TDengine Testing Report

目录

为帮助用户了解 TDengine 的指标，特将它与 MySQL, Cassandra, InfluxDB, OpenTSDB, ClickHouse 做了读写性能对比测试。同时，对 TDengine 单机的其他指标以及集群的水平扩展性做了测试。

1 测试用数据说明

测试采用的数据模型来源于车辆的移动定位数据，每行记录包含 5 个字段，分别是：车辆信息（12 bytes 字符串）、时间戳（8 bytes 长整型）、经度（4 bytes 单精度浮点数）、纬度（4 bytes 单精度浮点数）、车辆行驶方向（4 bytes 整数），数据长度共计 32bytes。数据样例如下：

```
B11719B11719,2015-06-11 01:01:08.1,116.36566,39.85797,0
B11212B11212,2015-06-11 01:01:08.2,116.36803,39.85196,0
BU2993BU2993,2015-06-11 01:01:08.3,116.43937,39.90597,0
B11336B11336,2015-06-11 01:01:08.4,116.24511,39.98002,0
BR4992BR4992,2015-06-11 01:01:08.5,116.40895,39.97141,126
B11520B11520,2015-06-11 01:01:08.6,116.364851,39.839113,0
B11520B11520,2015-06-11 01:01:08.7,116.364851,39.839113,0
B11520B11520,2015-06-11 01:01:08.8,116.364851,39.839113,0
B13016B13016,2015-06-11 01:01:08.9,116.422056,39.882576,0
B12961B12961,2015-06-11 01:01:08.10,116.367883,39.878393,0
B12961B12961,2015-06-11 01:01:08.11,116.367883,39.878393,0
```

图 1 数据示例

图中的数据时间戳在测试中进行了调整，进入到系统中的时间戳没有采用如图所示的非严格的 RFC3339 格式，而是使用与 1970-01-01 00:00:00(UTC)时间的差（精确到毫秒）的时间戳表示方式（8bytes 的长整型）。为便于测试，实际数据均由计算机随机生成并按照上述格式和数据库语法格式写入各数据库 。

主要测评过程使用的 SQL 建表语句如下：

```
create table tablename (ts timestamp, idtag binary(12), lat float, lon float, direction int);
```

2 统计结果说明

对于任何一个测试，每个操作过程重复运行 5 次，且最终的统计结果是 5 次测试结果的算数平均值。

每次测试进行之前均重启数据库服务，避免缓存对于之前执行获得的结果产生影响。

本文中所说的延迟(latency)是指全部完成该操作所耗费的时间，例如针对多条记录的查询操作，其时间延迟是指完全将结果获取到客户端后，相对于请求发出时间之间的间隔。 吞吐量(throughput)是单位时间内完整完成该操作的数量。

3 TDengine 与其他数据库单节点对比测试

3.1 测试环境及步骤说明

所有数据库的对比测试在同一台八核亚马逊云服务器上进行，该服务器的详细配置如下：

Instance type: c4.2xlarge
Instance family: Compute optimized
vCPUs: 8
Memory (GiB): 15
Instance Storage (GB): EBS only
Network Performance: High
Volume Type: 64G General Purpose SSD (GP2)
IOPS: 100/3000
AMI ID: Ubuntu-16.04 (ami-3532e258)

本测试将 TDengine 与 MySQL, Cassandra, InfluxDB, OpenTSDB 等数据库进行了对比。所测试的 InfluxDB 版本为 1.4.3; OpenTSDB 则由 github 上最新代码编译而来; MySQL 版本为 5.7.19-0ubuntu0.16.04.1; Cassandra 版本为 3.6; ClickHouse 版本为 1.1.54388，而 TDengine 的测试版本为 1.1.0。

在测试时，TDengine, MySQL 以及 Cassandra 采用各自的 C 语言同步接口。InfluxDB 则采用其提供的 golang 客户端接口且 golang 版本为 1.6.2。由于 OpenTSDB 只给出了 HTTP 接口，在测试中我们采用 C 语言和第三方 libcurl 库来编写测试程序。对于 ClickHouse，我们采用了 ClickHouse 官方提供的 JDBC 接口。

在本次测试中，TDengine、InfluxDB 和 MySQL 都采用默认设置，Cassandra 则只修改参数 read_request_timeout_in_ms 为 6000000 以避免读取时间过长而失败的情况。而 OpenTSDB 的设置项则较为复杂，以保证 OpenTSDB 的性能：除了文件目录和端口的设置以外，我们也打开了自动创建 metrics 的开关（tsd.core.auto_create_metrics = true），允许多条记录的块插入（tsd.http.request.enable_chunked = true）并设置块的最大容量为 30K（tsd.http.request.max_chunk = 30000）。除此之外，OpenTSDB 设置的选项还包括 tsd.core.uid.random_metrics = true 和 tsd.storage.enable_compaction = false 等。而且 OpenTSDB 在写入时只能插入单值数据，因此对除时间戳以外的字段配以不同的标签作为一条记录进行测试。为了保证测试的流畅性，我们将 clickhouse_server 端的配置 keep_alive_timeout 从 3 调整为 3000000。

为最大程度发挥 MySQL 性能，在 MySQL 整个写入过程中不进行数据提交，最后写入完成，才进行 commit 操作。

TDengine 提供性能更加优异的异步接口，但 MySQL, InfluxDB, OpenTSDB,

Cassandra, ClickHouse 都不提供。为保证公平，本测试中，TDengine 只使用其同步接口。

为避免网络延迟的影响，应用与数据库均在同一台服务器上运行。为更进一步控制变量，本测试确保在对每个数据库进行测试时只有该数据库运行且数据库所占用的 CPU 资源被限制在一个核上，因此数据库所能占用的 CPU 资源最高可达 100%。客户端应用可以启动多个实例并被限制在其他 7 个核上（总共 8 核）。需要声明的是，由于 OpenTSDB 需要 HBase 服务，因此在对 OpenTSDB 进行测试时，OpenTSDB 以及 HBase 都被限制在一个核上运行。

测试中，OpenTSDB 展现出不稳定的一面，尤其是在多客户端连接的情况下。在测试过程中发现，当有三个或三个以上的客户端连接 OpenTSDB 时，OpenTSDB 经常会出现不工作的情况，甚至崩溃。为了保证测试的准确性，本测试只对 OpenTSDB 在 1 和 2 各客户端连接的情况下的性能进行了测试和分析

本测试测试了各个数据库在不同客户端连接数同时写入和读取的表现，且每种情况均进行 5 次测试，最终结果为 5 次测试结果的平均值。同时为了避免测试与测试之间的干扰，每次测试结束后，都会删除数据，重启数据库服务并且等待 5 分钟左右。

3.2 写入性能对比

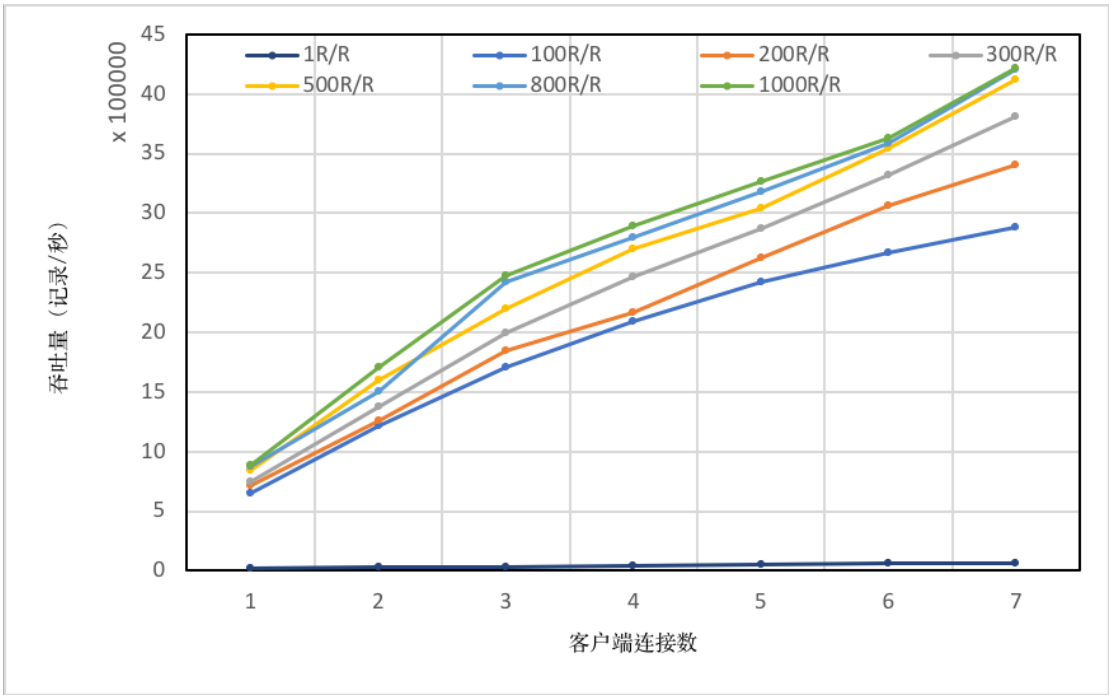
数据库的一个写入请求可以包含一条或多条记录。总的而言，一次请求里包含的记录条数越多，写入性能就会相应提升。在以下测试中, R/R 表示 Records/Request (一请求中记录条数), R/B 表示 Records/Batch（批处理中的记录数），R/R 与 R/B 没有本质区别，只是术语不同。

同时，一个数据库可以支持多个客户端链接，链接数增加，其系统总的插入通吐量会相应增加。因此测试中，对于每一个数据库，都会测试一个客户端和多个客户端连接的情况。

3.2.1 TDengine 结果

此测试中，按照每次请求包含 100，200，300，500，800，1000 条记录各进行了测试，同时也测试了不同客户端连接数的数据。具体结果如下：

条数	1 client	2 clients	3 clients	4 clients	5 clients	6 clients	7 clients
1	15,423	26,354	35,202	43,881	51,029	57,677	62,893
100	650,067	1,220,082	1,711,284	2,093,300	2,418,551	2,669,339	2,881,982
200	708,613	1,260,036	1,840,844	2,168,322	2,624,607	3,062,971	3,403,106
300	745,956	1,373,426	1,997,378	2,465,129	2,869,752	3,317,376	3,815,635
500	840,398	1,596,034	2,197,933	2,697,447	3,038,449	3,544,252	4,117,354
800	874,901	1,502,599	2,426,118	2,791,263	3,186,201	3,590,730	4,211,332
1000	886,653	1,708,037	2,478,045	2,896,079	3,267,590	3,630,836	4,213,323

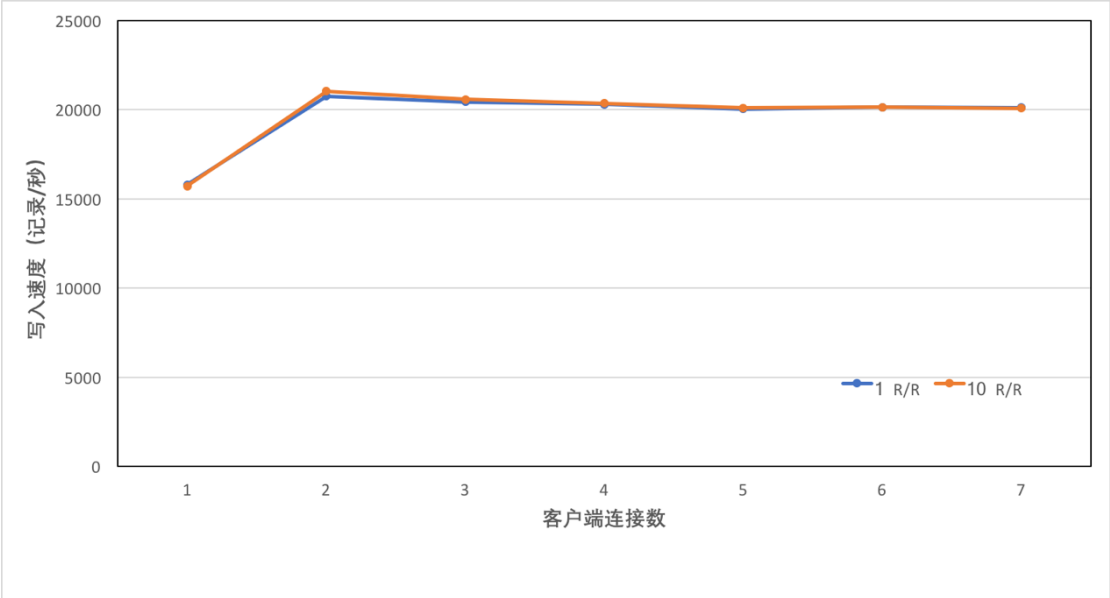


上述数据清楚表明，TDengine 的写入速度随客户端连接数的增加近乎线性增加，而且随单请求中记录数目的增加而增加。在七个客户端下，一个请求，写入 1 条数据时，TDengine 就有超过 60,000 记录/秒的写入速度。在一个写入请求 1000 条记录时，在 7 个客户端同时写入时的情况下，TDengine 有高达 4,213,323 记录/秒的写入速度。同时，从图 21 中可以看出，这个速度并非 TDengine 写入速度的上限，该速度还会随客户端数目的增加而增加。

3.2.2 MySQL 批量写入结果

测试中发现，MySQL 的写入性能只与 commit 操作有关，批量写入并不能有效提升其写入效率，因此仅仅测试了 MySQL 一行记录写入和 10 行记录批量写入的数据。结果如下：

条数	1 client	2 clients	3 clients	4 clients	5 clients	6 clients	7 clients
1	15,781	20,755	20,438	20,296	20,040	20,130	20,111
10	15,701	21,020	20,571	20,349	20,082	20,138	20,064

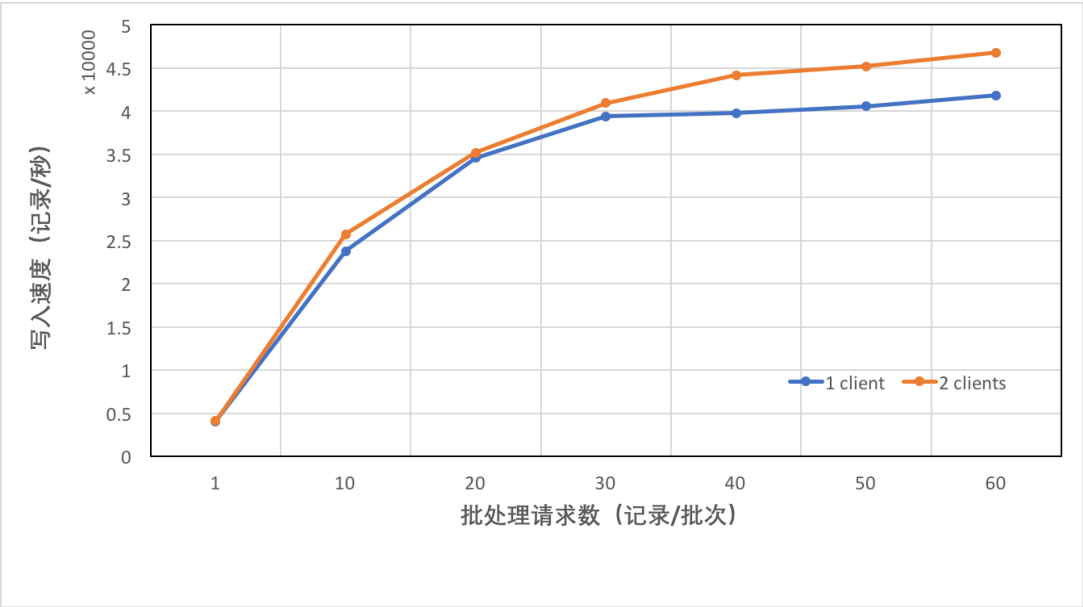


从上述数据可以看出，MySQL 在单客户端单条记录写入时，具有不错的表现。但是其写入性能并不会由于客户端链接数量的增加或单请求中记录条数的增加而显著改变。

3.2.3 Cassandra 结果

在测试中发现，Cassandra 的写入吞吐量与客户端连接数没有太大关系，而与批处理时单请求中的记录数相关。批请求中记录数目越大，其吞吐量也越大。由于客户端连接数对 Cassandra 的吞吐量影响较小，本测试只给出 Cassandra 在单客户端和双客户端连接情况下的测试结果。测试数据如下：

	1 R/B	10 R/B	20 R/B	30 R/B	40 R/B	50 R/B	60 R/B
1 client	4,064	23,808	34,558	39,411	39,760	40,564	41,802
2 clients	4,091	25,731	35,201	40,959	44,174	45,205	46,782

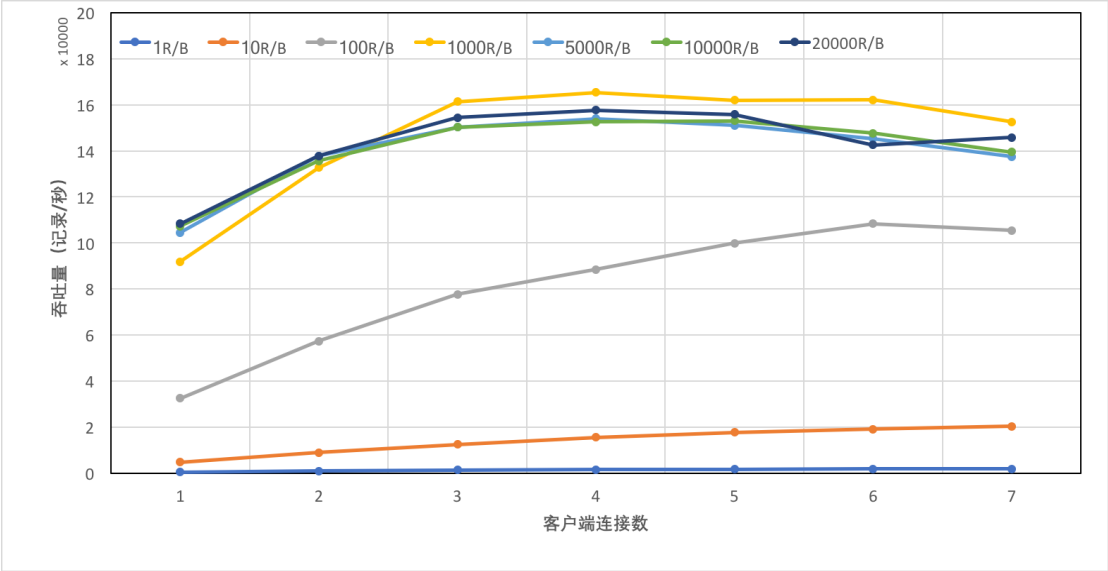


上述数据表明，Cassandra 在批处理请求数目增加时，其写入速度增加。而当单批次写入记录数目达 30 条以后，Cassandra 的写入性能不再提升，且其写入速度稳定在 45,000 记录/秒。同时我们可以看出，多客户端连接并不能有效提高 Cassandra 的写入速度。

3.2.4 InfluxDB 结果

InfluxDB 提供了一个数据存储结构（Batch Points）用来暂时存储记录，后经（Write）操作写入数据库。本测试中同样测试了 InfluxDB 在不同客户端连接数以及不同 Batch Points 中暂存记录数目下的写入速度。

条数	1 client	2 clients	3 clients	4 clients	5 clients	6 clients	7 clients
1	510	933	1,371	1,625	1,738	1,876	1,977
10	4,865	8,916	12,525	15,650	17,770	19,087	20,335
100	32,518	57,418	77,615	88,442	99,867	108,274	105,407
1000	91,855	132,672	161,340	165,385	161,970	162,157	152,597
5000	104,519	137,812	150,250	153,971	150,997	145,363	137,473
10000	107,264	135,657	150,311	152,592	153,017	147,580	139,434
20000	108,336	137,820	154,390	157,606	155,721	142,557	145,802

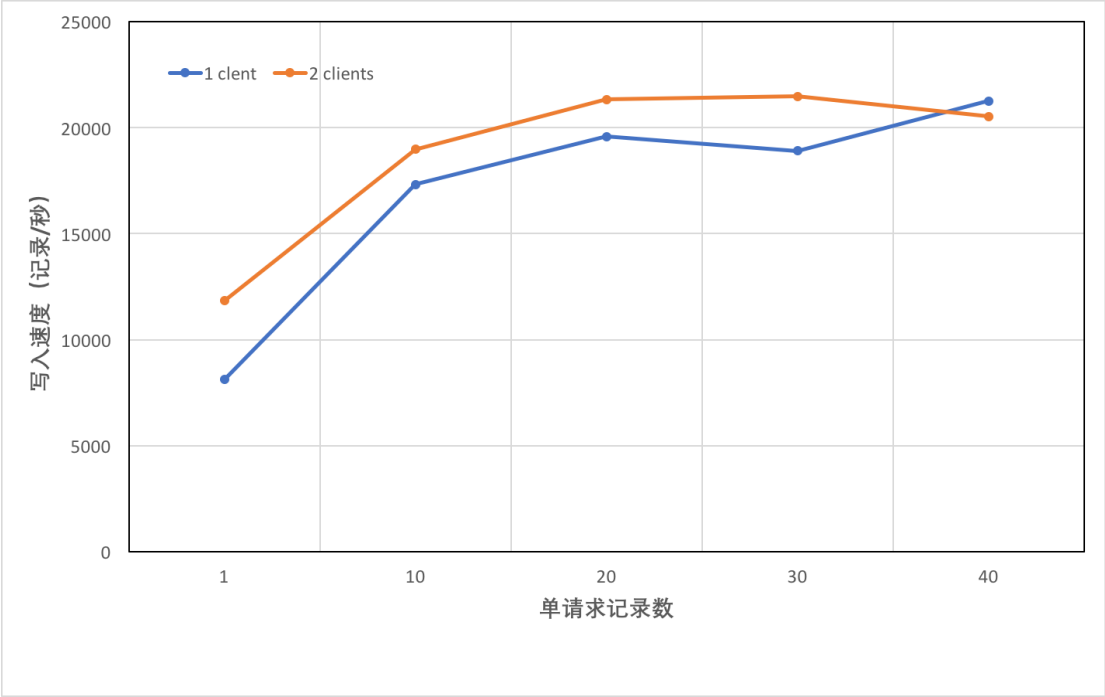


上述数据表明，在 Batch Points 中暂存记录数目比较少时，InfluxDB 的写入速度随客户端连接数的增加近乎线性增加。当 Batch Points 中暂存记录的数目较多时，InfluxDB 的写入速度刚开始随客户端的增加而增加。但当客户端连接数增加到一定数目时，InfluxDB 的写入速度不会再增加。一般认为此速度为数据库的极限写入速度。从图中可以看出，单节点情况下的 InfluxDB 在本测试中的极限写入速度大约为 160,000 记录/秒。

3.2.5 OpenTSDB 结果

在测试过程中发现，当有三个或三个以上的客户端连接 OpenTSDB 时，OpenTSDB 经常会出现不工作的情况，甚至崩溃。为了保证测试的准确性，本测试只对 OpenTSDB 在 1 和 2 各客户端连接的情况下的性能进行了测试和分析。其结果如下：

	1 client	2 clients
1 record/request	8,118	11,836
10 records/request	17,330	18,983
20 records/request	19,586	21,328
30 records/request	18,916	21,473
40 records/request	21,262	20,522

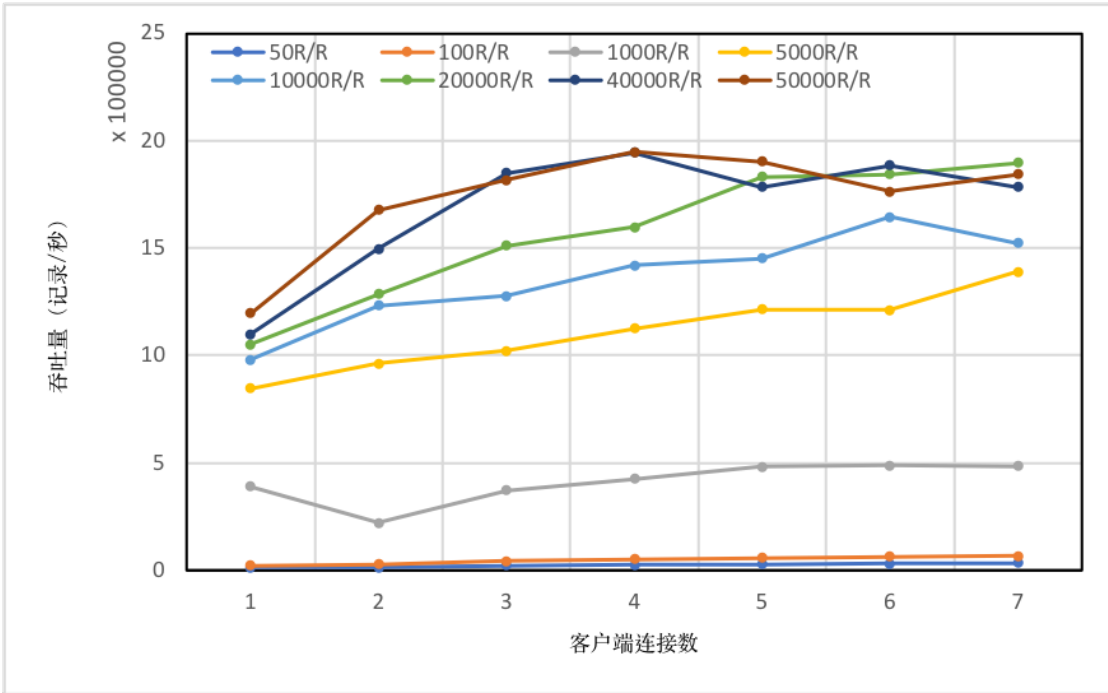


上述结果表明，OpenTSDB 的写入速度随单请求中记录数目的增加而增加，但最终增长速度趋缓，且最终写入速度大概在 20,000 记录/秒。

3.2.6 ClickHouse 结果

本次我们测试的是 ClickHouse 最为先进的 MergeTree 存储引擎。由于此引擎对于单请求的记录条数有一定要求（官网推荐是至少 1000 条记录/请求），本测试分别对于 50R/R，100R/R，1000R/R，5000 R/R，10000 R/R，20000 R/R，40000 R/R，50000R/R 做出测试，结果如下：

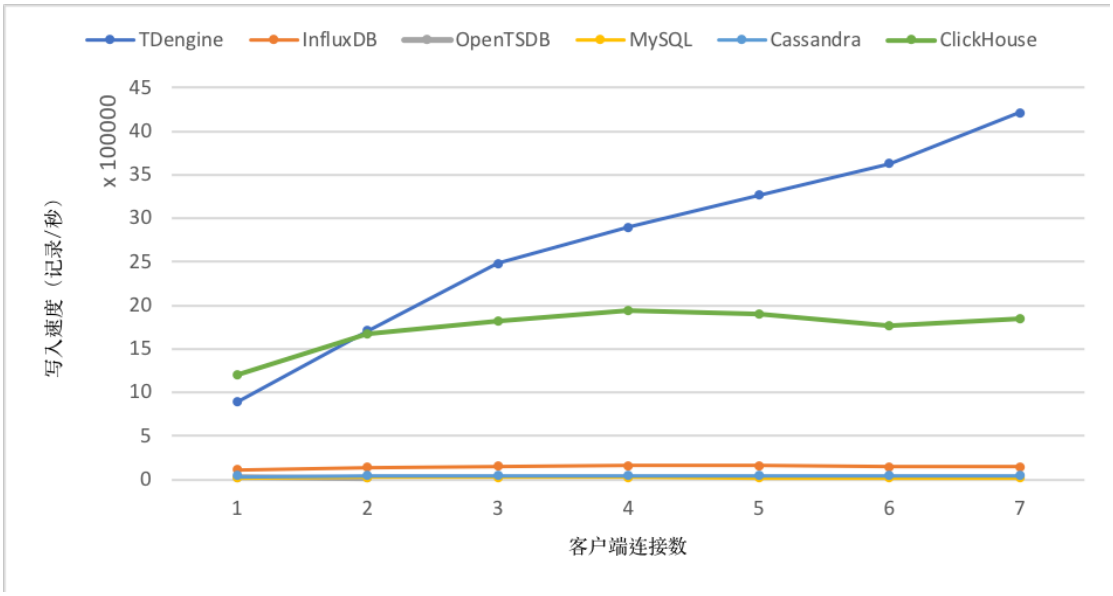
条数	1 client	2 clients	3 clients	4 clients	5 clients	6 clients	7 clients
50	11,784	14,502	21,873	24,802	28,163	31,672	34,894
100	22,476	28,377	44,061	52,128	56,740	63,,453	67,685
1000	390,447	221,509	371,381	425,478	480,771	488,884	484,150
5000	847,015	962,335	1,022,977	1,125,831	1,215,725	1,211,307	1,390,877
10000	980,480	1,231,578	1,276,821	1,419,505	1,453,369	1,646,251	1,524,263
20000	1,050,093	1,286,140	1,510,349	1,597,315	1,831,656	1,842,106	1,896,896
40000	1,100,040	1,496,384	1,850,463	1,943,466	1,783,984	1,886,177	1,785,681
50000	1,195,916	1,677,832	1,815,918	1,947,167	1,901,642	1,764,432	1,843,326



上述数据表明，ClickHouse 与 InfluxDB 比较相似。当单请求中记录数目比较少时，ClickHouse 的写入速度随客户端连接数的增加近乎线性增加。当单请求中记录的数目较多时，ClickHouse 的写入速度刚开始随客户端的增加而增加，但当客户端连接数增加到一定数目时（在本测试中是 4 或者 5），ClickHouse 的写入速度不会再增加。因此认定此速度为数据库的极限写入速度。从图中可以看出， ClickHouse 在本测试中的极限写入速度大约为 1,940,000 记录/秒。

3.2.7 各数据库最佳性能对比

基于以上的测试数据，将各数据库测试出来的最佳速度进行对比，结果如下:



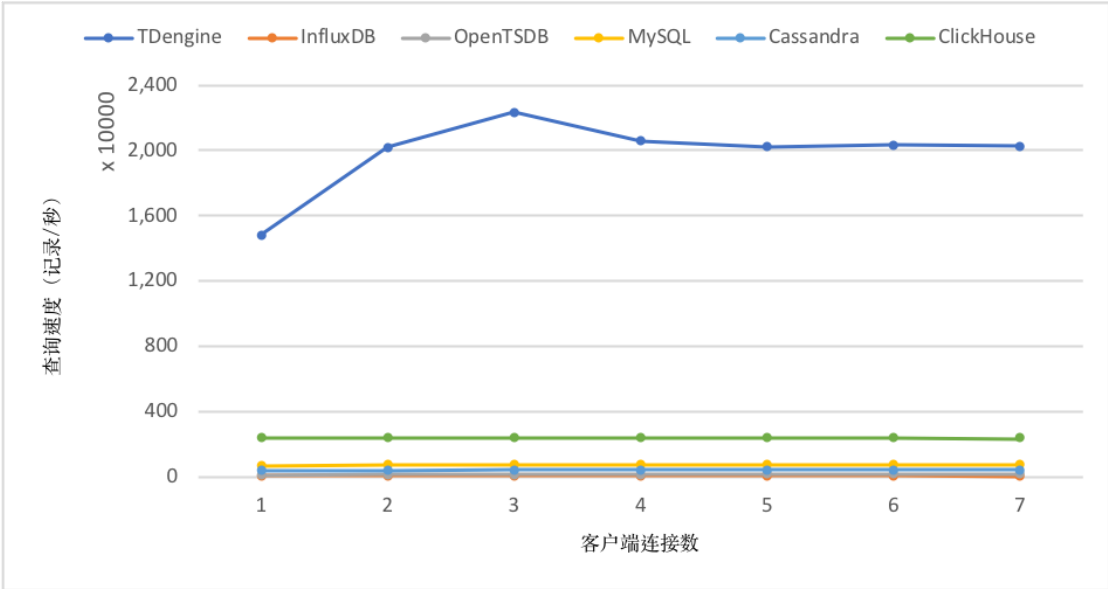
从图中可以看出，TDengine 是 6 个数据库中唯一一个写入速度随客户端一致线性增加的数据库，且其写入速度远远高于其他三个数据库，约为百万记录/秒的写入速度。而 InfluxDB、MySQL、ClickHouse 和 Cassandra 在测试中都展示出了瓶颈。ClickHouse 虽然在客户端较少的情况下速度较快，但是其速度无法因客户端增加而线性增加，从图可知 ClickHouse 的瓶颈在两百万记录/秒。而 Open TSDB 的单节点写入速度虽然有～20,000 记录/秒，但其稳定性非常差，在超过 3 个节点以后，数据库经常无法正常工作，导致图中 Open TSDB 的部分数据点丢失。

综上所述，TDengine 在多客户端连接同步写入的速度远远高于同等条件下的 MySQL，Cassandra, InfluxDB 和 Open TSDB 以及 ClickHouse。TDengine 在 7 个客户端连接同时写入的情况下可达到～4,000,000 记录/秒的写入速度，是已测得 Influx DB 的最大写入速度的 18 倍，是 Cassandra 最大写入速度的 75 倍，是 Open TSDB 和 MySQL 最大写入速度的 150 倍。

3.3 读取性能对比

本测试做了简单的查询测试，就是将插入的数据全部读出。总的读取通吐量与客户端链接数有关系，多个链接的通吐量应该比单个的要高，因此测试中测试了多个客户端的场景。因为各大数据库的性能差异很大，为节省测试时间，每个客户端查询数据库的记录条数有区别。TDengine，ClickHouse 各查询 6500 万条记录，Open TSDB 查询 500 万条记录，Influx DB、Cassandra、MySQL 各查询 100 万条记录。测试结果如下：

	1 client	2 clients	3 clients	4 clients	5 clients	6 clients	7 clients
TDengine	14,810,521	20,190,429	22,326,609	20,583,832	20,218,376	20,317,405	20,255,069
InfluxDB	70,589	76,530	79,680	78,500	75,235	71,687	70,138
OpenTSDB	135,436	146,488	149,329	149,808	153,441	164,561	184,580
MySQL	721,540	742,093	727,632	727,658	749,868	755,254	749,259
Cassandra	367,788	377,244	420,378	430,227	432,190	432,179	432,162
ClickHouse	2,385,712	2,386,898	2,379,902	2,380,021	2,379,399	2,376,074	2,373,440

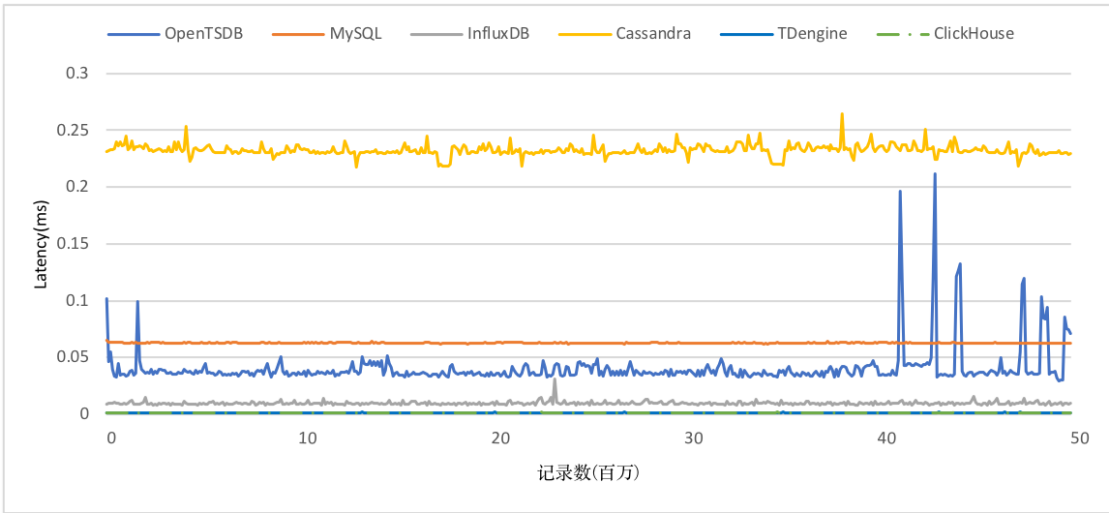


从图表中可以看出，TDengine 的最大读取速度稳定在 20,000,000 记录/秒左右。ClickHouse 的最大读取速度不会随着客户端连接数增加而增长，相反的，读取速度会有一些下降。在只有一个客户端连接的情况下，读取速度最快，为 2,400,000 记录/秒左右。InfluxDB 的最大读取速度为 75,000 记录/秒左右，OpenTSDB 为 180,000 记录/秒，而 MySQL 的最大读取速度为 750,000 记录/秒, Cassandra 的最大读取速度在 400,000 记录/秒。所以从测试结果来看，TDengine 的查询吞吐量远高于 InfluxDB, Opentsdb 和 Cassandra, 比 MySQL 的查询最大吞吐量高一个数量级。为保证公平，TDengine 查询 6500 万条记录，因为条数巨多，以确保不是从内存读取，而且从硬盘读出。其他数据库，没有仔细研究是否全部从内存读出还是从硬盘。

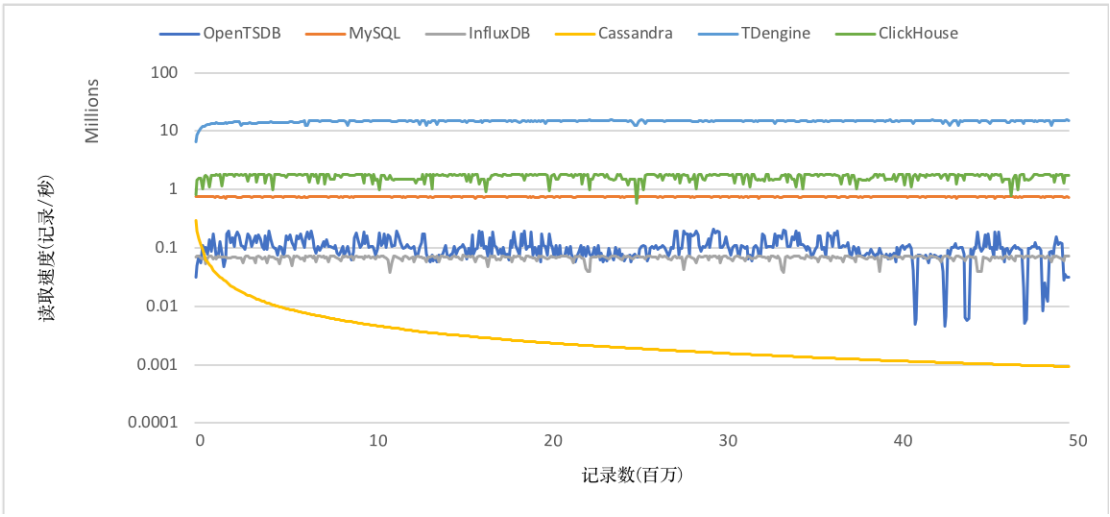
3.4 数据库现存数据量大小对写入与读取的性能影响

在插入或查询时，数据库现存数据量的大小会影响起读写性能。在该测试中，单客户端向数据库中总共写入 50,000,000 条记录，并且每插入 10 万条记录，就立即查询刚插入的 10 万条。客户端实时监控这 10 万条记录的写入速度与延迟，查询速度与延迟。结果如下：

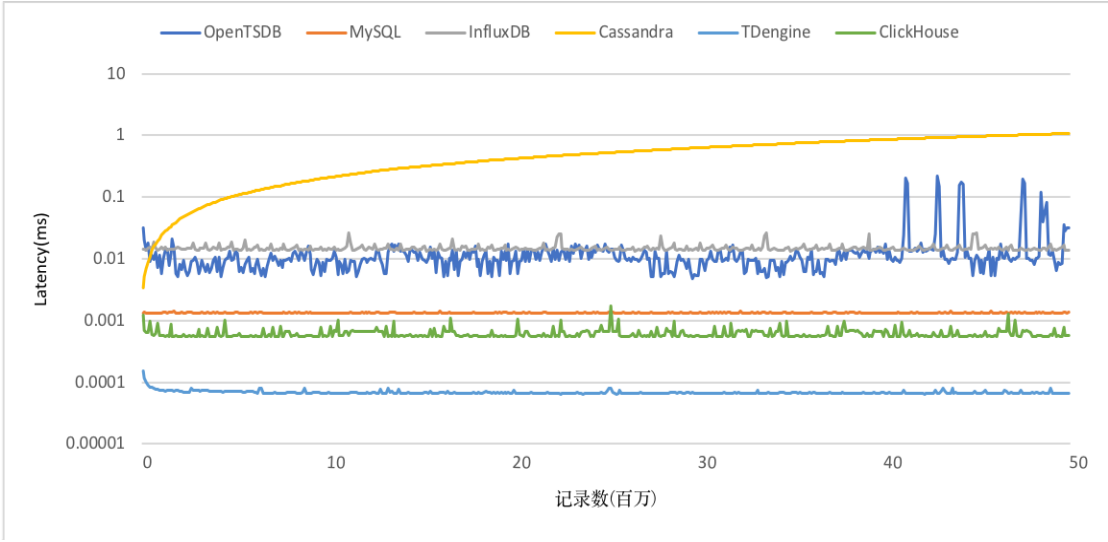
数据库写入速度随数据库现有记录条数的变化



数据库写入延时随数据库现有记录条数的变化



数据库读取速度随数据库现有记录条数的变化



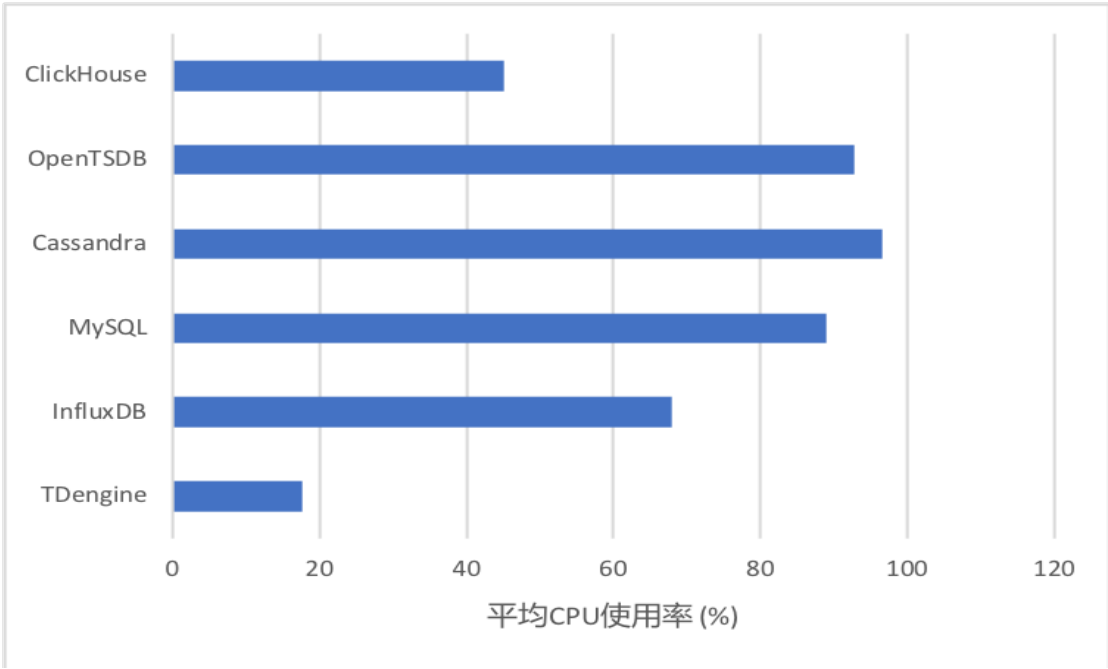
数据库读取延迟随数据库现有记录条数的变化

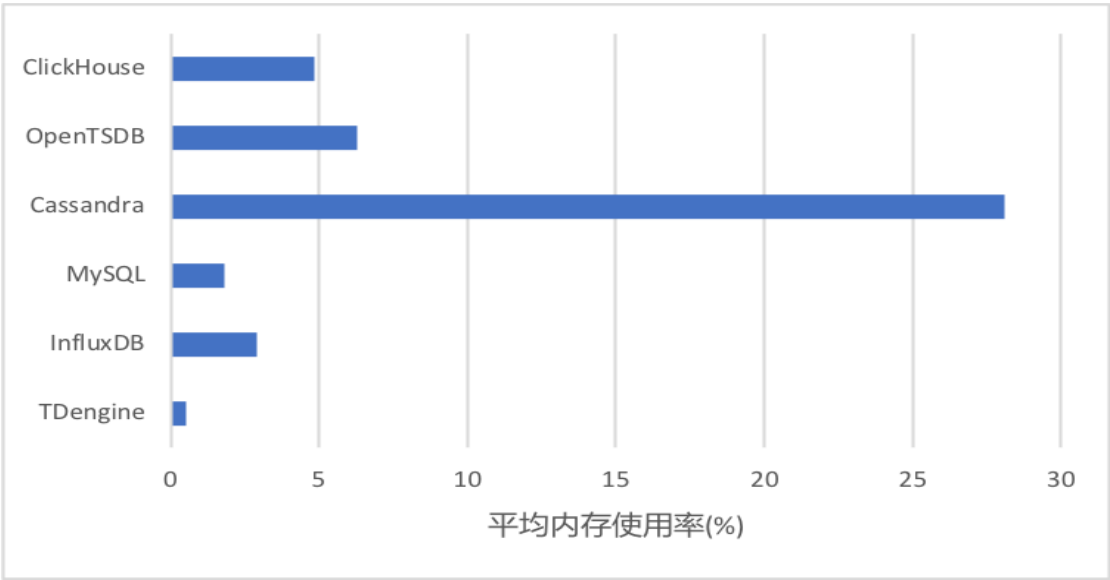
从图中我们可以看出，除了 ClickHouse 之外的所有数据库在写入过程中性能都比较平稳，速度和延迟几乎保持不变。除了 Cassandra，其余数据库在读取过程中性能也比较稳定。但是 TDengine 在写入速度和读取上表现尤其出色。且 TDengine 的写入和查询延迟均时最小的。从写入和读取过程中的平稳性来看，TDengine 和 MySQL 在写入和读取过程中性能波动很小，InfluxDB 的波动稍微较大，ClickHouse 写入时性能波动较大，而 Open TSDB 的性能波动是最大的。

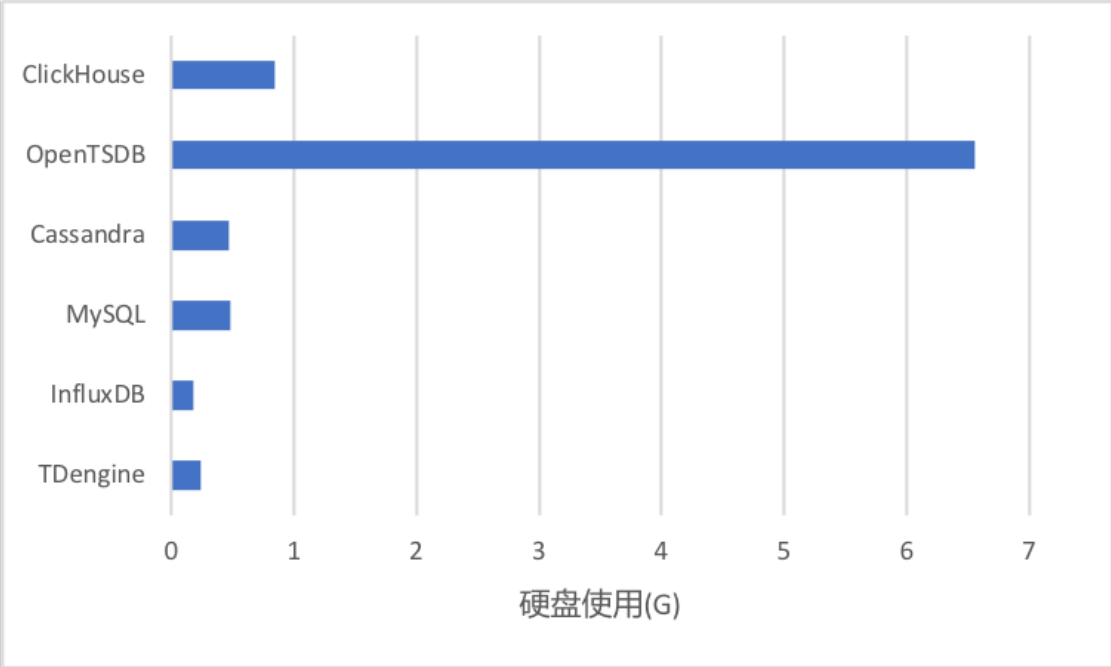
3.5 CPU、内存、硬盘消耗对比

单客户端，持续写入一千万条记录，每隔一秒钟对占用的 CPU, Memory 和硬盘空间进行取样，最后平均，得到如下结果：

	Average CPU Usage (%)	Average Memory Usage (%)	Disk Usage (K)
TDengine	17.72592593	0.514814815	245868
InfluxDB	67.89373134	2.909850746	184900
MySQL	88.93493205	1.811111111	487688
Cassandra	96.6736	28.082	467792
OpenTSDB	92.7164486	6.288598131	6557872
ClickHouse	45.2	4.86	846840


ClickHouse
OpenTSDB
Cassandra
MySQL
InfluxDB
TDengine
0
20
40
60
80
100
120
平均CPU使用率 (%)


ClickHouse
OpenTSDB
Cassandra
MySQL
InfluxDB
TDengine
0
5
10
15
20
25
30
平均内存使用率(%)



从上述结果看出，TDengine 在 CPU 和内存的使用率上占绝对优势。与 OpenTSDB 相比，CPU 不到其 1/5，内存不到其 1/10，这个数字与 TDengine 的超高写入性能是吻合的。Cassandra 在 CPU 和内存的消耗上十分惊人。硬盘使用空间上，Influx DB 略胜 TDengine，而 OpenTSDB 硬盘消耗是 TDengine 与 Influx DB 的 30 倍。

需要指出的是，测试用的数据，除时间戳外，其余的数据全部是随机生成的，因此对于 TDengine 而言，压缩率并不理想（1000 万条记录，原始数据为 320M，压缩后为 209M）。对于实际场景的数据，因为有一定的规律性，压缩率会大幅提高。

3.6 函数性能测试

3.6.1 测试环境及步骤

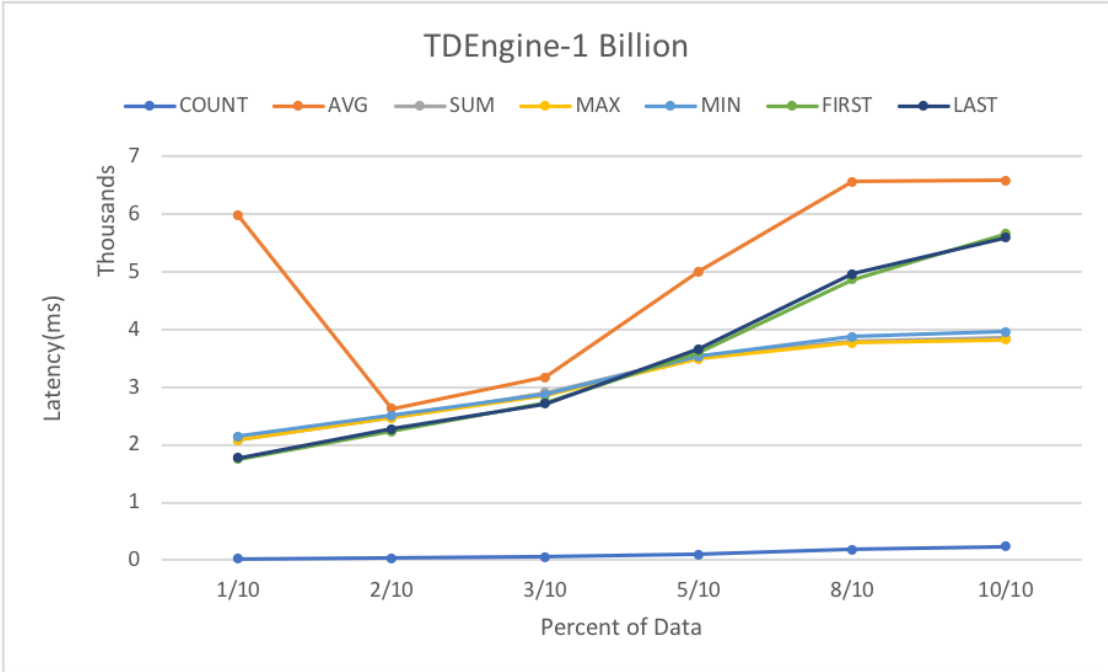
本单元测试解除了对 CPU 的单核限制来获取各种数据库在各种函数方面理论上的性能最大值，测试对象为 TDEngine，InfluxDB，OpenTSDB 这三个时序数据引擎，实时分析引擎 ClickHouse,传统关系数据引擎 MySQL 以及 NoSQL 的 Cassandra。测试使用的数据是由 5000 个感应器收集，每一条记录有以下内容：时间戳（8 字节），设备型号（8 字节），位置（4 字节），温度（4 字节），压力（4 字节），电流（4 字节），电压（4 字节）。一共 36 个字节。这些数据一共有 10 个可能的设备型号与 500 个可能的位置来模拟 5000 个感应器（每一个设备型号搭配位置可以定义一个感应器）。因为每个数据库数据模型不同，每个数据库插入时的数据结构与设定也不完全相同：为了符合 ClickHouse 的 MergeTree 引擎设定，所有数据都会插入一张表，该表的复合主键是由（设备型号，位置及时间(精确到日)，时间戳（24 小时内））组合而成，值得注意的是 MergeTree 要求有一个精确到日的时间(Date)，为了使其数据对标，再加上一列 24 小时内的时间戳；InfluxDB 设定一个 measurement，所有记录都属于该 measurement。每条记录有两个 tag，设备型号和位置，用以区分感应器；由于 OpenTSDB 一条记录只能记录一个值，本测试设定四个 metric，来分别记录温度，压力，电流，电压。

每个 metric 有两个 tag 用以区分感应器，该设定与 InfluxDB 相同；TDengine 首先建立一个 metric 表，再用该 metric 来建立 5000 个表对应 5000 个感应器，每条记录会插入其对应的感应器对应的表。MySQL 会建立一张复合主键是（设备型号，位置，时间）的表，为全部数据的插入对象；对于 Cassandra，我们会建立一张会建立一张复合主键是（设备型号，位置，时间）的表，其中 partition key 是设备型号，此设置符合接下来测试使用的 query，使性能尽可能的提高。测试具体步骤如下：单客户端，插入 1,000,000,000 条记录（所有数据由 5000 个感应器均匀收集产生），之后使用特定函数配合筛选条件来测试性能。为了使 OpenTSDB 能够大规模插入数据，在建表时会进行 pre-split，把表 tsdb 分成四个 region，对应四个 metrics。由于 OpenTSDB 和 MySQL 的性能限制，插入的记录限制为：100,000,000。值得注意的是，由于同一数据库下的函数测试之间并没有清除缓存，个别数据库的函数在选取十分之一的记录时的延迟会有一个较大的增加，建议忽略。

3.6.2 多个数据采集点聚合函数性能测试

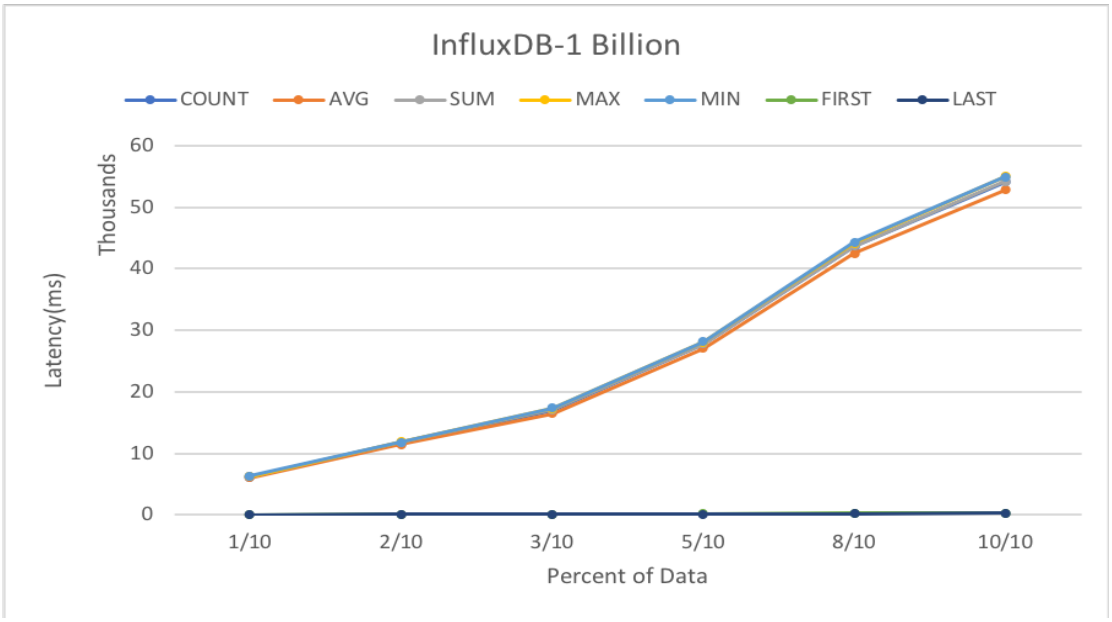
本单元的测试包含 COUNT，AVG，SUM，MAX，MIN，FIRST，LAST 这七个 TDEngine，InfluxDB，OpenTSDB 以及 ClickHouse 共有的聚合函数（名字可能有所不同）。对于 MySQL 以及 Cassandra，由于其不支持 FIRST，LAST，仅测试 COUNT，AVG，SUM，MAX，MIN 这五个聚合函数。所有测试函数都会搭配筛选条件（WHERE）来选取整个数据采集点（所有记录，包含全部 5000 个感应器）的十分之一，十分之二，十分之三，十分之五，十分之八以及十分之十（全部）的记录。测试结果如下：

3.6.2.1 TDEngine 结果



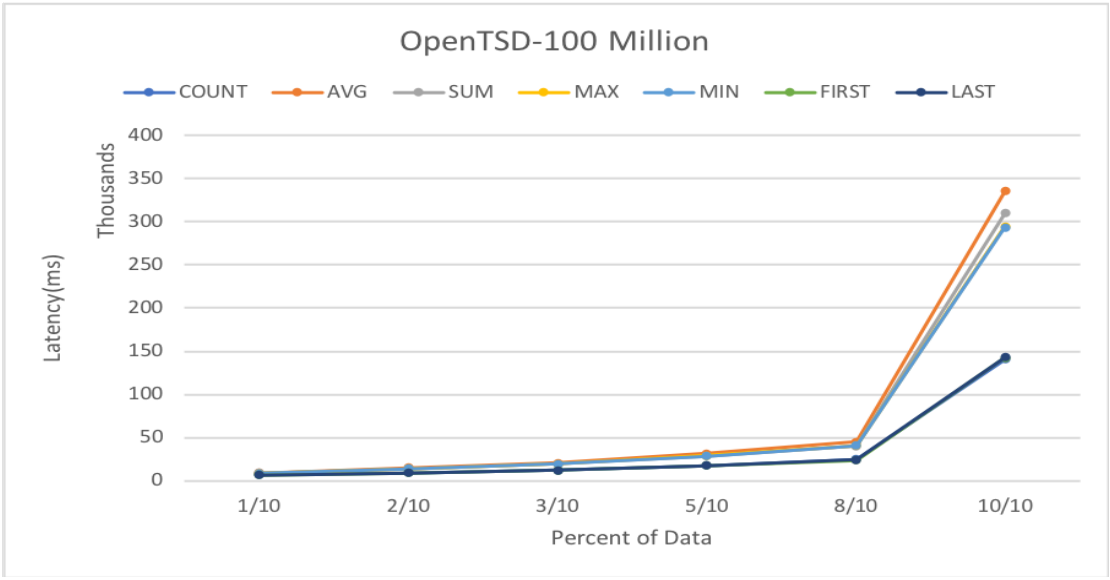
从上图可知，TDengine 在总共 10 亿条记录的情况下，COUNT 是 7 个函数中性能最佳的函数，总用时在 300ms 以内。其他函数用时全部小于 7000ms。

3.6.2.2 InfluxDB 结果



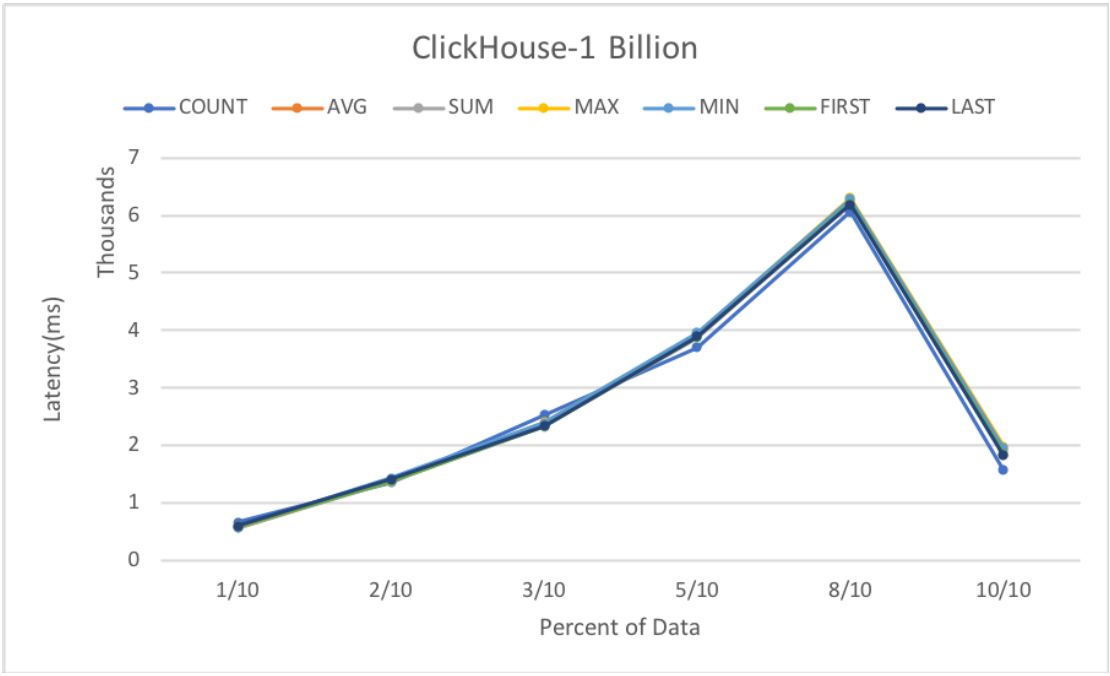
从上图可以看出 InfluxDB 的 SUM，MIN，MAX，COUNT，AVG 用时随着数据量的增多而线性增加，这五个函数查询全部十亿条记录时的延迟在 55,000ms 左右。另外两个函数，FIRST，LAST 的延时并不会随着数据量增加而增加，这两个函数的延迟稳定在 300ms 左右。

3.6.2.3 OpenTSDB 结果



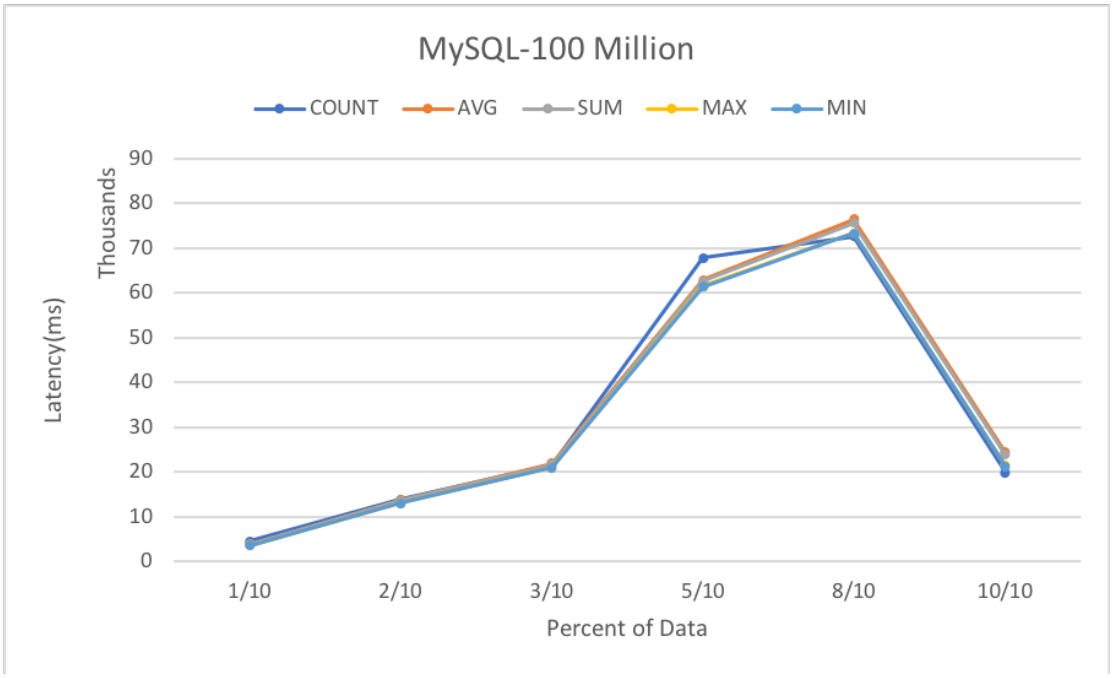
从上图可以看出 OpenTSDB 的所有函数的用时都会随着数据量的增多而增加，并且没有筛选条件的函数用时比有筛选条件的函数用时有较为显著的增加。总体来说 AVG，SUM，MAX，MIN 这四个函数的延迟较高，在查询全部一亿条记录时在 300,000ms 左右。另外三个函数 COUNT，FIRST，LAST 的延时相对较低，这三个函数查询全部一亿条数据时的延迟在 140,000ms 左右。

3.6.2.4 ClickHouse 结果



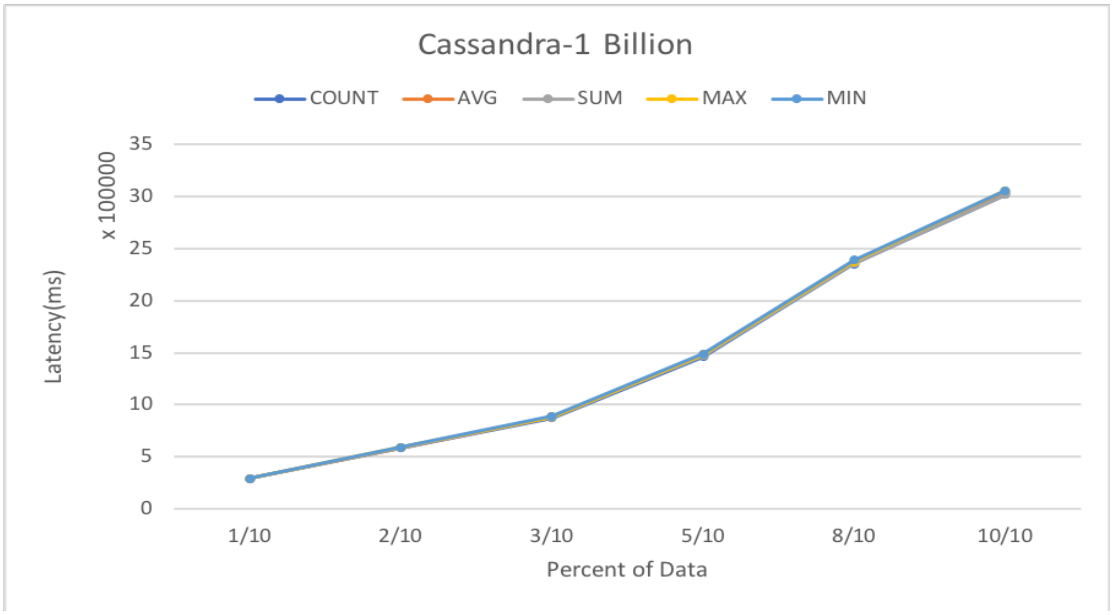
从上图可以看出 ClickHouse 的所有函数在有筛选条件的情况下的用时都会随着数据量的增多而线性增加。当查询语句不含筛选条件（查询全部数据）时，所有函数的延迟有一个明显的较低。所有的函数的延迟在同条件下基本相同，值得注意的是，ClickHouse 在查询全部十亿条记录的情况下性能比较优秀。ClickHouse 的聚合函数在有筛选条件时的平均用时大约为 6,300ms，当没有筛选条件时，其用时大约为 1,900ms。

3.6.2.5 MySQL 结果



分析上图，可以得知 MySQL 的以上五个函数的总体性能与 ClickHouse 比较相似：在没有筛选条件的情况下，查询全部一亿条记录用时比有条件的查询部分记录更少。查询全部记录用时为 20,000ms 左右，查询部分记录的最大用时为 76,000ms 左右。

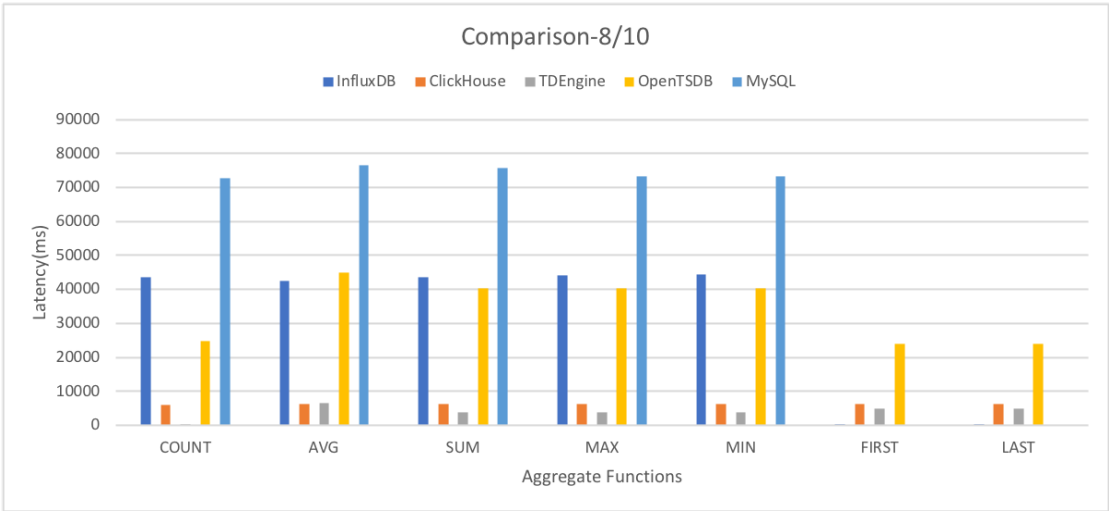
3.6.2.6 Cassandra 结果



从上图可以得知，Cassandrsa 的五个聚合函数在同条件下性能基本一致，并且用时随着数据量的增加而线性增加。查询全部十亿条记录的用时大约在 3,000,000ms 左右。

3.6.2.7 多个数据采集点聚合函数性能对比

	TDengine	InfluxDB	ClickHouse	OpenTSDB	MySQL	Cassandra
COUNT	183.4	43559.4	6057.4	24777.2	72648.2	2352268
AVG	6557.8	42539.4	6244	45033	76451.2	2371976.6
SUM	3804.6	43695.2	6211.4	40300	75671.8	2357950.2
MAX	3763.8	44136.4	6302	40411.6	73387	2374148.4
MIN	3872.2	44302.8	6285.6	40421.8	73311.8	2386035.6
FIRST	4867.4	270.4	6205.2	23869	NULL	NULL
LAST	4965.2	223.4	6183.4	24008.6	NULL	NULL

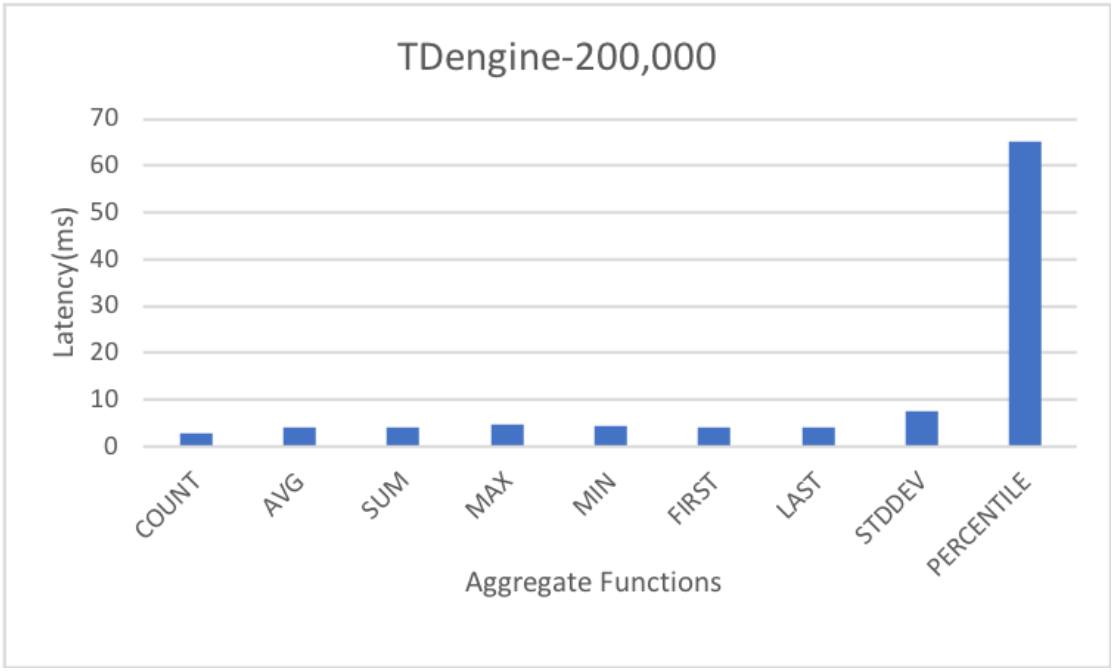


上图选取的是 InfluxDB，ClickHouse，TDengine 在查询 10 亿条记录的情况下，OpenTSDB 以及 MySQL 查询一亿条记录的情况下（由于性能限制），选取全部十分之八的函数性能用时。值得注意的是，由于 MySQL 不支持 FIRST，LAST，本图不包含 MySQL 在这两种函数下的用时。考虑到图标的清晰性，由于 Cassandra 的性能比图表中最差的 MySQL 还慢了 40 倍左右，图表中不包含 Cassandra 的函数性能对比。从上图可以得出，TDengine 在大数据情况下进行条件筛选时，上图中的大部分函数性能都优于其他数据库。TDengine 在 COUNT 函数方面有绝对的性能优势，比同条件下其他数据库中最快的 ClickHouse 快了 6 到 60 倍。在其他 5 个函数方面也都优于其他数据库中最快的 ClickHouse。TDengine 的 COUNT 函数比主流时序数据库 InfluxDB 的 COUNT 函数快了 240 到 400 倍。TDengine 的其他函数（除了 FIRST 和 LAST 外）也比 InfluxDB 的对应函数快了 10 到 14 倍。

3.6.3 单个数据采集点聚合函数性能测试

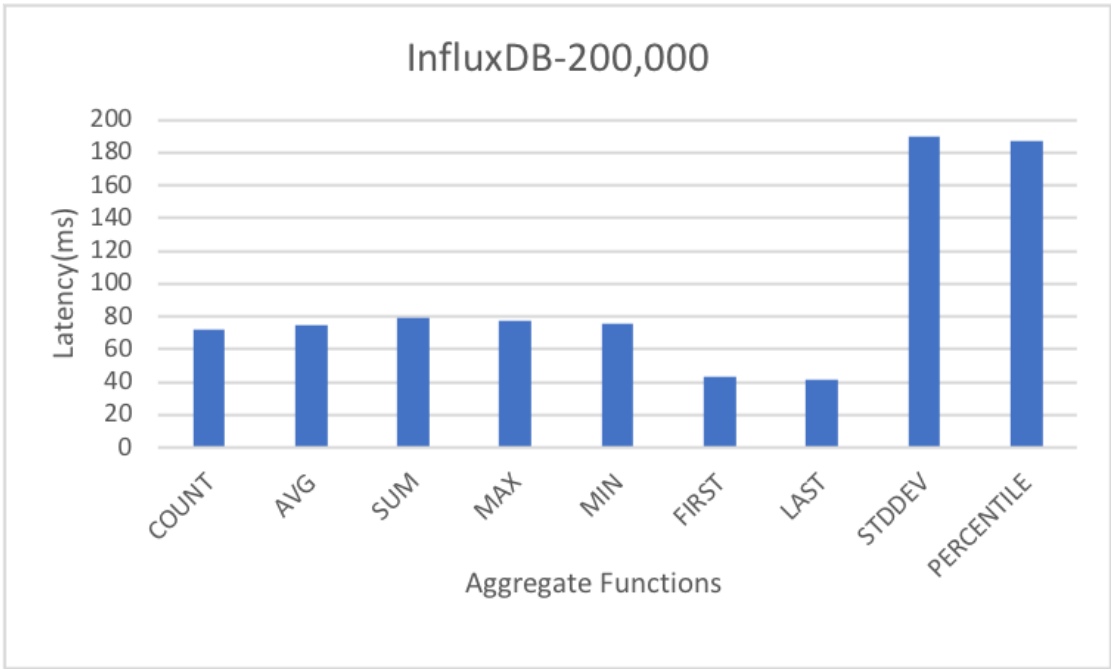
本单元的测试包含 COUNT，AVG，SUM，MAX，MIN，FIRST，LAST，STDDEV，PERCENTILE 这九个 TDEngine，InfluxDB，OpenTSDB 以及 ClickHouse 共有的适用于 Table（一个感应器）的函数（名字可能有所不同）。对于 MySQL 和 Cassandra 而言，本次测试 COUNT，AVG，SUM，MAX，MIN 这五个聚合函数。 由于是对于单个 Table/数据采集点的测试，所有测试函数都会选取单个表或者单个数据采集点的记录，必要时通过搭配筛选条件（WHERE）来确保对于不同数据模型以及数据库，所选取的记录相同。测试结果如下：

3.6.3.1 TDEngine 结果



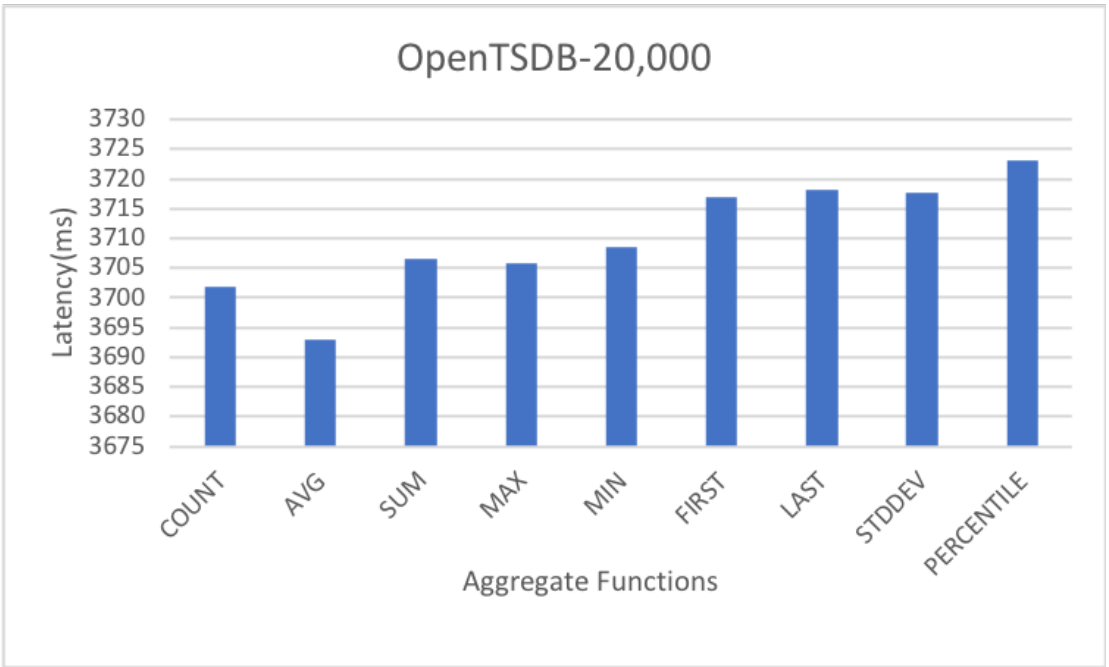
从上图可知，TDengine 从十亿总量的记录中选取单个感应器产生的二十万条数据用时非常短，最慢的延时在 65ms 左右，最快的函数延时在 10ms 以下。

3.6.3.2 InfluxDB 结果



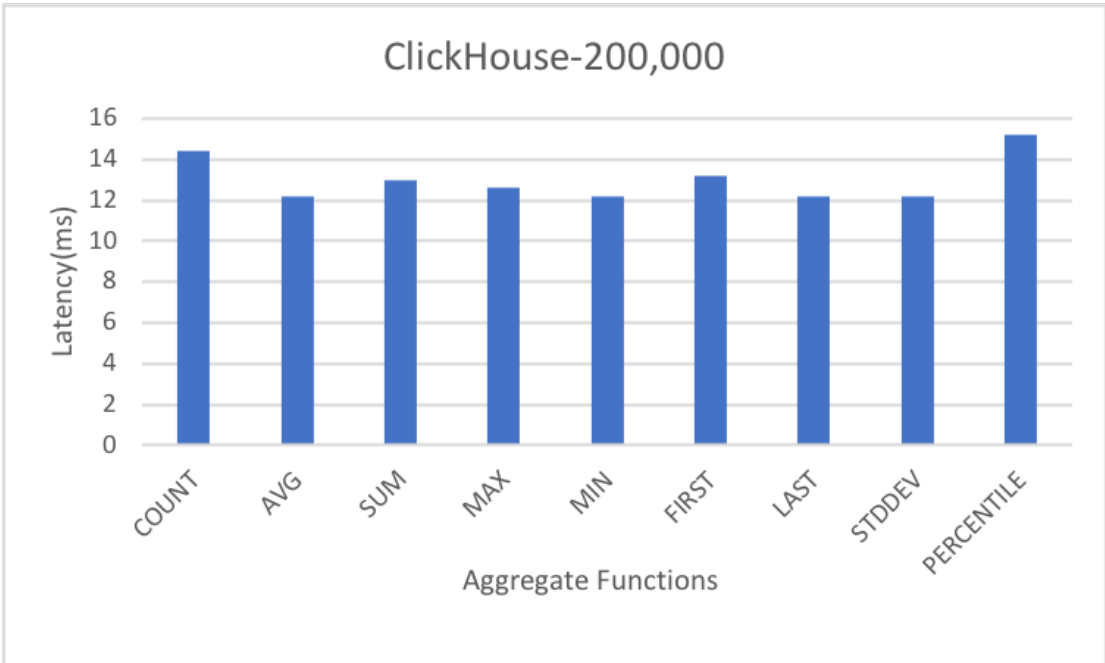
从上图可知，InfluxDB 在查询一个感应器的所有记录（二十万条记录）时，的 STDDEV 和 PERCENTILE 这两个函数的用时较高，大约在 180ms 左右。FIRST，LAST 两个函数用时较低大约在 50ms 左右。其余的五个函数的用时基本相同，在 70ms 左右。

3.6.3.3 OpenTSDB 结果



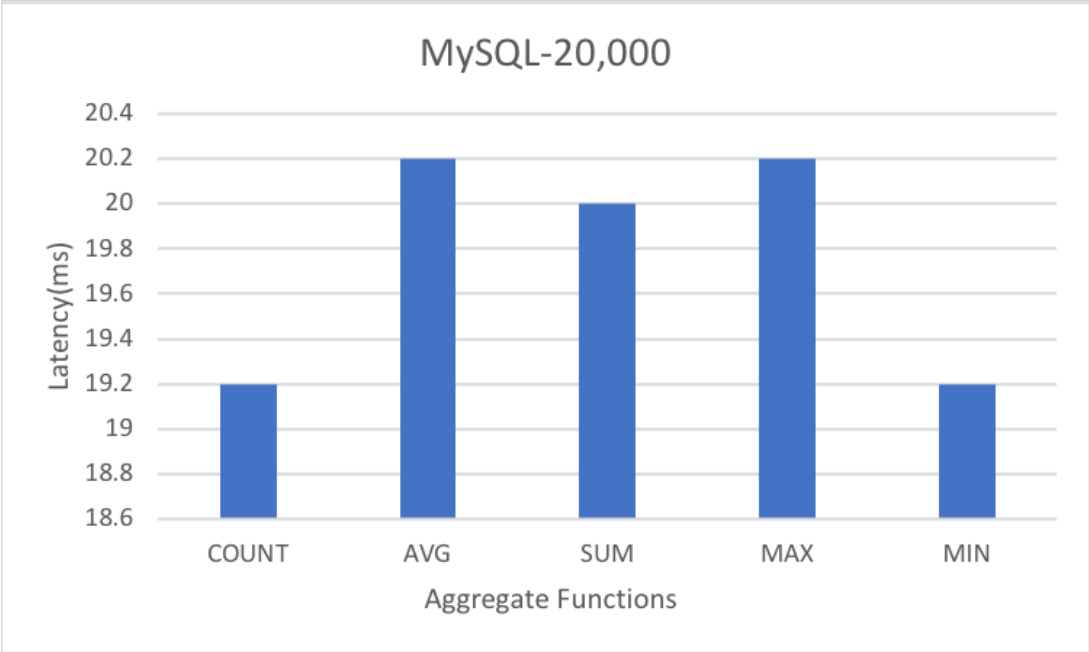
从上图可以得知，OpenTSDB 在查询一个感应器，共两万条记录时，所有函数用时基本相同，大约为 3700ms。

3.6.3.4 ClickHouse 结果



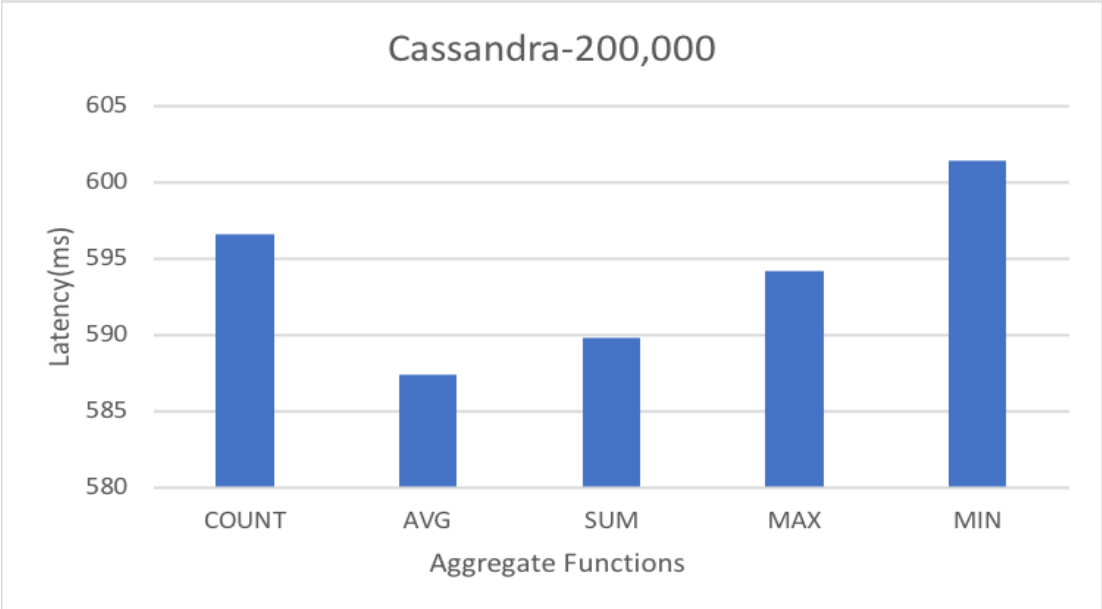
从上图可以得到，ClickHouse 在只查询一个感应器，共二十万条记录时所有函数用时有一定的波动，基本用时在 12ms 左右。

3.6.3.5 MySQL 结果



从上图可以得到，MySQL 在只查询一个感应器，共两万条记录时所有函数用时基本相同，在 20ms 左右。

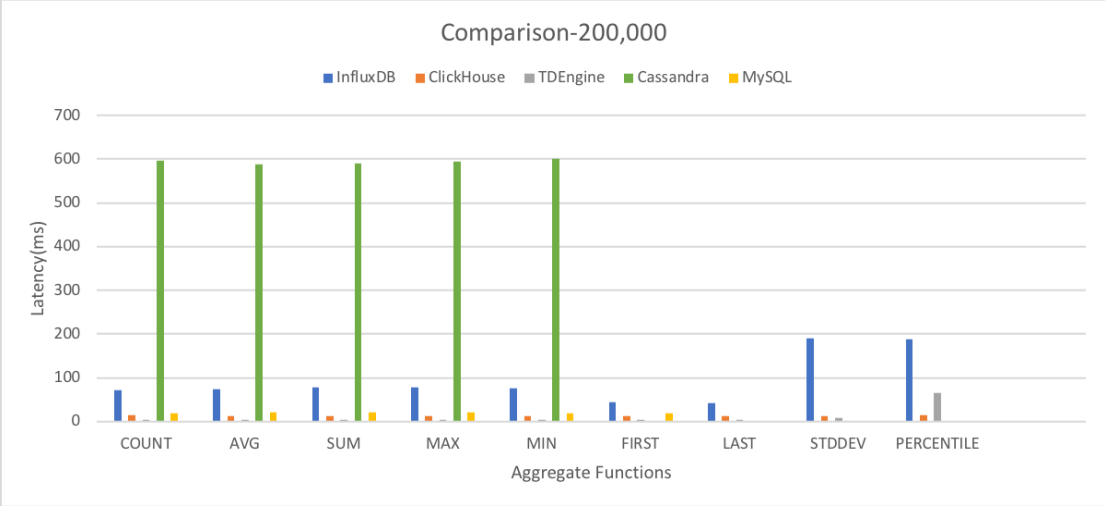
3.6.3.6 Cassandra 结果



结合上图，Cassandra 在只查询一个感应器，共二十万条记录时所有函数用时基本相同，为 600ms 左右。

3.6.3.7 单个数据采集点聚合函数性能对比

	TDengine	InfluxDB	ClickHouse	OpenTSDB	MySQL	Cassandra
COUNT	2.8	72	14.4	3701.8	19.2	596.6
AVG	4	74.6	12.2	3693	20.2	587.4
SUM	4.2	78.8	13	3706.6	20	589.8
MAX	4.6	77.4	12.6	3705.8	20.2	594.2
MIN	4.4	75.8	12.2	3708.4	19.2	601.4
FIRST	4	43.2	13.2	3716.8	NULL	NULL
LAST	4	41.8	12.2	3718.2	NULL	NULL
STDDEV	7.6	189.8	12.2	3717.6	NULL	NULL
PERCENTILE	65.2	187.2	15.2	3723	NULL	NULL



结合上图可知，TDEngine 在针对单个数据采集点的查询有非常明显的优势，相比较于第二快的 ClickHouse，TDEngine 在查询除了 PERCENTILE 的函数时的速度是 InfluxDB 执行相同函数同样条件下的 3 倍左右。与 OpenTSDB 做比较，TDEngine 的速度至少是 OpenTSDB 的五百倍。由于 OpenTSDB 比 TDengine 慢了至少 500 倍。为了确保图像的清晰以及可读，上图并不包含 OpenTSDB 的测试结果。还需要注意的是，Cassandra 的测试只包括了前五个聚合函数：COUNT，AVG，SUM，MAX，MIN。上图结果中,除了 MySQL 查询的是 20,000 条记录，其他所有数据库均查询了 200,000 条记录。

4 TDengine 单节点指标测试

4.1 测试环境说明

单节点的测试使用的是阿里云 ecs.c5.2xlarge(ecs.c5)。详细配置如下：

- CPU：8 核 Intel Skylake Xeon Platinum 8163 2.5GHz

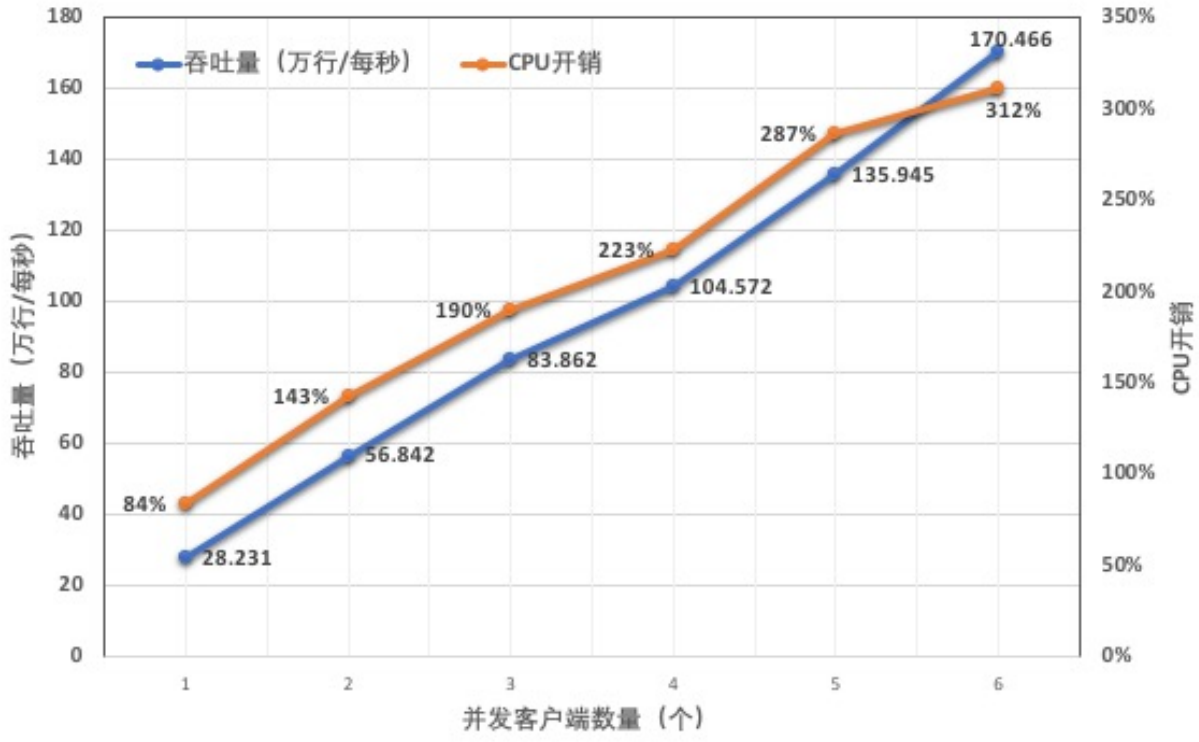
- 内存：16GB
- 内网带宽：2.5Gbps
- 每秒最大收发包：80 万 PPS
- 硬盘：40GB 高效云盘(非 SSD)，1240 IOPS
- 操作系统：Ubuntu 16.04 x64。

4.2 并发写性能

目的：改变并发进行数据写入的客户端数量，获取 TDengine 在多客户端同时并发写时候的系统性能指标。

每个连接进程向同一个数据库中 20 张表写入数据，每行数据长度 32 字节。例如：有 6 个进程的时候，6 个客户端同时向系统的 120 张表写入数据。数据写入调用异步写入接口，以 4 行记录为一个批次，进行批量连续写入。结果如下：

连接客户端（个）	吞吐量（万行/每秒.）	CPU 开销
1	28.231	84%
2	56.842	143%
3	83.862	190%
4	104.572	223%
5	135.945	287%
6	170.466	312%



可以看到，随着连接客户端的数量增长 TDengine 表现出了良好的吞吐量线性增长的趋势。在 6 个客户端连接并同时写入的情况下，系统整体的吞吐量超过 170 万行/秒。同时 CPU 负载也基本上呈现同步增长的态势，但是在 8 核的机器上仍然没有完全使用全部的 CPU 计算资源。

4.3 多表同时写性能

客户端建立单个数据库连接，调用异步写入接口同时向多个表中写入数据，以 4 条记录为一个批次，每条记录长度 32 字节。进行批量写入的性能测试。

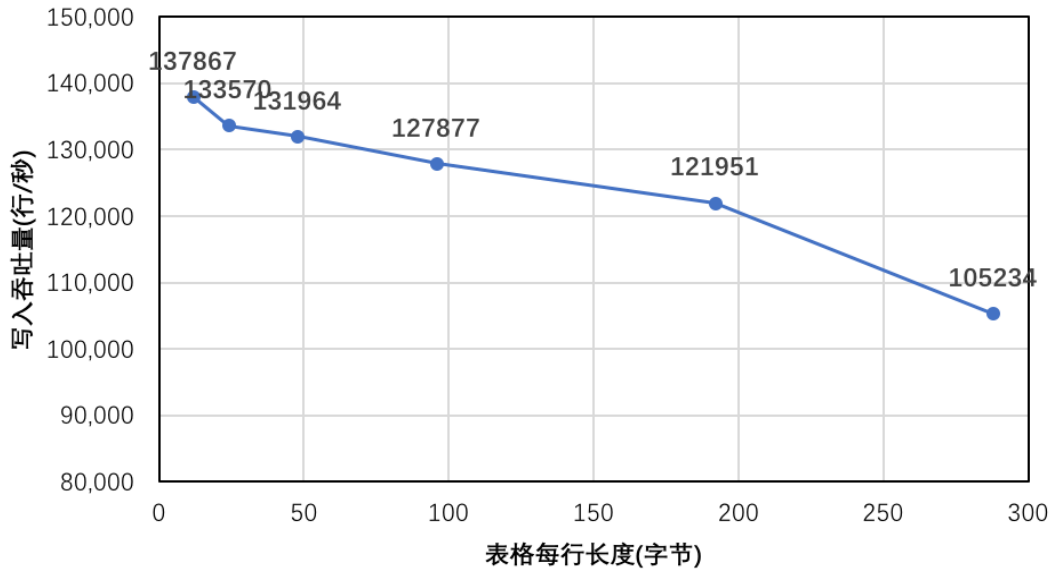
表数量	写入性能（条/秒）	CPU 开销
1	131,000	55.00%
10	280,000	97.95%
20	279,531	87.44%
30	281,921	83.00%
40	280,187	85.93%
50	289,110	88.60%
60	291,098	84.10%
70	289,810	86.43%
80	287,127	87.13%
90	279,000	82.36%
100	280,000	85.11%

上述结果表明，在同时写入表数量>=10，写入性能稳定在>270,000 行/秒。在同时向 60 个表写入数据的时候，性能达到最优（超过 290,000 行/秒），此时如果继续增加并发表的写入量，通吐量基本保持不变。如图所示，CPU 开销与系统吞吐量基本上呈同步变化趋势，基本上维持稳定的状态，整体 CPU 开销不超过 100%。

4.4 不同表长度写性能

单个客户端连接，每次向一个表中写入数据，以 4 条记录为一个批次进行连续写入，向每个表写入不同长度的数据记录。表长度的增加导致每秒写入的数据条数变小。结果如下：

记录长度（字节）	写入数据（条/每秒）
12	137867
24	133570
48	131964
96	127877
192	121951
288	105234

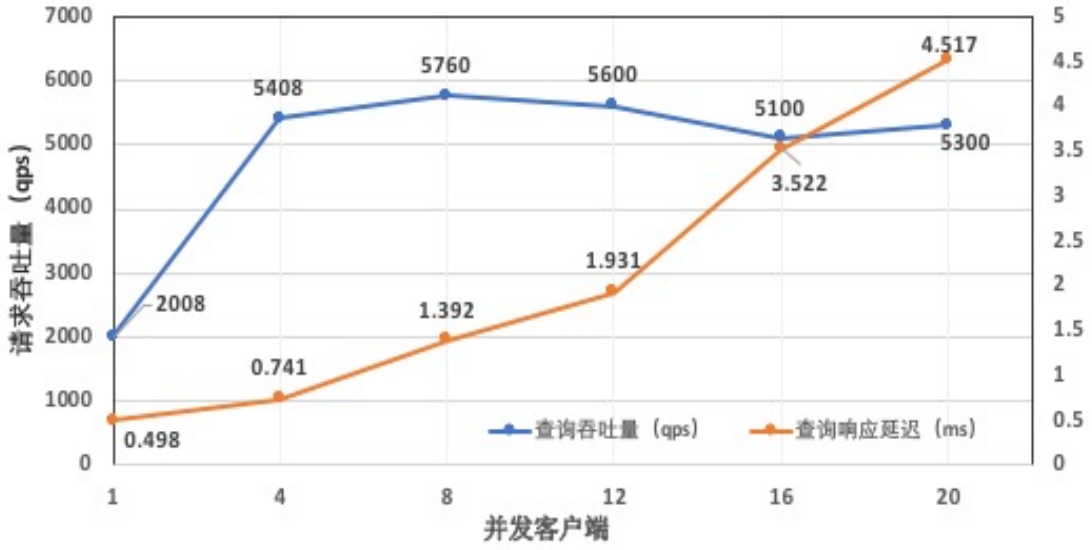


上述结果表明，随着记录长度的增加，每秒插入的条数确实下降，但下井幅度不大。当表每行长度为 288 字节的时候，性能约为表每行长度 12 字节时候的 72.57%。

4.5 并行查询性能

不同于连续性的大数据量读取，多客户并发小规模数据查询提取，能够有效地评估系统对突发性的查询响应能力。开启不同数量的客户端连接到数据库，针对 1 亿条记录的表格，进行返回固定数量记录的范围查询(range query)。为了避免缓存，起始值随机选取，连续执行 10000 次为一个周期，调整并发客户端数量共执行 5 个周期，并计算其查询吞吐量和查询延迟平均值。结果如下：

客户端数量	查询吞吐量(qps)	查询响应延迟(ms)
1	2008	0.498
4	5408	0.741
8	5760	1.392
12	5600	1.931
16	5100	3.522
20	5300	4.517

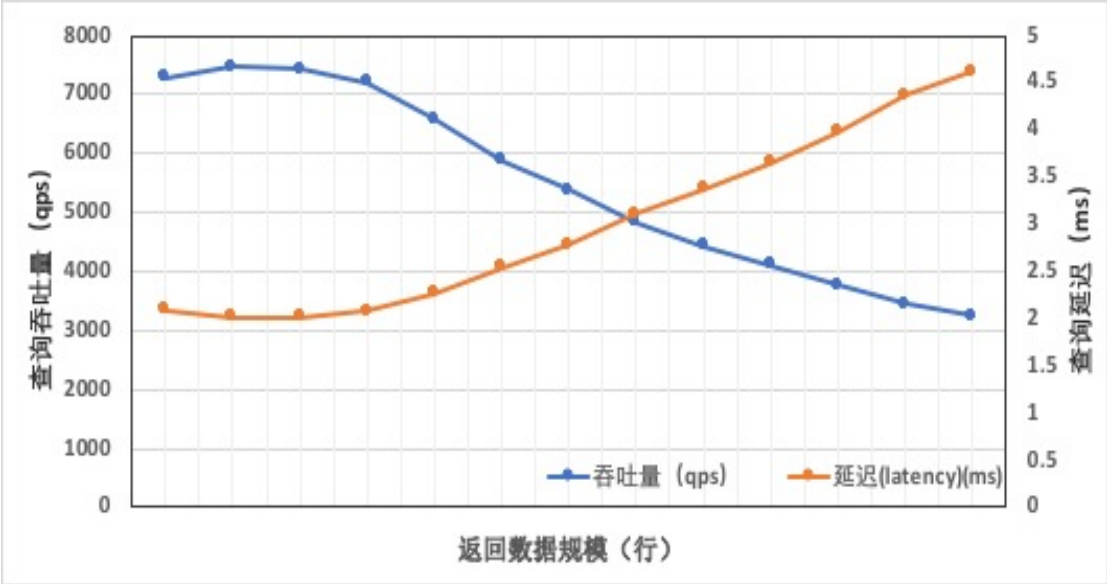


可以看到，在连接数为 8 的时候（与 CPU 核数量比为 1:1），查询的吞吐量最优，查询响应延迟随着并发客户端的增加一直呈现增长状态。

4.6 不同规模结果集查询性能

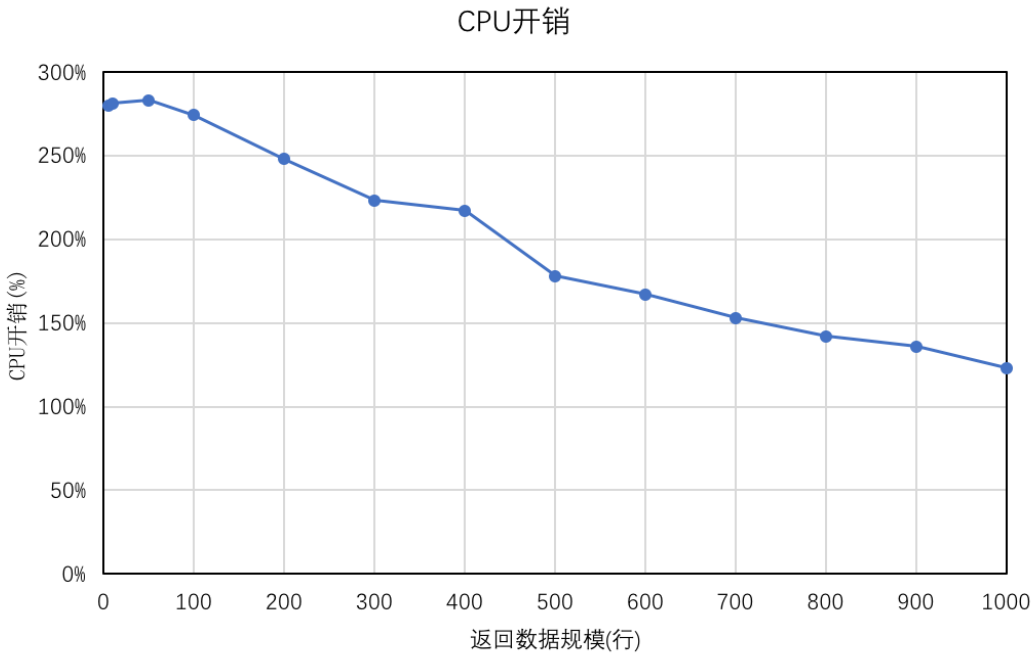
同时开启 15 个客户端，向具有 5000 万记录的表格进行范围查询(Range Query)请求，通过调整查询条件，让范围查询返回的结果集从 5 行到 1000 行之间改变。每行结果集大小为 32 字节。系统整体的查询性能和查询延迟如下所示。

结果集行数	吞吐量(qps)	延迟(latency)(ms)	CPU 开销
5	7,296.6000	2.0910	280%
10	7,467.7980	2.0110	281%
50	7,440.4755	2.0170	283%
100	7,204.2645	2.0821	274%
200	6,582.9900	2.2710	248%
300	5,883.8535	2.5487	223%
400	5,377.3080	2.7895	217%
500	4,828.2750	3.1067	178%
600	4,440.8505	3.3778	167%
700	4,107.1005	3.6522	153%
800	3,765.4485	3.9837	142%
900	3,437.7510	4.3675	136%
1000	3,250.6230	4.6145	123%



可以看到，范围查询的返回结果集<=100 行的时候，系统查询性能基本上处于相对稳定的状态，没有出现性能的衰减。当返回结果集>100 行的时候，吞吐量和延迟出现衰减。返回数据规模增大 200 倍，每次返回 1000 行记录(32,000 字节)的情况下，吞吐量下降约 50%，延迟增加 2.3 倍。如上可以看到 TDengine 对于规模变化的查询结果具有较好的响应稳定性，其衰减非常平缓。

同时记录的 CPU 开销如下：



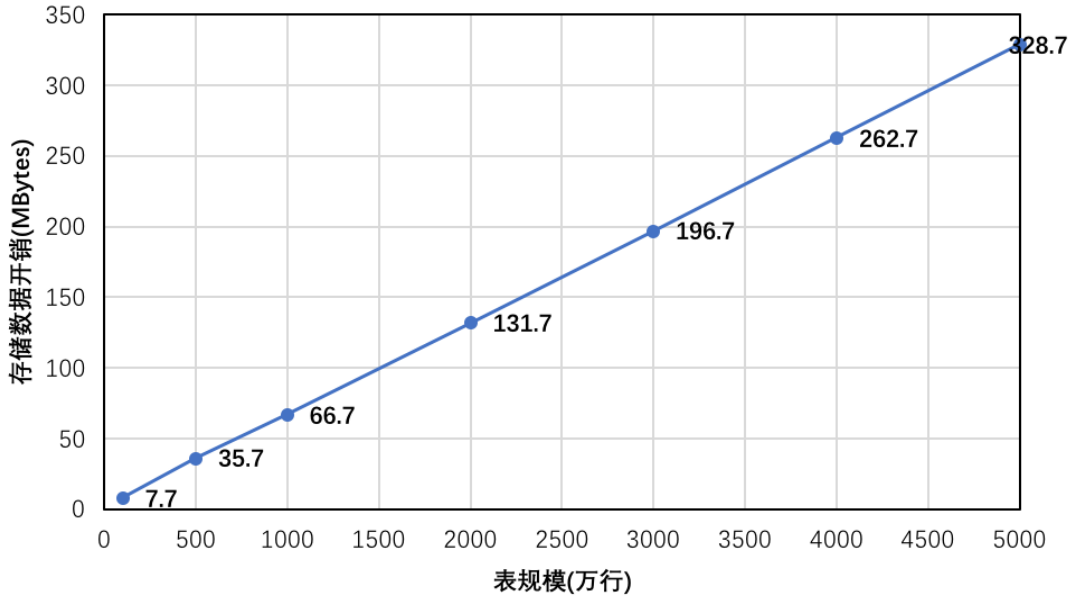
可以看出，随着查询返回的数据结果集规模的增加，系统吞吐量呈下降趋势，CPU 处理的开销在整个查询处理过程中的比重降低，从磁盘上读取数据并通过网络传输到客户端的过程是范围查询(Range Query)过程的主要开销，所以可以看到 CPU 的开销与系统的吞吐量呈

同步下降的趋势。

4.7 空间开销

针对不同行数的表(每行长度 32 字节)，查看磁盘占用空间。当表规模为 100 万时候，占用空间 7.7MB。当表规模到 5000 万时候，占用空间达到 328.7MB。整体上基本呈线性增长趋势。得益于 TDengine 的列数据压缩技术，可以看到随着表空间的增加，数据压缩比略有增加。具体数据如下：

记录数（万行）	占用空间(MBytes)
100	7.7
500	35.7
1,000	66.7
2,000	131.7
3,000	196.7
4,000	262.7
5,000	328.7



5 TDengine 集群性能测试

5.1 测试环境说明

系统部署在阿里云平台，数据节点及测试客户程序部署节点（ecs.c5.large）的配置如下：

CPU: 2 核 Intel Skylake Xeon Platinum 8163 2.5GHz

内存：4GB

内网带宽：1Gbps
内网每秒最大收发包：30 万 PPS
硬盘：40GB 高效云盘(非 SSD)，1240 IOPS
操作系统：Ubuntu 16.04 x64。

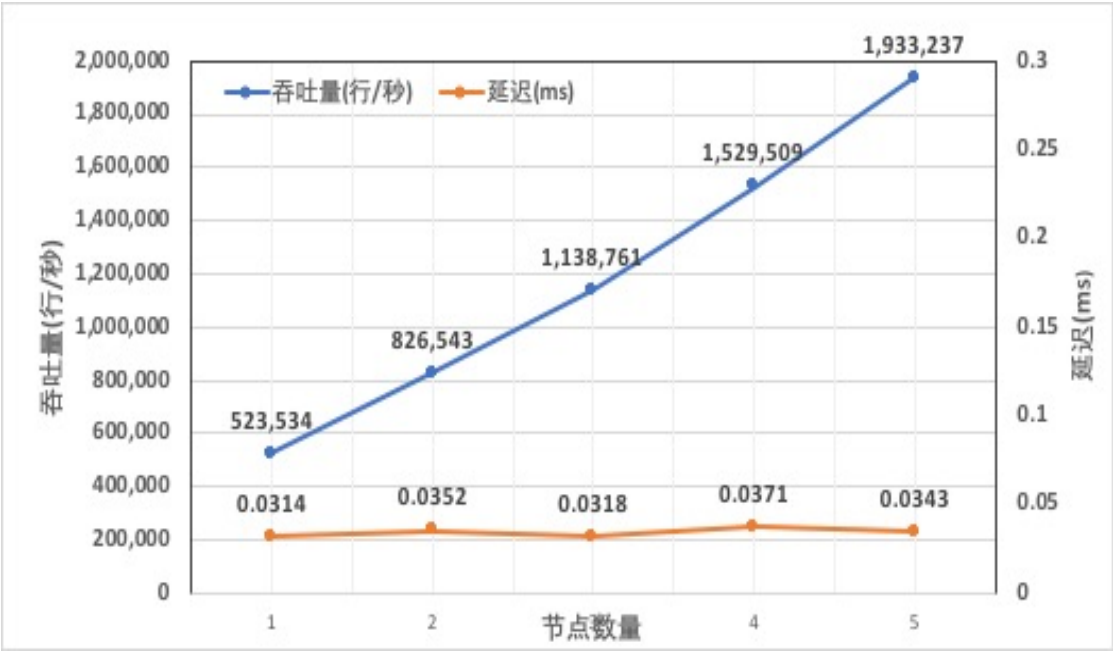
5.2 集群写入

5.2.1 异步批量写入

客户端节点采用异步写入方式，每次写入 4 行记录的批量写入模式，关闭 Naggle 开关，节点部署不进行任何优化，每个客户端程序连接系统后随机选择并连接系统中的数据库，向其中的单表连续写入数据。

客户端的进程的数量为服务节点 CPU 数量的 5 倍，客户端节点的数量与数据库节点数量保持为 1:1。结果如下：

节点数量	吞吐量(行/秒)	延迟(ms)	CPU 平均负载
1	523,534	0.0314	178%
2	826,543	0.0352	186%
3	1,138,761	0.0318	167%
4	1,529,509	0.0371	143%
5	1,933,237	0.0343	146%



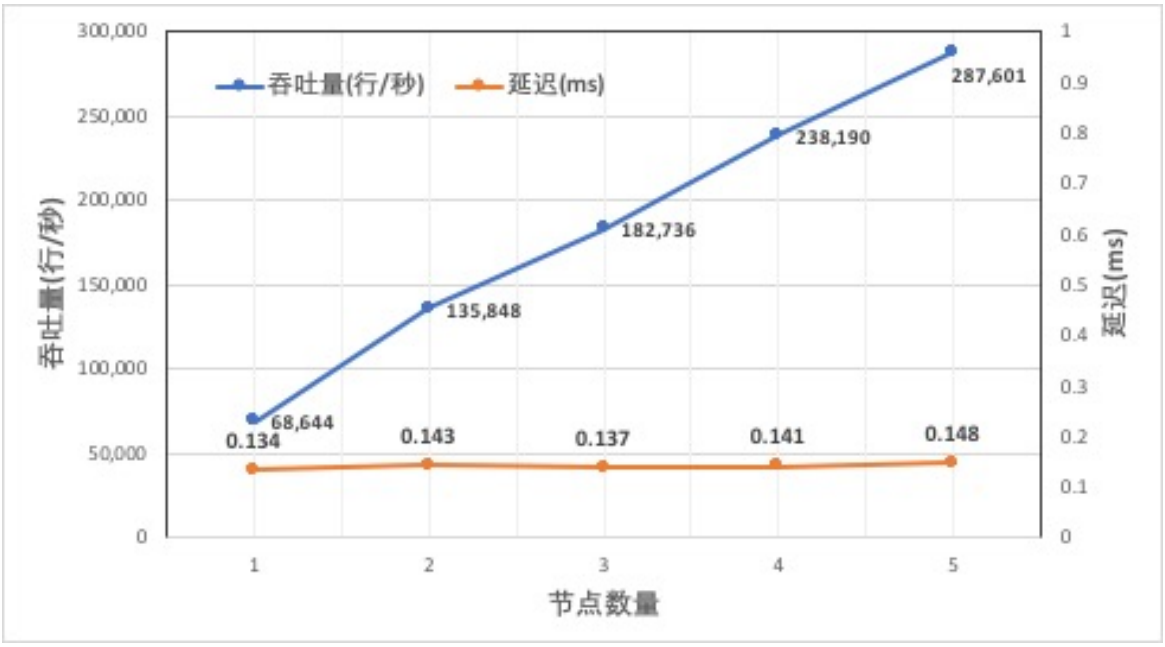
可以看到，随着节点数量的增加，集群系统写入性能基本保持线性增长的趋势，当节点数量达到 5 台的时候，集群提供了接近 200 万行/秒的写入性能。而系统的处理延迟基本保存

了稳定的状态，只在极小的范围内波动，并没有随着吞吐量的增加而增长。
单节点部署数据库系统的时候，采用的服务器整体配置和网络状况都更好，所以单节点表现出的系统性能更加突出，多节点的情况下，基本保持了线性增长能力。

5.2.2 同步写入

采用同步写入接口，一次向服务器写入一行记录，每行记录长度 32 字节，启动的客户端机器的数量与数据库集群的数量是 1:1，即当集群是单节点，启动客户机 1 台，当数据库集群达到 5 个节点，部署客户端节点的机器是 5 台。每台客户端机器，部署 10 客户端程序，随机选取集群中建立的数据库，向各自的表格中连续写入数据。结果如下：

节点数量	吞吐量(行/秒)	延迟(ms)
1	68,644	0.134
2	135,848	0.143
3	182,736	0.137
4	238,190	0.141
5	287,601	0.148



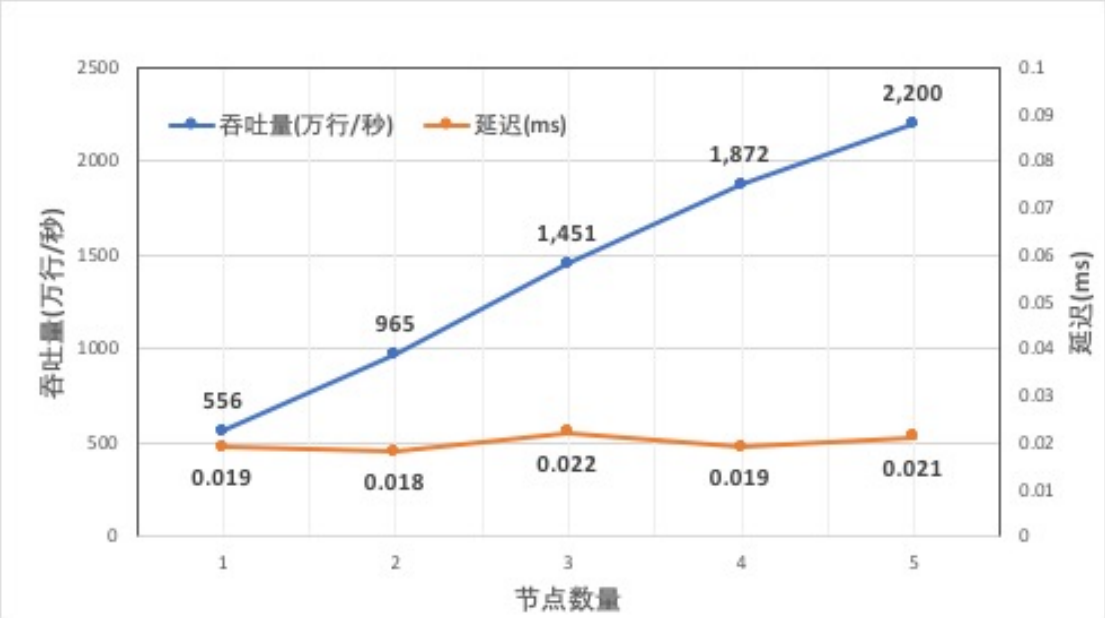
可以看出，单节点能够提供 6.86 万行/秒写入支持，当节点数量达到 5 的时候，写入数据达到 28.76 万行/秒，同时延迟基本上没有变化。整体系统对于连续的同步数据写入具有良好的横向支持和扩展能力。

5.3 集群读取

连续读取测试中，每台客户端机器启动 15 个客户端进程连续使用同步读取接口，随机从多

个数据库系统中进行连续的数据读取操作，每个表包含 1 千万数据记录，每行记录长度 32 字节。启动的客户端机器的数量与数据库集群的数量是 1:1，即当集群是单节点，启动客户机 1 台，当数据库集群达到 5 个节点，部署客户端程序的机器是 5 台。结果如下：

节点数量	吞吐量(万行/秒)	延迟(ms)
1	556	0.019
2	965	0.018
3	1,451	0.022
4	1,872	0.019
5	2,200	0.021



可以看出，读取的延迟保持了稳定，系统的吞吐量随着节点数量的增加，呈现了良好的线性增长的趋势。当节点数为 5 的时候，系统提供了 2200 万/秒的数据读取能力，同时保持了极低的响应延迟。

综上可以看到，不论是读取还是写入，TDengine 具有良好的横向扩展（scale-out）的能力，对于多节点构成的集群也能提供优秀的扩展支持。