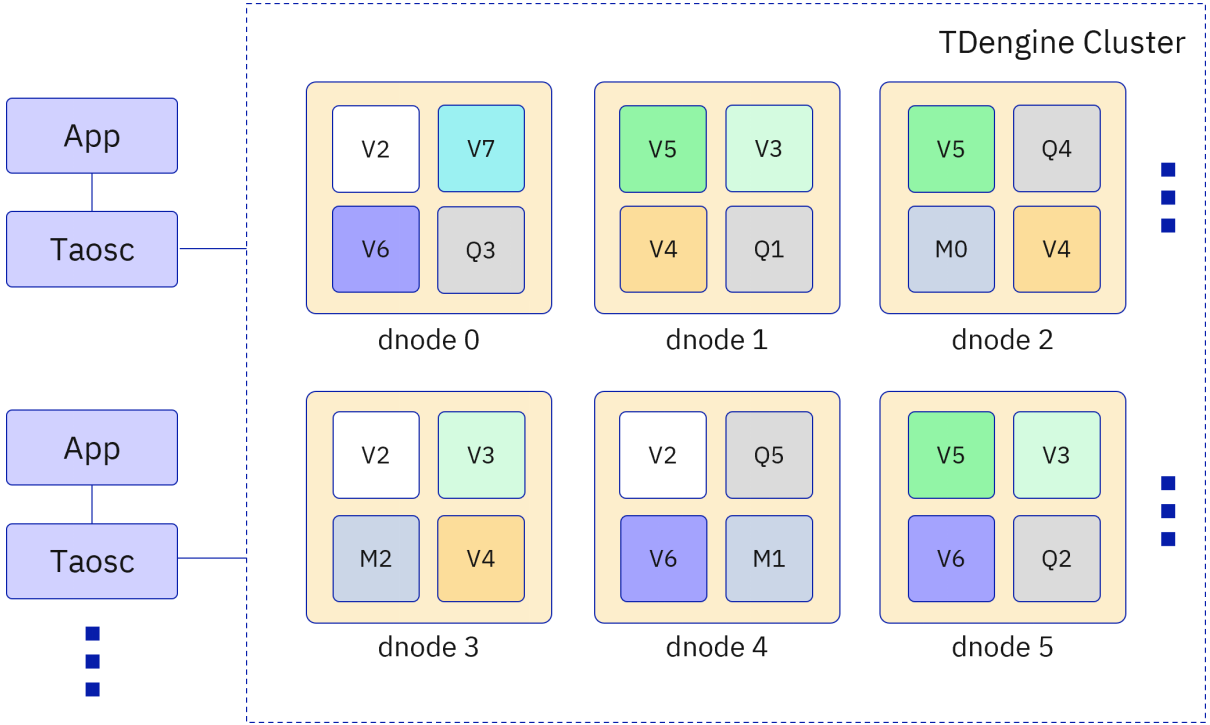
TDengine Scalability



Distributed Architecture

Logical units:

- Dnode
- Mnode
- Vnode
- Qnode
- Taosc



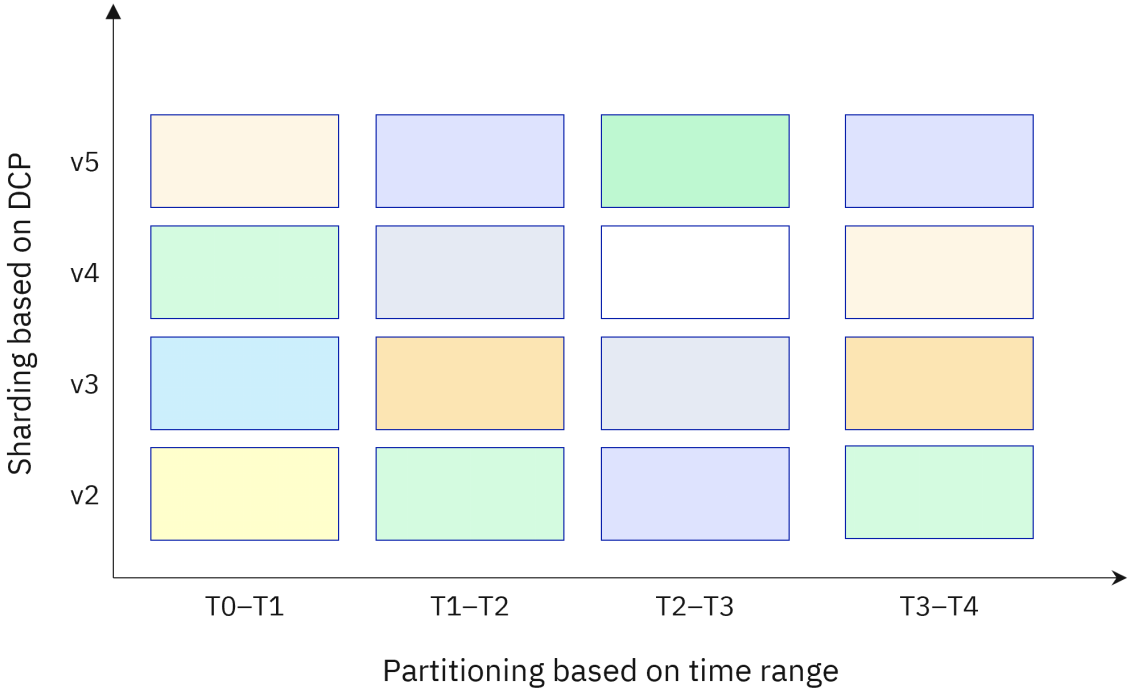
Processing & Storing Big Data

Sharding design:

- Sharded by DCP (vnode)
- 1 shard = one or more DCPs
- Distributed among hosts
- 1 DCP data = 1 shard
- Managed by mnode
- Load balancing on vnodes

Partitioning design:

- Data is partitioned by time



Distributed design for Meta Data

01 Mnode stores only node, user, and database meta information

02 The metadata for each table is stored in the vnode only

03 B+ tree is used for metadata storage

04 Consistent hash to find the vnode directly via table name

High Cardinality

OVER	WITH	SYSTEM STARTUP TIME WITHIN
1 billion	zero	one
TABLES	PERFORMANCE DETERIORATION	MINUTE

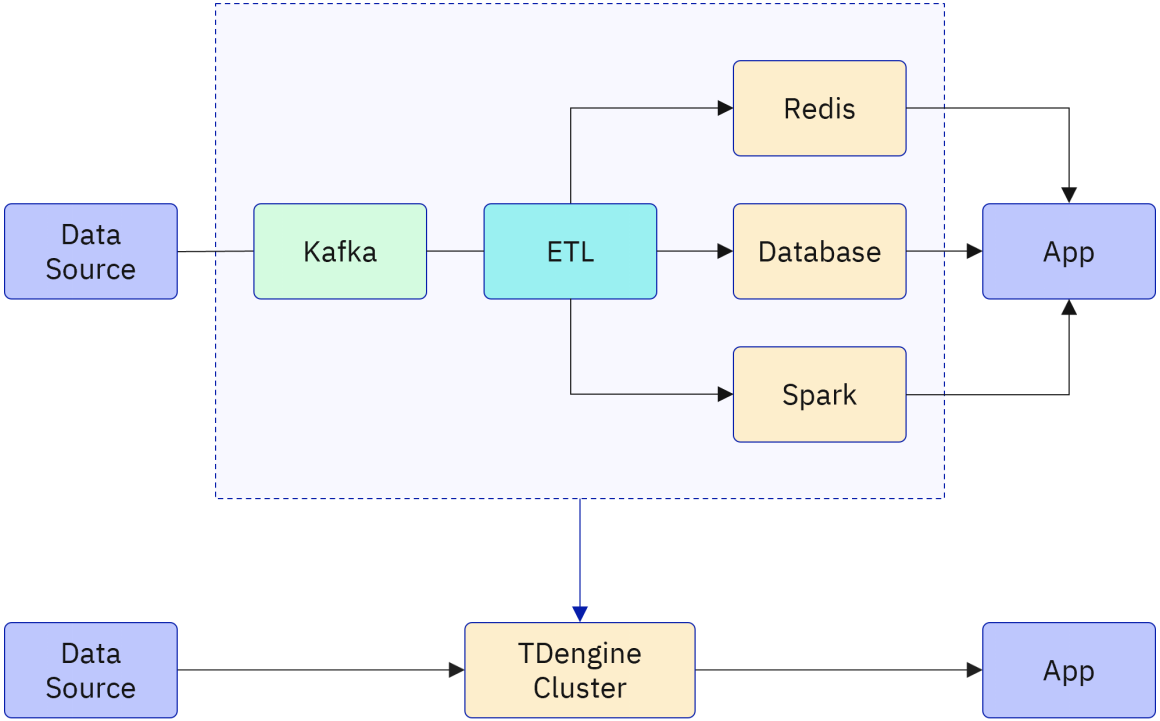
System Components

Time series database plus:

- Caching
- Data subscription
- Stream processing
- ETL

Greatly reduces:

- System complexity
- O&M costs



www.github.com/taosdata/TDengine

TDengine's core is fully open-sourced

Single Node	Clustered	Cloud Native	TDgpt
7/12/2019	8/3/2020	8/13/2022	3/26/2025

24,500+	5,000+	22,600+
Stars	Forks	Pull Requests

More than 926,000 installations in over 60 countries

* As of August 2024

User Base Distribution

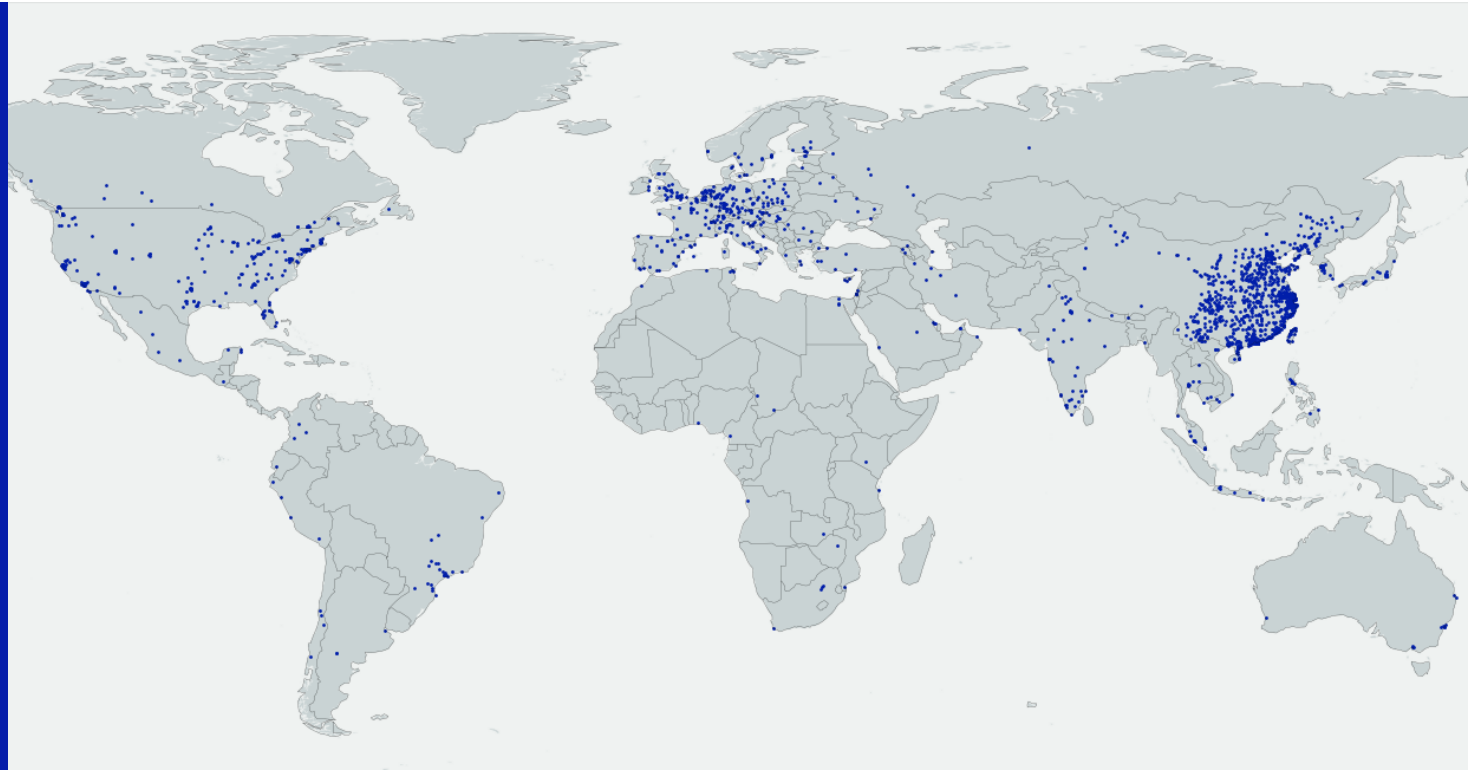
24,500+
Stars

5,000+
Forks

22,600+
Pull Requests

926,000+
Instances

500+
Daily Installations



* As of August 2024

TDengine Cloud: Fully Managed Cloud Service

Highlights:

- Fast, elastic, purpose-built platform for IoT time-series data.
- Worry-free operations with a fully managed cloud native solution.
- Enterprise ready with backup, multi-cloud replication, VPC peering and IP whitelisting.
- Free plan is available
- Run on AWS, Azure, GCP and Alibaba Cloud

cloud.tdengine.com



Engineering Challenges for TSDB

01 How to join a huge number of tables (>1000) efficiently?

02 How to explore a huge number of time series?

03 Data contextualization



陶建辉 / Jeff Tao

- Email: jhtao@taosdata.com
- Wechat: jeff-tao
- Weibo: 陶建辉-Jeff
- 微信公众号: 爱倒腾的程序员
- github.com/taosdata
- www.taosdata.com

北京涛思数据科技有限公司