**DolphinDB与InfluxDB性能对比测试报告**

1. **概述**

DolphinDB是以C++编写的一款分析型的高性能分布式时序数据库，使用高吞吐低延迟的列式存储引擎，集成了功能强大的编程语言和高容量高速度的流数据分析系统，可在数据库中进行复杂的编程和语言，显著减少数据迁移所耗费的时间。DolphinDB使用内嵌的分布式文件系统自动管理分区数据及其副本，为分布式计算提供负载均衡和容错能力。DolphinDB支持类标准的SQL，提供类似于Python的脚本语言对数据进行操作，也提供常用编程语言的API，在金融领域的历史数据分析建模与实时流数据处理，以及物联网中的海量数据处理和实时分析等场景中表现出色。

InfluxDB是目前最为流行的高性能开源时间序列数据库，由Go语言编写。它的核心是一款定制的存储引擎TSM Tree，对时间序列数据做了优化，优先考虑插入和查询数据的性能。InfluxDB使用类SQL的查询语言InfluxQL，并提供开箱即用的时间序列函数，同时对外提供基于HTTP的接口来支持数据的插入和查询。InfluxDB允许用户定义数据保存策略来实现对存储超过指定时间的数据进行删除或降采样，被广泛应用于存储系统的监控数据，IoT行业的实时数据等场景。

在本报告中，我们对DolphinDB和InfluxDB进行了性能对比测试，测试涵盖数据导入导出、数据查询、磁盘空间占用三个方面。测试数据集涵盖了日益流行的物联网数据集和规模更大的金融数据集。

在本次的所有测试中，DolphinDB表现更出色，主要结论如下：

* 数据导入方面，小数据集情况下DolphinDB的导入性能是InfluxDB的75倍，大数据集情况下DolphinDB的导入性能是InfluxDB的100倍，且InfluxDB原生不支持csv导入，需要手动转换Line Protocol格式。
* 数据导出方面，DolphinDB的性能是InfluxDB的11倍左右，且InfluxDB在导出大批量csv格式数据时容易发生内存溢出的问题。
* 磁盘空间占用方面，DolphinDB占用的空间总是小于等于InfluxDB占用的空间。
* 查询功能方面，InfluxDB不支持对比查询、表连接以及对time以外的tag和field进行排序，并且函数的参数只能是一个field，不能是field的表达式，功能上有极大的限制，有多个测试样例无法在InfluxDB中实现，而DolphinDB则没有这样的限制，对数据的处理更加灵活方便。
* 查询性能方面，DolphinDB在2个测试样例中性能超过InfluxDB 1000多倍，在6个测试样例中性能超过InfluxDB 50多倍，在2个测试样例中性能为InfluxDB的10多倍，其余测试样例性能也全部优于InfluxDB。

1. **测试环境**

由于InfluxDB集群版本闭源，本次测试中DolphinDB与InfluxDB均使用单机版本。

主机：DELL OptiPlex 7060

CPU ：Intel Core i7-8700（6 核 12 线程 3.20 GHz）

内存：32 GB （8GB × 4, 2666 MHz）

硬盘：2T HDD （222 MB/s 读取；210 MB/s 写入）

OS：Ubuntu 16.04 LTS

测试使用的DolphinDB版本为Linux 0.89，最大内存设置为28G。

测试使用的InfluxDB版本为1.7.5，根据InfluxDB官方配置文件中的说明，结合测试机器的实际硬件对配置做了优化，主要将wal-fsync-delay调节为适合机械硬盘的100ms，将cache-max-memory-size设置为28GB，将series-id-set-cache-size设置为400。

两者的具体配置见附录。

1. **测试数据集**

本报告测试了小数据量级（4.2GB）和大数据量级（270GB）下DolphinDB和InfluxDB的表现情况，以下是两个数据的表结构和分区方法：

1. **4.2GB设备传感器记录小数据集（csv格式，3千万条）**

我们使用物联网设备的传感器信息作为小数据集测试，测试集包含3000个设备在2016年11月15日到2016年11月19日10000个时间间隔上的传感器时间，设备ID，电池，内存，CPU等时序统计信息。数据集共30,000,000条数据，包含包含一张设备信息表device\_info和一张设备传感器信息记录表readings。

数据来源：<https://docs.timescale.com/v1.1/tutorials/other-sample-datasets>

下载地址：<https://timescaledata.blob.core.windows.net/datasets/devices_big.tar.gz>

以下是readings表在DolphinDB和InfluxDB中的结构：

|  |  |  |
| --- | --- | --- |
| **列名** | **DolphinDB数据类型** | **InfluxDB数据类型** |
| time | DATETIME | Timestamp(second precision) |
| device\_id | SYMBOL | tag |
| battery\_level | INT | field |
| battery\_status | SYMBOL | tag |
| battery\_temperature | DOUBLE | field |
| bassid | SYMBOL | tag |
| cpuavg1min | DOUBLE | field |
| cpuavg5min | DOUBLE | field |
| cpuavg15min | DOUBLE | field |
| mem\_free | LONG | field |
| mem\_used | LONG | field |
| rssi | SHORT | field |
| ssid | SYMBOL | tag |

我们在 DolphinDB 中的分区方案是将 time 作为分区的第一个维度，按天分为 4 个区，分区边界为 [2016.11.15 00:00:00, 2016.11.16 00:00:00, 2016.11.17 00:00:00, 2016.11.18 00:00:00, 2016.11.19 00:00:00]；再将 device\_id 作为分区的第二个维度，每天一共分 10 个区，最后每个分区所包含的原始数据大小约为 100 MB。

InfluxDB 中使用 Shard Group 来存储不同时间段的数据，不同 Shard Group 对应的时间段不会重合。一个 Shard Group 中包含了大量的 Shard， Shard 才是 InfluxDB 中真正存储数据以及提供读写服务的结构。InfluxDB 采用了 Hash 分区的方法将落到同一个 Shard Group 中的数据再次进行了一次分区，即根据 hash(Series) 将时序数据映射到不同的 Shard，因此我们使用以下语句手动指定每个 Shard Group 的 Duration，在时间维度上按天分区。

create retention policy one\_day on test duration inf replication 1 shard duration 1d default

1. **270GB股票交易大数据集（csv格式，23个csv，65亿条）**

我们将纽约证券交易所（NYSE）提供的 2007.08.01 - 2007.08.31 一个月的股市 Level 1 报价数据作为大数据集进行测试，数据集包含 8000 多支股票在一个月内的交易时间,股票代码,买入价,卖出价,买入量,卖出量等报价信息。

数据集中共有 65 亿（65,6169,3704）条报价记录，一个 CSV 中保存一个交易日的记录，该月共 23 个交易日，未压缩的 CSV 文件共计 270 GB。

数据来源：<https://www.nyse.com/market-data/historical>

以下是TAQ表在DolphinDB和InfluxDB中的结构：

|  |  |  |
| --- | --- | --- |
| **列名** | **DolphinDB数据类型** | **InfluxDB数据类型** |
| symbol | SYMBOL | tag |
| date | DATE | timestamp(second precision) |
| time | SECOND | 与date合并为一个字段 |
| bid | DOUBLE | field |
| ofr | DOUBLE | field |
| bidsiz | INT | field |
| ofrsiz | INT | field |
| mode | INT | tag |
| ex | CHAR | tag |
| mmid | SYMBOL | tag |

在 DolphinDB 中我们按date(日期)，symbol(股票代码)进行分区，每天再根据 symbol 分为 100 个分区，每个分区大概 120 MB 左右。

在InfluxDB中使用与小数据集相同的策略。

1. **数据导入导出测试**
2. **导入数据**

DolphinDB使用以下脚本导入：

timer {

for (fp in fps) {

loadTextEx(db, `taq, `date`symbol, fp, ,schema)

print now() + ": 已导入 " + fp

}

}

4.2 GB 设备传感器记录小数据集共3 千万条数据导入用时 20 秒, 平均速率 150 万条/秒。

270 GB 股票交易大数据集共 65 亿条数据（TAQ20070801 - TAQ20070831 23 个文件），导入用时 38 分钟 。

InfluxDB 本身不支持直接导入 CSV，只能通过 HTTP API 或者influx -import的方式导入，出于导入性能考虑，我们选择将 CSV 中的每一行先转换为 Line Protocol 格式，如：

readings,device\_id=demo000000,battery\_status=discharging,bssid=A0:B1:C5:D2:E0:F3,ssid=stealth-net battery\_level=96,battery\_temperature=91.7,cpu\_avg\_1min=5.26,cpu\_avg\_5min=6.172,cpu\_avg\_15min=6.51066666666667,mem\_free=650609585,mem\_used=349390415,rssi=-42 1479211200

并添加如下文件头：

# DDL

CREATE DATABASE test

CREATE RETENTION POLICY one\_day ON test DURATION INF REPLICATION 1 SHARD DURATION 1d DEFAULT

# DML

# CONTEXT-DATABASE:test

# CONTEXT-RETENTION-POLICY:one\_day

保存到磁盘中，再通过以下命令导入：

influx -import -path=/data/devices/readings.txt -precision=s -database=test

经过转换后，4.2 GB 设备传感器记录小数据集共3 千万条数据导入用时25 分钟 10 秒, 平均速率2 万条/秒 。

将 TAQ 数据插入到 InfluxDB 的过程中，如果多次插入时间相同 (1185923302)，tag 相同的记录 (这里是 symbol, mode, ex)，比如下面两条，即使 value 不同 (bid, ofr, bidsiz, ofrsiz)，后面的那一条记录也会覆盖前面的记录，最终数据库中只保留了最后一条记录。

taq,symbol=A,mode=12,ex=T bid=37,ofr=54.84,bidsiz=1,ofrsiz=1 1185923302

taq,symbol=A,mode=12,ex=T bid=37,ofr=38.12,bidsiz=1,ofrsiz=1 1185923302

要解决这个问题文档（[InfluxDB frequently asked questions | InfluxData Documentation](https://docs.influxdata.com/influxdb/v1.7/troubleshooting/frequently-asked-questions/" \l "how-does-influxdb-handle-duplicate-points)）里给出了两种方法，一种是新增一个 tag 来对相同时间的数据设置不同的 tag value 手动区分，另一种是强行微调时间戳使其不同。

设置不同的 tag value 手动区分这种方法只适用于数据完全按照时间顺序插入的情况。在使用其他的编程语言插入数据的过程中判断该条记录的时间戳是否与上一条记录完全相同，并在插入数据库时手动指定不同的 tag value 作区分，效率低下且操作繁琐；若数据并非完全按照时间顺序插入，则无法判断当前的时间点是否已经有数据记录的存在，是否会覆盖先前的数据。

我们在本次测试中使用强行微调时间戳的方法，由于原有 TAQ 交易记录的时间精度为秒，因此我们可以在将 CSV 的数据转换至 Line Protocol 格式的过程中，在原来精确到秒的时间戳的基础上，随机加上一个毫秒值，产生一个新的精度为毫秒的伪时间戳，以防止数据冲突。

经过转换后，270 GB 股票交易大数据集所包含的65 亿条数据导入用时65 小时，平均导入速率2.7 万条/秒。

导入性能对比如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集大小(GB)** | **DolphinDB** | **InfluxDB** | **DolphinDB/InfluxDB** |
| 4.2 | 150 万条/秒, 共 20 秒 | 2 万条/秒, 共 25 分钟 | 75 |
| 270 | 290 万条/秒, 共 38 分钟 | 2.7 万条/秒, 共 65 小时 | 98 |

结果显示，DolphinDB 的导入速率远大于 InfluxDB 的导入速率。在导入过程中还可以观察到，随着时间的推移，InfluxDB 的导入速率不断下降，而 DolphinDB 保持稳定。而且 InfluxDB 在导入数据时需要先编写代码将 CSV 格式文件转换为 InfluxDB 的 Line Protocol 格式，复杂繁琐，还会产生多余的中间文件占用大量空间。

1. **导出数据**

在 DolphinDB 中使用saveText((select \* from readings), '/data/devices/readings\_dump.csv')进行数据导出，用时仅需 28 秒。

在 InfluxDB 中若使用influx -database 'test' -format csv -execute "select \* from readings > /data/devices/export\_15.csv进行数据导出内存占用会超过 30 GB，最终引发fatal error: runtime: out of memory，最后采用分时间段导出 CSV 的方法。代码如下所示：

for i in 1{5..8}; do

time influx -database 'test' -format csv -execute "select \* from readings where '2016-11-$i 00:00:00' <= time and time < '2016-11-$((i+1)) 00:00:00'" > /data/devices/export\_$i.csv

done

总耗时5 min 31 s。

除性能差距悬殊之外，InfluxDB 的 CSV 数据导出操作复杂，容易发生内存溢出问题，而且导出的 CSV 文件首行无字段名称，用户体验远不如 DolphinDB。

导出性能对比如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集大小(GB)** | **DolphinDB** | **InfluxDB** | **DolphinDB/InfluxDB** |
| 4.2 | 107 万条/秒 共 28 秒 | 8.8 万条/秒 共 5 分钟 31 秒 | 11 |

1. **磁盘空间占用对比**

导入数据后，DolphinDB和InfluxDB占用空间如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集大小(GB)** | **DolphinDB(GB)** | **InfluxDB(GB)** | **DolphinDB/InfluxDB** |
| 4.2 | 1.2 | 1.2 | 1 |
| 270 | 51 | 110 | 2 |

小数据集中 DolphinDB 的空间利用率与 InfluxDB 相近，两款数据库都对数据进行了压缩存储，压缩率大致处于同一个数量级，在 20% - 30% 之间；大数据集中 InfluxDB 对数据的压缩效果不好，占用空间为 DolphinDB 的两倍。

1. **数据库查询性能测试**

我们一共对比了以下八种类别的查询：

* 点查询指定某一字段取值进行查询
* 范围查询针对单个或多个字段根据时间区间查询数据
* 精度查询针对不同的标签维度列进行数据聚合，实现高维或者低维的字段范围查询功能
* 聚合查询是指时序数据库有提供针对字段进行计数、平均值、求和、最大值、最小值、滑动平均值、标准差、归一等聚合类 API 支持
* 对比查询按照两个维度将表中某字段的内容重新整理为一张表格（第一个维度作为列，第二个维度作为行）
* 抽样查询指的是数据库提供数据采样的 API，可以为每一次查询手动指定采样方式进行数据的稀疏处理，防止查询时间范围太大数据量过载的问题
* 关联查询对不同的字段，在进行相同精度、相同的时间范围进行过滤查询的基础上，筛选出有关联关系的字段并进行分组
* 经典查询是实际业务中常用的查询

查询测试的时间包含磁盘 I/O 的时间，为保证测试公平，每次启动程序测试前均通过 Linux 系统命令sync; echo 1,2,3 | tee /proc/sys/vm/drop\_caches分别清除系统的页面缓存、目录项缓存和硬盘缓存，启动程序后依次执行所有查询语句执行一次。

4.2GB设备传感器记录小数据集查询测试结果如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **测试项目** | **DolphinDB(ms)** | **InfluxDB(ms)** | **InfluxDB/**  **DolphinDB** |
| 1 | 查询总记录数 | 2 | 6,700 | 3,350 |
| 2 | 点查询：按设备 ID 查询记录数 | 3 | 7 | 2 |
| 3 | 范围查询：单分区维度，查询某时间段内的所有记录 | 7 | 590 | 84 |
| 4 | 范围查询：多分区维度，查询某时间段内某些设备的所有记录 | 1 | 4 | 4 |
| 5 | 范围查询：分区及非分区维度，查询某时间段内某些设备的特定记录 | 3 | 19 | 6 |
| 6 | 精度查询：查询各设备在每5分钟内的内存使用量最大值、最小值之差 | 65 | 10,500 | 162 |
| 7 | 聚合查询：单分区维度，计算设备电池的最高温度 | 25 | 1,400 | 56 |
| 8 | 聚合查询：多分区维度，计算各时间段内设备电池的平均温度 | 602 | 25,000 | 42 |
| 9 | 对比查询：对比10个设备24小时中每小时平均电量的变化情况 | 2 | 不支持该功能 |  |
| 10 | 关联查询：等值连接，查询连接某个WIFI的所有设备的幸好 | 73 | 不支持表连接 |  |
| 11 | 关联查询：左连接，列出所有的WIFI及其连接设备的型号、系统版本，并去除重复条目 | 5 | 不支持表连接 |  |
| 12 | 关联查询：笛卡尔积 | 261 | 不支持表连接 |  |
| 13 | 关联查询：全连接 | 1,800 | 不支持表连接 |  |
| 14 | 经典查询：计算某时间段内高负载高电量设备的内存大小 | 15 | 1,6000 | 1,067 |
| 15 | 经典查询：统计连接不同网络的设备的平均电量和最大、最小电量，并按平均电量降序排列 | 59 | 不支持对tag, field排序 |  |
| 16 | 经典查询：查找所有设备平均负载最高的时段，并按照负载降序排列、按照时间升序排列 | 32 | 不支持对tag, field排序 |  |
| 17 | 经典查询：计算各个时间段内某些设备的总负载，并在各时段内按总负载降序排列 | 3 | 不支持对tag, field排序 |  |
| 18 | 经典查询：查询充电设备的最近20条电池温度记录 | 2 | 27 | 14 |
| 19 | 经典查询：未在充电、电量少于33%、平均1分钟内最高负载的5个设备信息 | 96 | 不支持表连接 |  |
| 20 | 经典查询：某两个型号的设备每小时最低电量的前20条数据 | 70 | 不支持in关键字 |  |

查询脚本见附录。

结果显示，DolphinDB的查询性能远超于InfluxDB。在功能上，InfluxDB不如DolphinDB强大，比如：

* InfluxDB不支持对比查询和表连接，无法完成许多常规SQL数据库支持的查询。
* InfluxDB 不支持对除 time 以外的 tag, field 进行排序，即不能select \* from taq order by <some-field>，详见<https://github.com/influxdata/influxdb/issues/3954>。
* InfluxDB中函数的参数只能是某一个 field ，而不能是 field 的表达式（field1 + field2等）因此只能用 subquery 先计算出表达式的值再套用函数，非常繁琐，而且需要在子查询和父查询的 where 子句中重复指定时间范围，否则会从最旧的数据记录的时间开始一直扫描到当前时间。 InfluxDB 和 DolphinDB 第14个查询语句的对比如下：

//14. 经典查询：计算某时间段内高负载高电量设备的内存大小

//DolphinDB

select max(date(time)) as date, max(mem\_free + mem\_used) as mem\_all

from readings

where

time <= 2016.11.18 21:00:00,

battery\_level >= 90,

cpu\_avg\_1min > 90

group by hour(time), device\_id

//InfluxDB

select max(mem\_total)

from (

select mem\_free + mem\_used as mem\_total

from readings

where

time <= '2016-11-18 21:00:00' and

battery\_level >= 90 and

cpu\_avg\_1min > 90

)

where time <= '2016-11-18 21:00:00'

group by time(1h), device\_id

270GB股票交易大数据查询测试结果如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **测试项目** | **DolphinDB(ms)** | **InfluxDB(ms)** | **InfluxDB/**  **DolphinDB** |
| 1 | 点查询：按股票代码、时间查询 | 738 | 909 | 1.2 |
| 2 | 范围查询：查询某时间段内的某些股票的所有记录 | 1,000 | 结果错误 |  |
| 3 | 范围查询：按股票代码、日期过滤，按卖出与买入价格差降序排列，并取前1000条记录 | 375 | 不支持对tag, field排序 |  |
| 4 | 聚合查询：单分区维度，查询每分钟的最大卖出报价、最小买入报价 | 184 | 302 | 1.6 |
| 5 | 聚合查询：多分区维度，按股票代码分组查询每分钟的买入报价标准差和买入数量总和，并按股票代码和时间排序 | 62 | 不支持对tag, field排序 |  |
| 6 | 经典查询：按多个股票代码、日期、时间范围和报价范围过滤，查询股票代码、时间、买入价、卖出价 | 16 | 17 | 1 |
| 7 | 经典查询：按日期、时间范围、卖出买入价格条件、股票代码过滤，查询各个股票每分钟的平均变化幅度 | 16,000 | 3,060,000 | 191 |
| 8 | 经典查询：计算某天每个股票每分钟最大卖出报价与最小买入报价之差 | 8,100 | 不支持多个selector |  |
| 9 | 经典查询：按股票代码、日期范围、时间范围过滤，查询每天某时段内每分钟均价 | 63 | 4,500 | 71 |
| 10 | 经典查询：按日期范围、时间范围过滤，查询每个股票每天的均价 | 16,000 | 3,300,000 | 206 |
| 11 | 经典查询：计算某日期范围内有成交记录的各个股票每天的加权平均价格，并按日期、股票代码排序 | 4,200 | 不支持对tag, field排序 |  |

查询脚本见附录。

结果显示，某些查询，两者的性能差别不大，但某些查询，DolphinDB比InfluxDB快将近100到200倍。

在测试中，我们发现：

* InfluxDB在第2个查询中，对于where条件中选择的多个非连续的时间分区返回的结果为空，而不是各个时间分区的结果的总和。InfluxDB中第2个查询的代码如下所示：

//2. 范围查询：查询某时间段内的某些股票的所有记录

select symbol, time, bid, ofr

from taq

where

(symbol = 'IBM' or symbol = 'MSFT' or symbol = 'GOOG' or symbol = 'YHOO') and

('2007-08-03 01:30:00' <= time and time < '2007-08-03 01:30:59')

//该语句返回的结果不为空

select symbol, time, bid, ofr

from taq

where

(symbol = 'IBM' or symbol = 'MSFT' or symbol = 'GOOG' or symbol = 'YHOO') and

(('2007-08-03 01:30:00' <= time and time < '2007-08-03 01:30:59') or ('2007-08-04 01:30:00' <= time and time < '2007-08-04 01:30:59'))

//扩展了时间范围后，该语句返回的结果反而为空

* 在InfluxDB中无法完成第8个查询。DolphinDB的第8个查询的代码如下所示：

//8. 经典查询：计算某天每个股票每分钟最大卖出与最小买入价之差

select symbol, max(ofr) - min(bid) as gap

from taq

where

'2007-08-03' <= time and time < '2007-08-04' and

bid > 0 and

ofr > bid

group by symbol, time(1m)

InfluxDB会抛出异常，ERR: mixing multiple selector functions with tags or fields is not supported，即不能在select语句中同时使用max和min函数，而DolphinDB可以正常执行。

* InfluxDB 对时间进行 group by 之后返回的结果包含了所有的时间段，即使当前时间段内无有效数据也返回（如下所示），对于稀疏数据增加了处理复杂度且降低性能，而DolphinDB只返回包含有效数据的时间段。

2007-07-31T23:02:00Z 22.17 54.84

2007-07-31T23:03:00Z

2007-07-31T23:04:00Z

2007-07-31T23:05:00Z

2007-07-31T23:06:00Z

2007-07-31T23:07:00Z

2007-07-31T23:08:00Z 37 38.12

2007-07-31T23:09:00Z

2007-07-31T23:10:00Z 37.03 38.12

1. **其他方面比较**

DolphinDB 除了在基准测试中体现出优越的性能之外，还具有如下优势：

* 语言上，InfluxDB 通过 InfluxQL 来操作数据库，这是一种类 SQL 语言；而 DolphinDB 内置了完整的脚本语言，不仅支持 SQL 语言，而且支持命令式、向量化、函数化、元编程、RPC 等多种编程范式，可以轻松实现更多的功能。
* 数据导入方面，InfluxDB 对于特定文件格式数据例如 CSV 文件的批量导入没有很好的官方支持，用户只能通过开源第三方工具或自己实现文件的读取，规整为 InfluxDB 指定的输入格式，再通过 API 进行批量导入。单次只能导入 5000 行，不仅操作复杂，效率也极其低下；而DolphinDB提供了ploadText、loadText、loadTextEx函数，可以直接在脚本中导入CSV文件，对用户更加友好，并且效率更高。
* 功能上，InfluxDB不支持表连接，部分常规查询无法完成；而DolphinDB不仅支持常用的表连接功能，还对asof join和window join等非同时连接做了很多性能优化。InfluxDB对时间序列的分组（group by）的最大分组是星期（week）；而DolphinDB支持对所有内置时间类型进行分组，最大单位为月（month）。InfluxDB在查询中函数的参数只能是某一个字段，不能是字段的表达式，而DolphinDB无此限制。
* DolphinDB提供了600多个内置函数，可满足金融领域的历史数据建模与实时流数据处理以及物联网领域中的实时监控与实时分析处理等不同场景的需求，并且大部分聚合函数、处理时序数据需要的领先、滞后、累计窗口、滑动窗口函数都做了性能优化。
* DolphinDB 的集群版本支持事务，而且在一个分区的多个副本写入时，保证强一致性。

**附录**

**数据预览（取前20行）**

* [readings](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/readings.csv)
* [readings(Line Protocol)](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/readings.lineprotocol.txt)
* [readings表转换为Line Protocol脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/convert_small.coffee)
* [TAQ](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/TAQ.csv)
* [TAQ(Line Protocol)](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/TAQ.lineprotocol.txt)
* [TAQ表转换为Line Protocol脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/convert_big.coffee)

**DolphinDB**

* [安装、配置、启动脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/test_dolphindb.sh)
* [配置文件](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/dolphindb.cfg)
* [小数据集测试脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/test_dolphindb_small.txt)
* [大数据集测试脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/test_dolphindb_big.txt)

**InfluxDB**

* [安装、配置脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/test_influxdb.sh)
* [修改配置](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/influxdb.conf)
* [小数据集测试脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/test_influxdb_small.sql)
* [大数据集测试脚本](https://github.com/dolphindb/Tutorials_CN/blob/master/benchmark/DolphinDB%20vs%20InfluxDB/test_influxdb_big.sql)