

TS-Benchmark: a benchmark for time series databases

Yueguo CHEN

DBIIR Lab, Information School

Renmin University of China

Dec., 2018

TS-Benchmark: a benchmark for time series databases

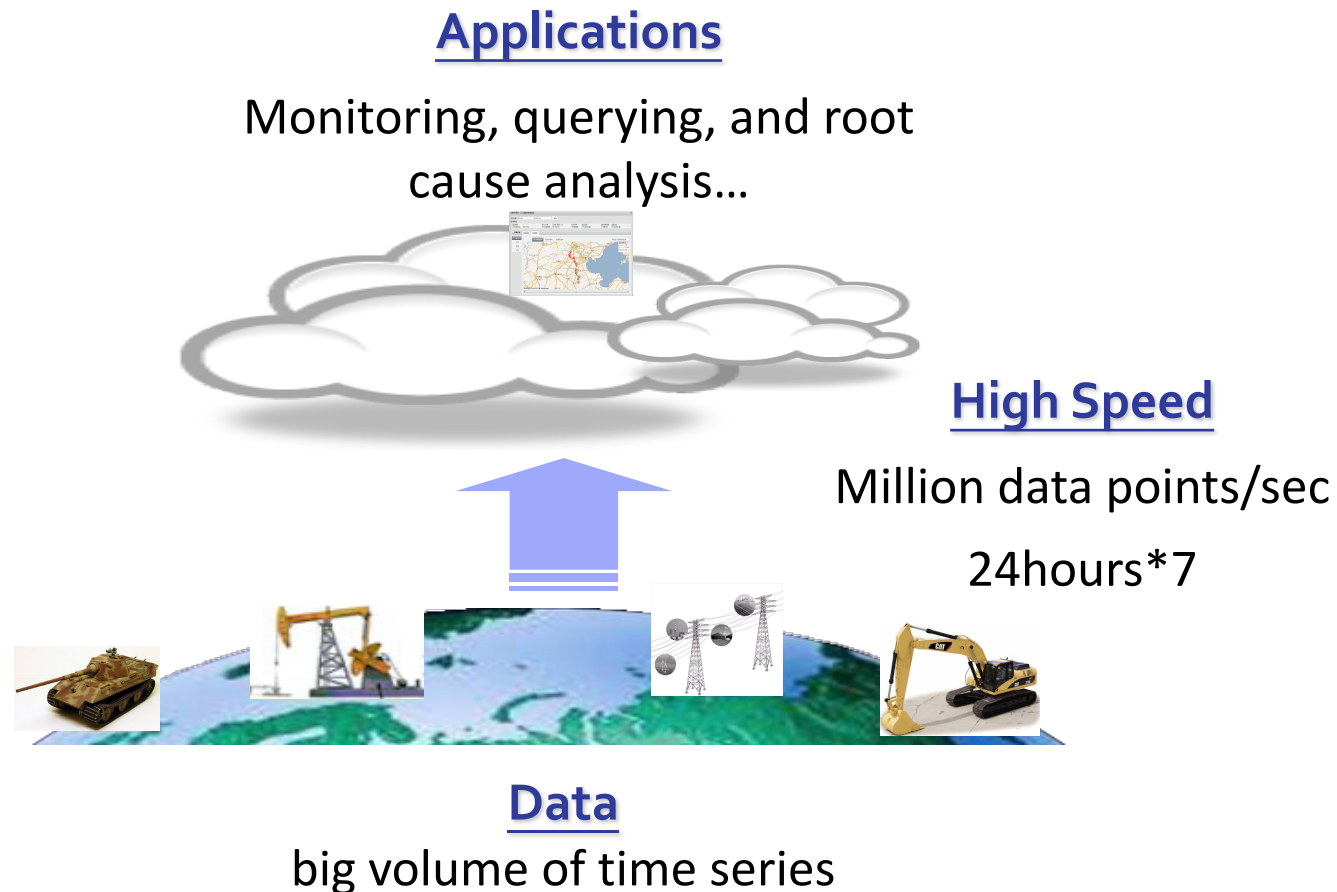
- **Agenda**

- Background
- Ideas of a new benchmark
- Application scenarios
- Data model and data generation
- Workload model and performance metrics
- Testing results of some representative time series database systems
- Q&A

TS-Benchmark: a benchmark for time series databases

• Background

- Big volume of time series is generated from IOT applications of various business domains, including
- Manufacturing
- Agriculture
- Military
- Smart city
- Logistics
- Sensors in scientific instruments
- Energy (turbine)
- ...

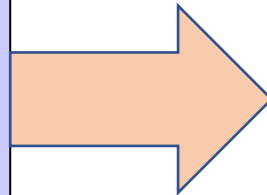


TS-Benchmark: a benchmark for time series databases

- Ideas of a new benchmark

Existing benchmarks

- **TPC-C**
 - test RDBMSs' transaction processing capability
- **YCSB**
 - Test data serving capability of NoSQL databases
- **InfluxDB-comparisons**
 - Model after server farm monitoring scenarios
 - Data is small, workload is simple
- **Stream Bench, RIoTBench, Linear Road**
 - Designed many years ago, not considering data processing requirements of time series of big data applications
 - Some benchmarks test stream systems, which process data in a streaming manner



Design and implement a new benchmark

- Designed Specifically for time series databases in IOT application scenarios
- Model the sensor data processing scenario of wind farms
- Testing of data appending, querying and combination of the two, i.e. when data is continuously injected into the target database, users can query the data in the same time
- Testing of data correction, including updating of error data, and inserting of missing data
- Long enough testing time to see how target databases' house keeping (data compaction etc.) affects its data appending and query performance

TS-Benchmark: a benchmark for time series databases

- **Application scenarios**

- A wind plant needs to monitor wind farms
- There are hundreds of turbines(devices) in a wind farm
- And there are many sensors attached to a device
- Every several seconds(7s), a snapshot of sensor data is sent to back end(cloud) for monitoring and persistency



TS-Benchmark: a benchmark for time series databases

- The **Data model** and **data generation**

- **Schema** : Each data point has attributes as follows
 - Wind farm id, device id, sensor id, time stamp and the reading of the sensor
- There are two types of sensors
 - Sensors sensing the environment : temperature, humidity, wind speed, wind direction etc.
 - Sensors sensing the turbines : pitch angle俯仰角, upwind angle迎风角, angular velocity角速度, electric voltage电压, electric current电流, installed power安装功率, nominal power额定功率, temperature inside, humidity inside, vibration frequency etc.
- Firstly, we train an ARIMA time series model with one year long wind power data provided by GoldWind company.
- Secondly, we use the trained ARIMA model to continuously generate wind data for each turbine. Based on wind data, and the mechanism to transform wind data to wind power we generate core sensor data for each turbine.
- Beside that, we also use ARIMA models to learn and generate other sensor readings, such as temperature and humidity etc.

TS-Benchmark: a benchmark for time series databases

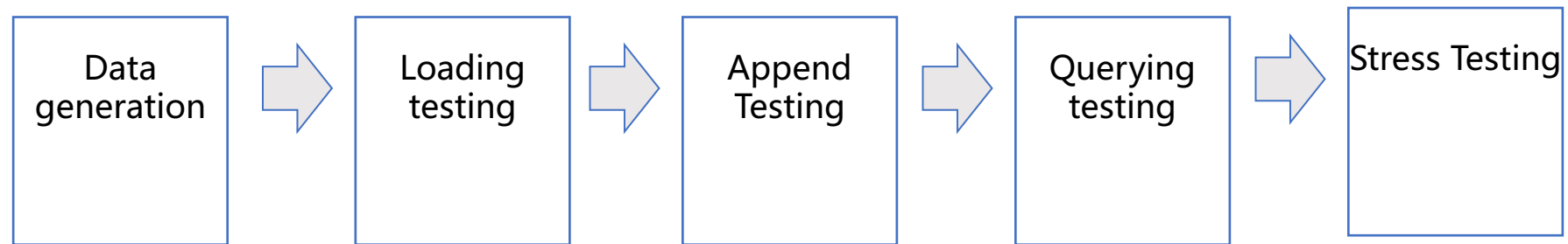
- **Workload** model and performance **metrics**
 - **Load** : generate a dataset of a windfarm for 7 days, and test loading performance (**points/sec**)
 - **Append** : increase number of devices (through number of client threads), test appending performance(**points/sec**)
 - **Query**(simple Read and Analysis) : run a query workload which includes simple window range query and aggregation query, against the target database (**requests/sec, average response time(us)**)
 - Simple query is for monitoring purpose
 - Aggregation query is for root cause analysis
 - The mix ratio is 90:10, think time between query is 50ms
 - **Stress Test** : includes two modes
 - **Mode 1**: Stress test of data injecting capability when there is a query workload (points/sec)
 - **Mode 2**: Stress test of query handling capability when there is a data injecting stream (requests/sec, average response time(us))

In future version of benchmark, we will add two tests, including updating of error data, and inserting of missing data

TS-Benchmark: a benchmark for time series databases

- **Testing procedure**

- The whole testing include 5 stages as follows



TS-Benchmark: a benchmark for time series databases

- **Target database systems** to be tested

- **InfluxDB**

- InfluxDB is a scalable time series database for recording metrics, events, and performing analytics. It uses the TSM (Time-Structured Merge Tree) data structure for data storage and enjoys a very high data appending speed.

- **lotDB**

- Developed by Tsinghua University of China. Built upon TsFile, which is a columnar file format supporting highly data compression, fast data fetch, and data updating.

- **TimescaleDB**

- A time series database built upon PostgreSQL, which is a mature relational database system. TimescaleDB has a complete SQL support.

- **Druid**

- an open-source column-oriented data store, designed for online analytical processing (OLAP) queries on event data.

- **OpenTSDB**

- A time series database built upon HBase, which is a scalable distributed NoSQL database running on Hadoop

TS-Benchmark: a benchmark for time series databases

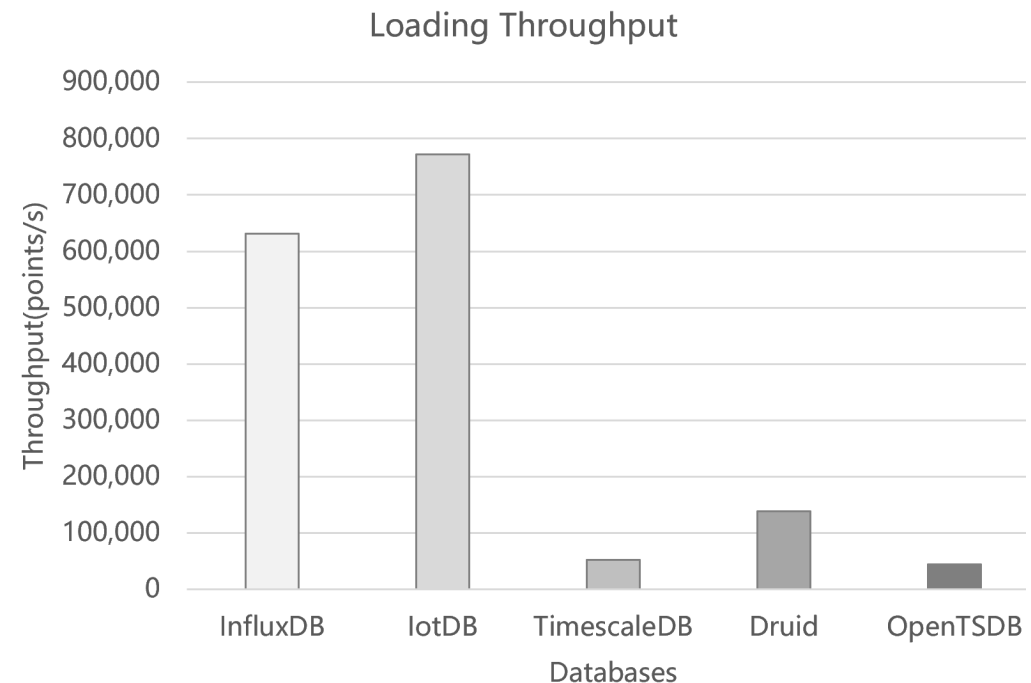
- Testing hardware and software environment
 - We are benchmarking single server version of InfluxDB, IotDB, TimescaleDB, Druid, and OpenTSDB
 - Hardware
 - a server with Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz(12cores, 2 thread each core, 24threads in total), 32GB of memory, 2TB of 7200rpm SATA Hard disk
 - Database server is bound to 4 cores(8 threads)
 - The client program runs on the same server to avoid network latency
 - Software
 - CentOS 7.4.1708
 - JDK 1.8

TS-Benchmark: a benchmark for time series databases

• Loading Test

- We generate data set of wind farm sensor data for 7 days, and store the data in a file.
 - 100 devices, 150 sensors/device, a snapshot of sensor data for every device per 7 seconds.
- The data file is loaded into memory first, and loaded into target databases to avoid I/O cost to see how fast the databases handle the data injection.
 - 10 threads are loading the data into databases, the database under test is running on 4 cores(8threads).
- [IotDB](#) achieves a highest throughput of 772,406 points/s, and [InfluxDB](#) achieves a comparable throughput of 631,227 points/s.
- [TimescaleDB](#), [Druid](#), and [OpenTSDB](#) are much inferior on loading, they achieve throughputs of 52,794p/s, 138,907p/s, and 44,027p/s respectively.

InfluxDB	IotDB	TimescaleDB	Druid	OpenTSDB
631,227	772,406	52,794	138,907	44,027

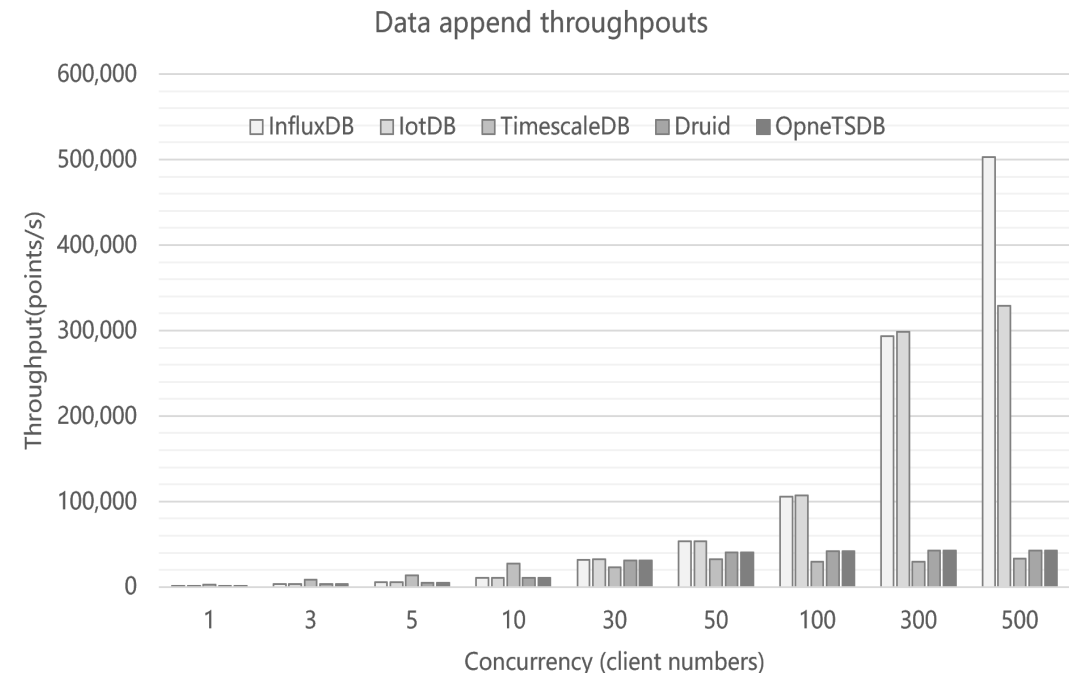


TS-Benchmark: a benchmark for time series databases

- **Appending Test – throughput**

- Each client thread is responsible for appending data of 50 devices. Each device has 150 sensors, and every 7 seconds a snapshot of sensor readings is sent.
- We gradually increase the number of clients to increase pressures and measure the throughputs and response times of target database.
- With the number of clients increasing, **IotDB** and **InfluxDB** achieve comparable throughputs when the **concurrency is low** (client number ≤ 300).
- When the number of clients further increase(**concurrency is high**), InfluxDB achieve much higher throughput than IotDB.
- **TimescaleDB**, **Druid**, and **OpenTSDB** are much inferior on appending than IotDB and InfluxDB.

	InfluxDB	IotDB	TimescaleDB	Druid	OpenTSDB
1	1,070	1,071	2751	1,070	1,070
3	3,214	3,214	8257	3,180	3,180
5	5,357	5,357	13284	5,288	5,288
10	10,683	10,709	27647	10,510	10,510
30	32,047	32,138	23254	30,810	30,810
50	53,146	53,466	32761	40,510	40,510
100	105,687	107,003	29250	41,710	41,710
300	293,063	298,808	29,631	42,566	42,566
500	502,900	329,172	33,268	42,841	42,841

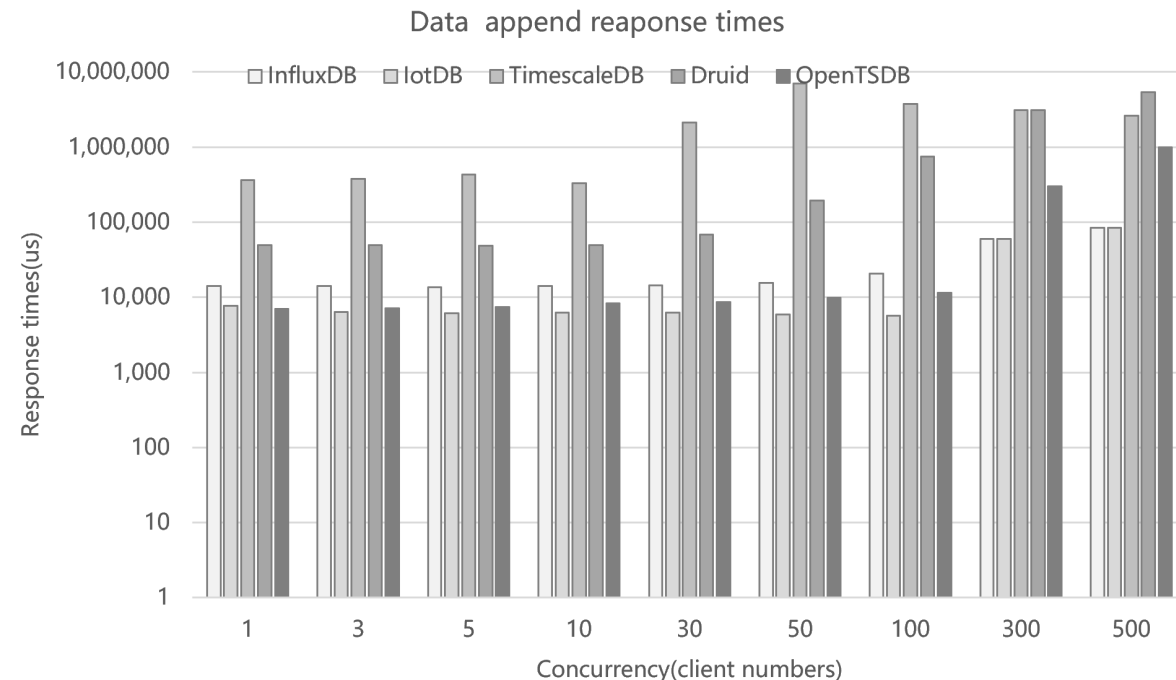


TS-Benchmark: a benchmark for time series databases

- **Appending Test – response times**

- We have measured max, min, average response times of each data points handled by target databases.
- When **concurrency is low**, **OpenTSDB**(client number ≤ 300) and **IotDB** achieve better response times than **InfluxDB**.
- However when **concurrency is high**(client number > 300), **InfluxDB** achieves better response time than **OpenTSDB**, and equal response time to **IotDB**.
- Although when the concurrency is low, **Druid** is better than **TimescaleDB** in terms response time.
- Both **Druid** and **TimescaleDB** get the worst response times.

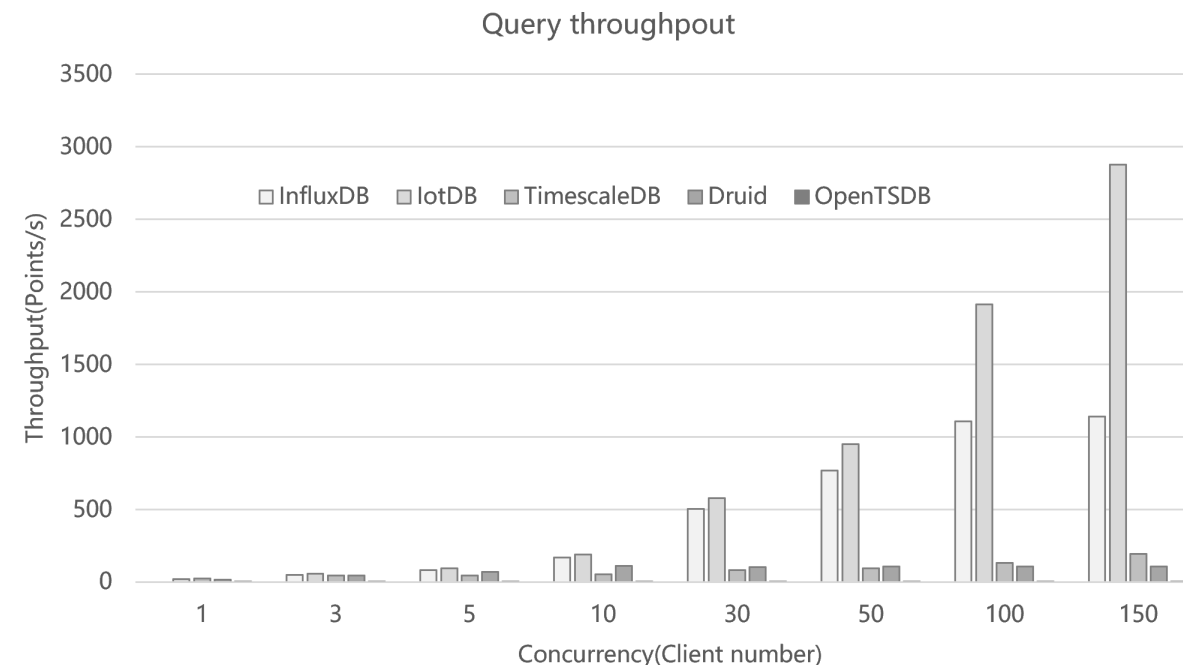
	InfluxDB	IotDB	TimescaleDB B	Druid	OpenTSDB
1	14,096	7,666	363347	49,935	7,000
3	14,054	6,412	380465	49,429	7,130
5	13,720	6,129	428345	48,281	7,422
10	14,156	6,282	332666	49,568	8,369
30	14,297	6,274	2131416	68,728	8,601
50	15,461	5,856	7015567	194,478	9,934
100	20,575	5,666	3765096	752,511	11,412
300	59,317	59,317	3,103,219	3,083,258	298,656
500	84,571	84,571	2,600,111	5,419,355	993,259



TS-Benchmark: a benchmark for time series databases

- **Query Test – throughput**
- We increase client numbers to increase query pressure on target databases, and measure query throughput and max, min, average response time of each query
 - Each client sent 20 request /sec, the think time is set to 50ms
 - The query workload include window query for monitoring purpose, and aggregation query for problem diagnosis purpose, the mix ratio is 9:1
- With increasing of the client number, **InfluxDB** and **IotDB** achieve much higher throughput than other database, IotDB achieves the highest throughput.
- **TimescaleDB, Druid** achieve less throughput.
- And **OpenTSDB** performs the worst.

	InfluxDB	lotDB	TimescaleDB	Druid	OpenTSDB
1	16	18	22	15	0.0489
3	49	56	43	44	0.0455
5	83	95	44	69	0.0388
10	167	191	55	111	0.0164
30	505	577	80	104	0.0127
50	770	948	93	108	0.0112
100	1,106	1,911	131	108	0.0084
150	1,138	2,878	193	106	0.0072

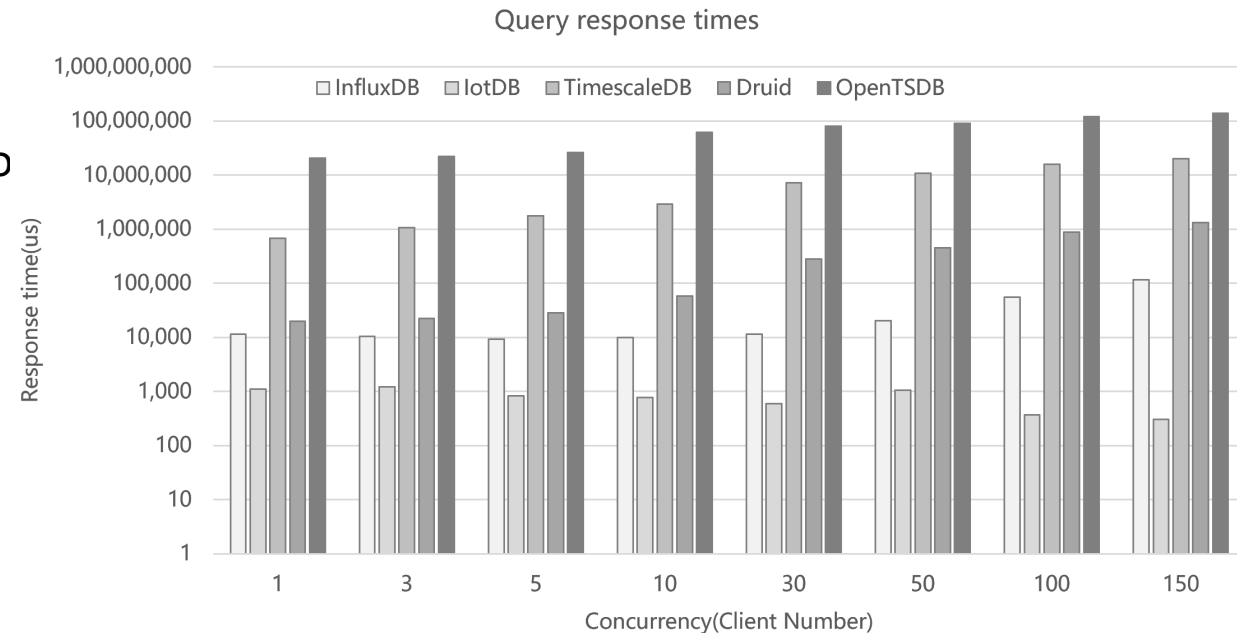


TS-Benchmark: a benchmark for time series databases

- **Query Test – response time**

- **IotDB** achieves better response time than other databases.
- **InfluxDB** is second to IotDB, followed by **Druid**.
- **TimescaleDB** and **OpenTSDB** are the most bad performing ones in terms of response time.
- And **OpenTSDB** is the slowest database to handle the query load.

	InfluxDB	lotDB	TimescaleDB	Druid	OpenTSDB
1	11,542	1,107	677126	19,806	20,440,966
3	10,345	1,205	1065172	22,341	21,975,824
5	9,348	822	1749476	28,546	25,747,924
10	9,953	776	2922655	57,336	60,951,978
30	11,402	595	7139071	279,311	78,965,533
50	20,080	1,051	10819995	448,233	89,587,663
100	54,772	373	15747781	882,152	118,731,148
150	115,777	306	19,877,354	1,332,498	139,723,308

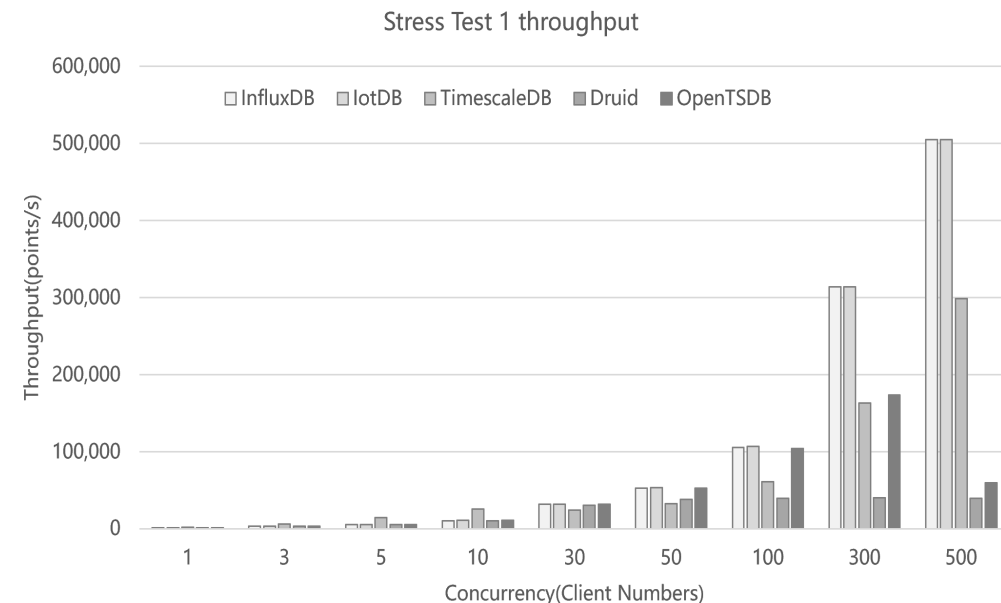


TS-Benchmark: a benchmark for time series databases

- **Stress Test 1 append (query)– throughput**

- We test how target databases handle appending when there is a background query stream.
 - The background steady query stream includes 50 clients, each client sends 2 queries per second, so there are 100 queries posed on target database per second.
 - In the mean time We gradually increase the number of append clients to increase appending pressures. Each client thread is responsible for appending data of 50 devices.
- With the number of clients increasing, **IotDB** and **InfluxDB** achieve comparable throughput, followed by **TimescaleDB**.
- Although **TimescaleDB** perform better than IotDB and InfluxDB when the concurrency is low.
- **Druid**, and **OpenTSDB** are much inferior on appending than IotDB and InfluxDB.

	InfluxDB	lotDB	TimescaleDB	Druid	OpenTSDB
1	1,071	1,071	1878	1,069	1,071
3	3,214	3,214	6352	3,179	3,214
5	5,305	5,357	14354	5,282	5,353
10	10,567	10,709	25878	10,522	10,682
30	32,065	32,100	24045	30,572	32,028
50	52,998	53,494	32825	37,735	52,588
100	105,620	106,973	60893	39,255	104,054
300	313,603	313,603	163,254	39,894	173,728
500	504,728	504,728	298,832	39,452	59,285

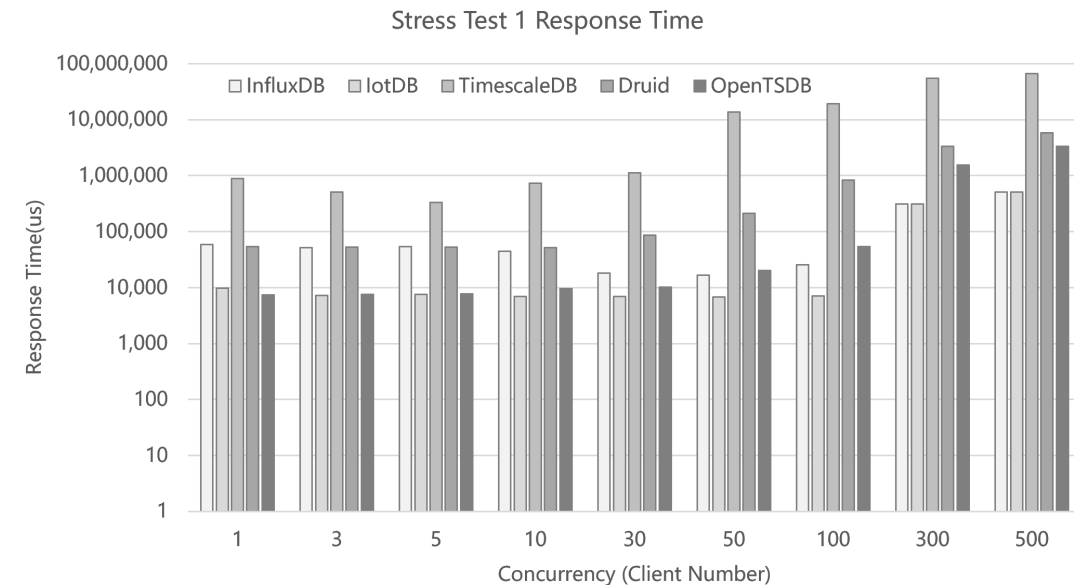


TS-Benchmark: a benchmark for time series databases

- **Stress Test 1 append (query)– response times**

- When **concurrency is low**, **OpenTSDB**(client number ≤ 300) and **IotDB** achieve better response times than **InfluxDB**.
- However when **concurrency is high**(client number > 300), **InfluxDB** achieves better response time than **OpenTSDB**, and equal response time to **IotDB**.
- Although when the concurrency is low, **Druid** is better than **TimescaleDB**.
- Both **Druid** and **TimescaleDB** get the worst response times.

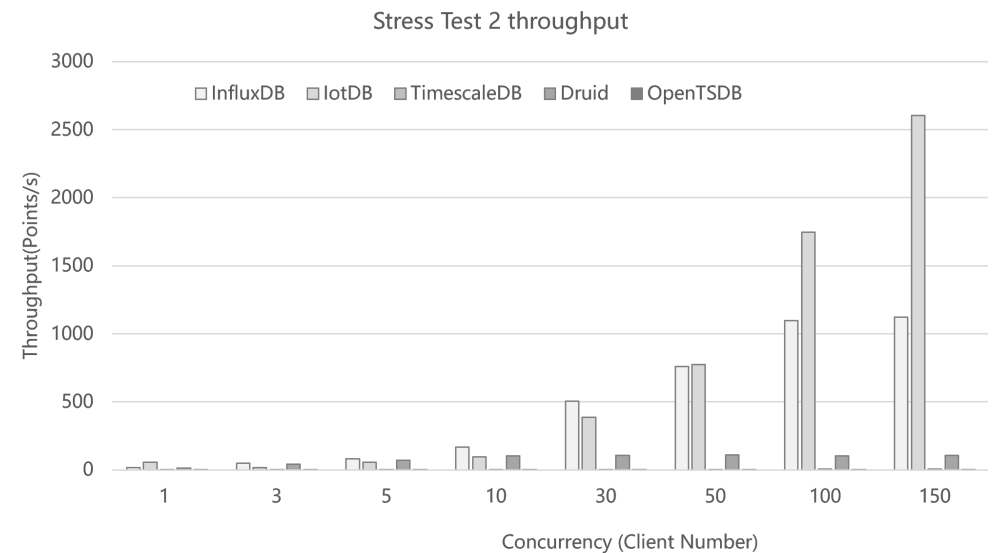
	InfluxDB	IotDB	TimescaleDB	Druid	OpenTSDB
1	59,348	9,789	888251	53,944	7,380
3	51,535	7,307	509346	53,154	7,506
5	53,999	7,496	329546	52,448	7,744
10	45,015	6,904	733725	52,078	9,502
30	18,298	7,015	1130014	86,610	10,153
50	16,527	6,784	13581278	211,984	20,034
100	25,472	7,052	19452755	831,797	54,402
300	313,603	313,603	54,873,464	3,317,888	1,564,813
500	504,728	504,728	67,354,656	5,871,429	3,372,715



TS-Benchmark: a benchmark for time series databases

- **Stress Test 2 query(append)– throughput**
- We test how target databases handle queries when there is a background appending stream.
 - The background steady appending stream includes 10 clients, each client is responsible for 10 devices, with 100 devices in total. Each client send a snapshot of sensor data per 7 seconds.
 - In the mean time We gradually increase the number of clients to increase querying pressures. Each client thread send 20 query requests per second with a think time of 50ms.
- With increasing of the client number, [InfluxDB](#) and [IotDB](#) achieve much higher throughput than other database, [IotDB](#) achieves the highest throughput.
- [TimescaleDB](#), [Druid](#) achieve less throughput.
- And [OpenTSDB](#) performs the worst.

	InfluxDB	lotDB	TimescaleD	Druid	OpenTSDB
1	16	57	1	14	0.0263
3	49	19	2	43	0.0260
5	83	57	2	70	0.0223
10	167	96	3	105	0.0203
30	504	387	3	107	0.0094
50	760	775	3	110	0.0064
100	1,097	1,747	5	105	0.0045
150	1123	2603	5	108	0.0043

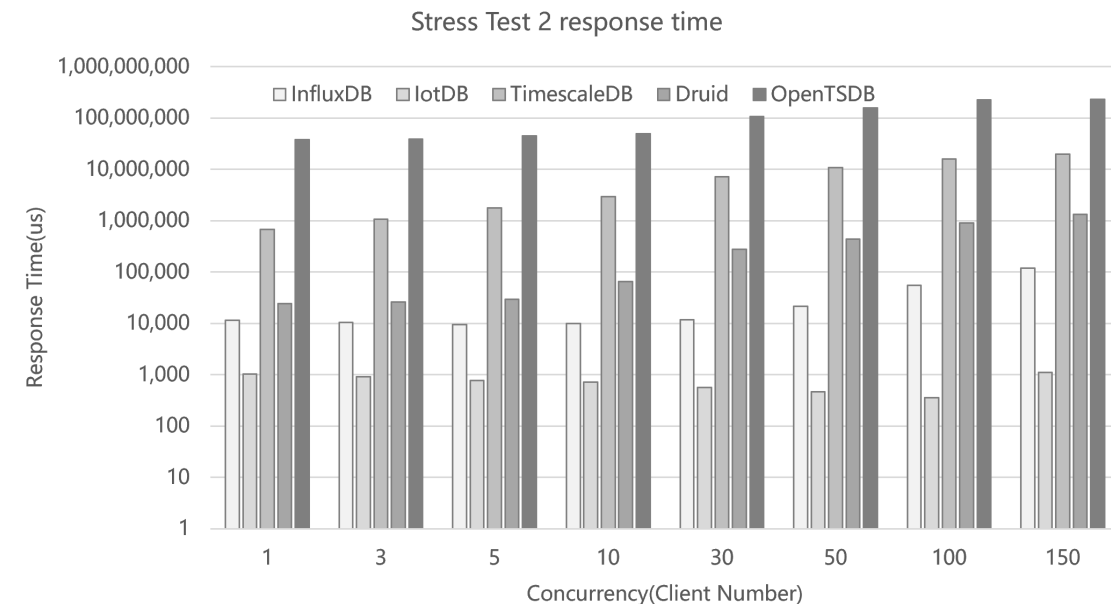


TS-Benchmark: a benchmark for time series databases

- **Stress Test 2 query(append)– response times**

- **IotDB** achieves better response time than other databases.
- **InfluxDB** is second to IotDB, followed by **Druid**.
- **TimescaleDB** and **OpenTSDB** are the most bad performing ones in terms of response time.
- And **OpenTSDB** is the slowest database to handle the query load.

	InfluxDB	IotDB	TimescaleDB	Druid	OpenTSDB
1	11,454	1,018	677126	24,178	38,019,957
3	10,375	924	1065172	25,924	38,434,704
5	9,441	765	1749476	29,395	44,796,683
10	9,854	715	2922655	64,618	49,191,116
30	11,790	560	7139071	275,062	106,852,689
50	21,577	464	10819995	437,607	156,009,483
100	55,122	355	15747781	908,555	222,922,643
150	119,461	1119	19871354	1,312,020	231,635,392



TS-Benchmark: a benchmark for time series databases

- Preliminary conclusions
 - Specifically designed **storage engine for time series** benefit data injection and appending greatly
 - The result of **stress test 1 append (query)** is consistent with the result of **append**, when the background query workload is light, it is interesting to see how databases digest data when query workload is heavy.
 - The result of **Stress Test 2 query (append)** is consistent with the result of **query**, when the background append pressure is light, it is interesting to see how databases handles query when the append pressure is heavy.
 - Why OpenTSDB is so slow (**query**) need more analysis. (frequent compaction?)

Thanks for your attention!
chenyueguo@ruc.edu.cn
Q&A

