TDEngine

1、rpm包安装

rpm包下载：点此链接下载

如下安装操作在root用户下进行：

```
rpm -iv TDengine-server-2.0.10.0-Linux-x64.rpm
```

卸载命令：

```
rpm -e tdengine
```

TDengine成功安装后，主安装目录是/usr/local/taos，目录配置如下说明：

自动生成配置文件目录、数据库目录、日志目录。

```
配置文件缺省目录：/etc/taos/taos.cfg， 软链接到/usr/local/taos/cfg/taos.cfg；
数据库缺省目录：/var/lib/taos， 软链接到/usr/local/taos/data；
日志缺省目录：/var/log/taos， 软链接到/usr/local/taos/log；
/usr/local/taos/bin目录下的可执行文件，会软链接到/usr/bin目录下；
/usr/local/taos/driver目录下的动态库文件，会软链接到/usr/lib目录下；
/usr/local/taos/include目录下的头文件，会软链接到到/usr/include目录下；
```

可以在/usr/local/taos/cfg，编辑taos.cfg文件配置data、log目录，并重新创建软连接，指向新路径

2、TDengine集群配置

2.1、在第一个节点使用root用户或sudo通过如下命令启动tdengine：

```
systemctl start taosd

#补充：
#查看运行状态：
systemctl status taosd
#关闭运行：
systemctl stop taosd
```

2.2、启动后，请执行taos, 启动taos shell，在shell命令行输入"show dnodes;"查看集群节点

2.3、在其他节点也通过rpm安装tdengine，但不要启动

2.4、编辑除第一个节点外的其他节点配置文件：

```
vim /usr/local/taos/cfg/taos.cfg
```

2.5、将其他节点的firstEp全部配置成第一个节点的ip和port

```
firstEp  master01:6030
```

通过命令启动各节点的tdengine：

```
systemctl start taosd
```

2.6、在第一个节点，使用CLI程序taos, 登录进TDengine系统, 使用命令将各节点加入集群：

```
CREATE DNODE "slave02:6030";
CREATE DNODE "slave04:6030";
CREATE DNODE "slave05:6030";
CREATE DNODE "slave06:6030";
CREATE DNODE "slave07:6030";
```

3、vnode高可用

tdengine的taos数据库root登录默认账户和密码如下：

```
taos -u root -p taosdata
```

vnode副本策略实现数据冗余，确保数据安全性，在建库时指定：

```
#集群的节点数必须大于等于副本数
#ntables 数据库中表的数量
#vgroups vnode组数量
#replica 数据副本数，可在建库时指定
#quorum
#days 指定几天的数据保存一个数据块，可在建库时指定
#keep 指定数据保存时长，过期数据会从磁盘删除，可在建库时指定
#cache 内存块的大小，单位为兆字节（MB），可在建库时指定
#blocks    每个VNODE（TSDB）中有多少cache大小的内存块，可在建库时指定
#sion
#update 是否开启数据库支持更新相同时间戳数据，可在建库时指定
#status 数据库是否可用
create database if not exists train_realtime_status replica 2 KEEP 365 DAYS 10 BLOCKS
4 UPDATE 1;
```

5、数据查询

除了常规的聚合查询之外，还提供针对时序数据的窗口查询、统计聚合等功能。

5.1、单表查询

SQL语句的解析和校验工作在客户端完成。解析SQL语句并生成抽象语法树(Abstract Syntax Tree, AST)，然后对其进行校验和检查。以及向管理节点(mnode)请求查询中指定表的元数据信息(table metadata)。

根据元数据信息中的End Point信息，将查询请求序列化后发送到该表所在的数据节点（dnode）。

5.2、按时间轴聚合、降采样、插值

针对具有时间戳数据在时间轴上进行聚合，与流计算引擎的窗口查询相似。

在TDengine中引入关键词interval来进行时间轴上固定长度时间窗口的切分，并按照时间窗口对数据进行聚合，对窗口范围内的数据按需进行聚合。例如：

```
select count(*) from d1001 interval(1h);
```

针对d1001设备采集的数据，按照1小时的时间窗口返回每小时存储的记录数量。

在需要连续获得查询结果的应用场景下，如果给定的时间区间存在数据缺失，会导致该区间数据结果也丢失。TDengine提供策略针对时间轴聚合计算的结果进行插值，通过使用关键词Fill就能够对时间轴聚合结果进行插值。例如：

```
select count(*) from d1001 interval(1h) fill(prev);
```

针对d1001设备采集数据统计每小时记录数，如果某一个小时不存在数据，这返回之前一个小时的统计数据。TDengine提供前向插值(prev)、线性插值(linear)、NULL值填充(NULL)、特定值填充(value)。

5.3、多表聚合查询

多表聚合查询与单表查询的整体流程相同，但是存在如下的差异：

- 由于多表可能分布在不同的节点(dnode)，因此多表的聚合查询需要首先获得表所在的全部数据节点的信息，并且同时向相关的dnode发出查询请求。
- 每个vnode的计算获得的中间结果(partial results)需要进行第二阶段的聚合才能形成最终结果，第二阶段的聚合过程在客户端完成。
- 由于表标签信息存储在vnode中，因此针对标签信息的查询也需要vnode完成。客户端将标签的过滤表达式封装在查询请求结构体中发送给vnode，由vnode的查询执行线程从中抽取出标签查询条件，然后执行查询。标签查询与过滤是在针对表的查询之前完成。标签查询完成以后，将符合条件的表纳入到接下来的查询处理流程中。

5.4、预计算

为有效提升查询处理的性能，针对物联网数据的不可更改的特点，在数据块头部记录该数据块中存储数据的统计信息：包括最大值、最小值、和。

5.5、高效写入数据

TDengine支持多种接口写入数据，包括SQL, Prometheus, Telegraf, EMQ MQTT Broker, CSV文件等，后续还将提供Kafka, OPC等接口。数据可以单条插入，也可以批量插入，可以插入一个数据采集点的数据，也可以同时插入多个数据采集点的数据。支持多线程插入，支持时间乱序数据插入，也支持历史数据插入。

5.6、SQL写入

Tips:

- 要提高写入效率，需要批量写入。一批写入的记录条数越多，插入效率就越高。
- TDengine支持多线程同时写入，要进一步提高写入速度，一个客户端需要打开20个以上的线程同时写。

6、sql使用实例

6.1、建库

创建一个库时，除SQL标准的选项外，应用还可以指定保留时长、副本数、内存块个数、时间精度、文件块里最大最小记录条数、是否压缩、一个数据文件覆盖的天数等多种参数

```
CREATE DATABASE train_realtime_status KEEP 365 DAYS 10 REPLICA 3 BLOCKS 4;
```

上述语句将创建一个名为train_realtime_status的库，这个库的数据将保留365天（超过365天将被自动删除），每10天一个数据文件，副本数为3, 内存块数为4。

6.2、创建超级表

```
create table if not exists train_signal (ts timestamp, p_id int, value double) TAGS
('train_id' binary(10), 'v_id' int)

create table nlcj_t1_v0 USING train_signal TAGS ('NLCJ_T1',1)
```

6.3、查询语句

```
#查询2020-08-12 08:00:00 ~ 2020-08-13 08:00:00 所有列车v_id=100的每1m采样一次的信号平均值
select avg(value) from train_signal where v_id=100 and ts>1597190400000 and ts
<1597276800000 interval(1m) group by train_id;
```