
数据技术嘉年华
// Data Technology Carnival
开源 · 融合 · 数智化 — 引领数据技术发展 释放数据要素价值
中国DBA联盟
All China DBA Union
墨天轮

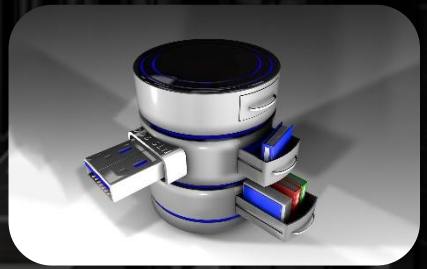
数据技术嘉年华

将流式计算引入时序数据库，TDengine 3.0的分布式架构实践

演讲人：廖浩均
涛思数据

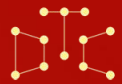
概述

TDengine 3.0 首次在数据库服务中内置了全新的轻量级流计算引擎。流计算引擎采用事件驱动模式运行，使用SQL-like语言与用户交互，面向写入 TDengine 时序数据库中的时序数据提供流计算服务。本次分享将介绍利用 TDengine 的分布式数据库基础设施基础，搭建面向特定领域和计算需求的轻量级、高性能的流计算引擎。



目录

CONTENTS



时序数据特点

1 产生的数据带有时间戳

大部分数据都按照时间顺序递增到达，但有不确定比例的乱序数据

2 包含数值（测量值）的结构化数据

数据模型相对固定，不升级固件基本不会发生变动

3 数据流量巨大且平稳可预测，可较精确估算数据流规模

基本不存在数据产生速率的抖动

4 一个时间序列的数据源唯一确定

不同数据源（设备）产生数据的速率无直接无关联或协同效应

5 数据价值随着时间流逝而递减

数据的价值随时间增加规整的递减，并最终被清理掉

6 数据很少有更新或删除

绝大部分数据产生以后即为最终状态，不再变动

时序数据应用需求

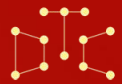
- 时序数据典型的应用场景

典型应用场景	解决策略	Cons
数据流复现或重放	手动创建数据流，系统从创建开始记录数据写入行为，之后完整重放该流程	无法涵盖创建时刻之前的数据重放
定时统计聚合	Continuous query（连续查询，一种定时拉起执行的查询处理功能，在给定时间窗口关闭后才开始计算该时间窗口的结果）	计算行为与数据写入行为无关联，无法处理时间窗口关闭后到达的乱序数据，因此结果可能会不准确
异常检测	Continuous query 或 standing query（异常检测触发的频率）	为及时发现异常，需要较高频率反复查询，浪费计算资源

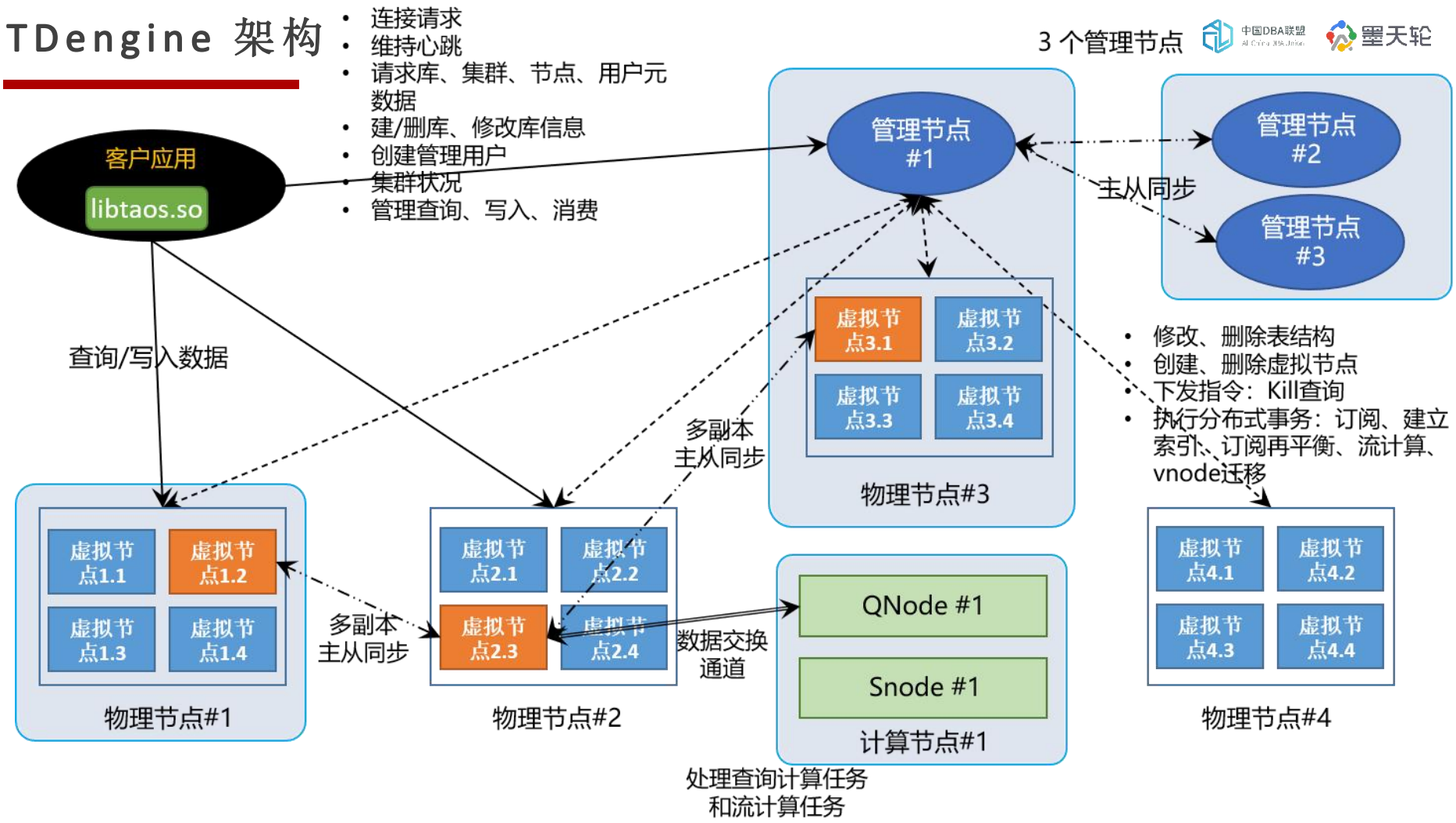
在时序数据库内置消息队列和轻量级的流计算引擎来更好地满足上述场景的应用需求

目录

CONTENTS

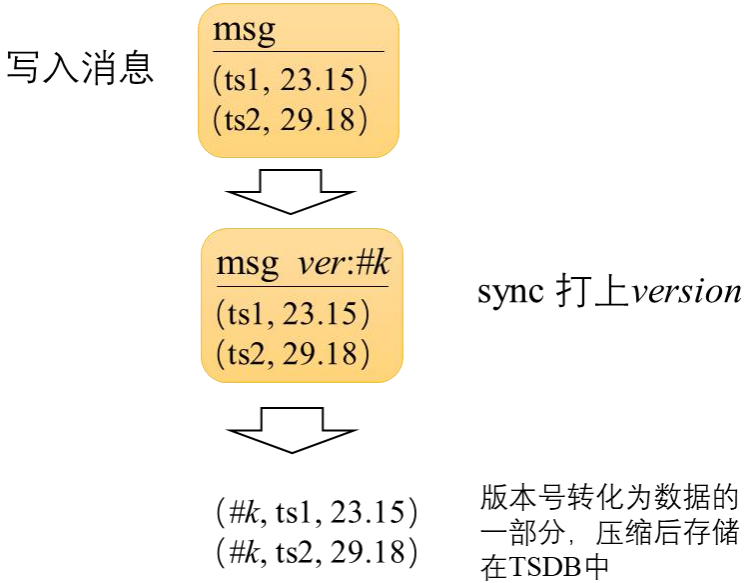


TDengine 架构


连接请求
维持心跳
请求库、集群、节点、用户元数据
建/删库、修改库信息
创建管理用户
集群状况
管理查询、写入、消费
3 个管理节点
客户应用
libtaos.so
管理节点 #1
管理节点 #2
管理节点 #3
主从同步
查询/写入数据
虚拟节点3.1
虚拟节点3.2
虚拟节点3.3
虚拟节点3.4
物理节点#3
修改、删除表结构
创建、删除虚拟节点
下发指令：Kill查询
执行分布式事务：订阅、建立索引、订阅再平衡、流计算、vnode迁移
多副本主从同步
虚拟节点1.1
虚拟节点1.2
虚拟节点1.3
虚拟节点1.4
物理节点#1
多副本主从同步
虚拟节点2.1
虚拟节点2.2
虚拟节点2.3
虚拟节点2.4
物理节点#2
数据交换通道
QNode #1
Snode #1
计算节点#1
处理查询计算任务和流计算任务
虚拟节点4.1
虚拟节点4.2
虚拟节点4.3
虚拟节点4.4
物理节点#4

TDengine时序数据版本化

- TDengine 提供数据版本化机制
  - 针对预写日志（WAL）记录的所有消息赋予版本号（version），版本号单调递增
  - 数据库支持批量写逻辑，单个消息支持多条记录。版本号以消息为单位，不以记录为单位。同一个消息中的多条记录共享同一个版本号
  - 数据版本化在 Vnode 级别生效，不同 Vnode 的版本号之间没有关联，不提供全局版本号



TDengine时序数据版本化

- 时序数据写入是 Append 操作，结果上看是 Upsert 操作
  - 所有时序数据都追加（Append）写入 TSDB
    - TSDB 内存中数据采用跳表组织
    - TSDB 磁盘文件采用混合结构组织，LSM + 整合（Packed）的时序数据。混合文件结构是在超大规模时间线与快速读取需求之间进行平衡（可以通过 DB级别的参数进行控制）
  - Upsert 逻辑位于数据读取模块
    - 不存在相同时间戳数据，新写入数据就是插入（insert）操作
    - 存在写入的时间戳，新写入的数据更新（update）已经存在的数据
  - 对于一个时间线，时序数据写入行为是幂等操作
    - 每行记录的时间戳是其唯一标识，重复写入时间戳不会对最后结果有任何影响
    - 重复写入的相同时间戳的数据具有不同的版本号

TDengine时序数据版本化

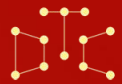
- 数据读取模块具备读取特定版本（范围）数据的功能
  - TSDB 提供按照版本号读取数据功能，任何一次查询范围查询均包含时间范围和版本范围两个维度
  - WAL 模块具备索引，可按照版本号快速进行寻址功能，以及顺序读取功能

- (T1,#1)+(T1,#3)=>T1
- (T2,#4) =>T2

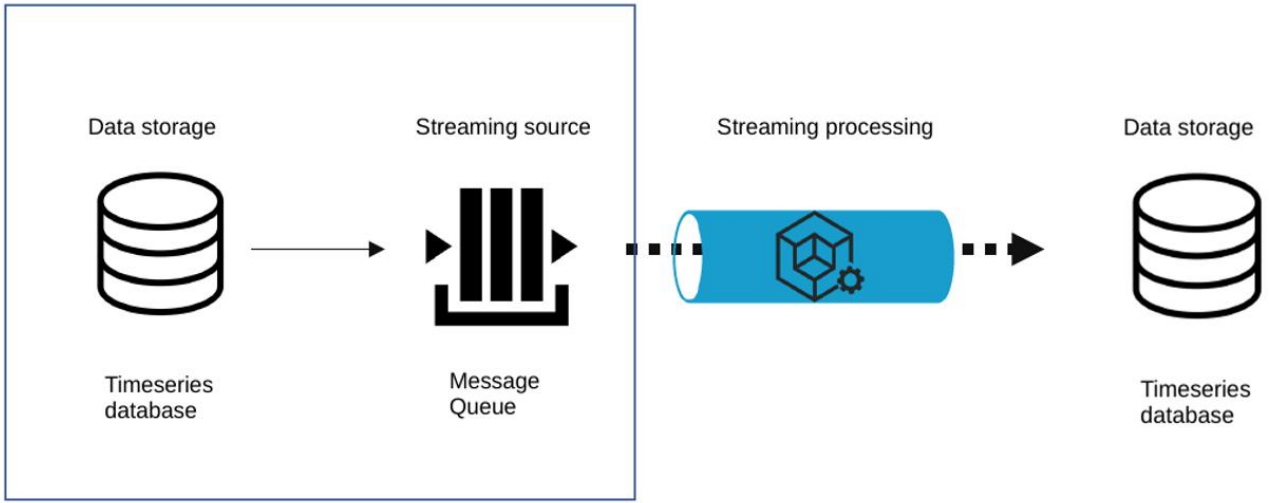
多版本合并过程中，高版本数据能够覆盖低版本数据

目录

CONTENTS



时序数据库中的流计算引擎架构



source

sink

时序数据流计算引擎架构—数据源

- 时序数据存储（TSDB）
  - TDengine中时序数据存储模块，负责将时序数据安全、可靠、高性能组织和存储
  - 磁盘中时序数据按照时间线和主时间戳（primary timestamp)进行组织
  - 同一个时间序列的数据记录按照时间递增顺序连续存放，按列组织，按类型压缩
  - 数据生存周期通过数据库参数 Keep 进行控制

- 预写日志（WAL）
  - 数据按照写入顺序记录在预写日志（WAL）中，因此具备**保证顺序**的特点
  - 由于保证了写入顺序，可以作为流数据处理的基础数据源
  - WAL 使用 wal_retention 控制其最长保留数据时间

时序数据流计算引擎架构—数据源

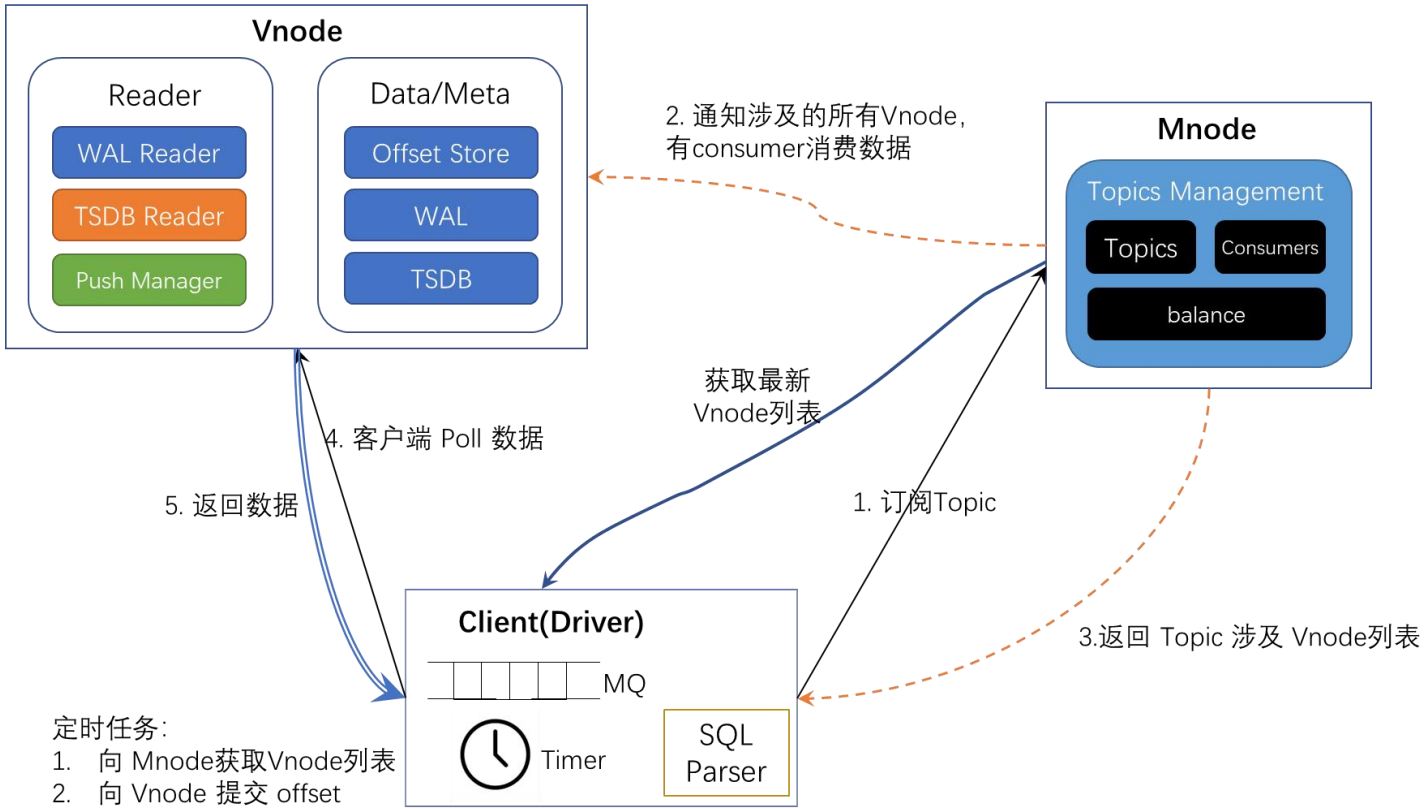
- 最直接处理流程中，写入 Vnode 的数据，在写入处理线程中，会自动复制一份数据，然后将其引用放入每个流计算任务的输入队列中，然后拉起流计算查询处理线程进行处理。
- 在下述几种情况下需要时序数据文件和预写日志。

类型	使用方式
时序数据文件	1. 计算窗口内数据出现更新之后，需要重新计算时间窗口的结果。此时需要从TSDB中将给定时间窗口和版本范围数据读取出来进行重新计算。 例如：Avg(col_1)，时间窗口内多条记录的 col_1 列出现更新，需要读取窗口内全部数据重新进行计算
预写日志	1. 流计算节点出现反压（流计算任务的输入队已满，无法再放入新数据），此时数据推送行为需要暂停。无法直接复制写入的数据，重试的时候需要从 WAL 中上次中断的 version 处扫描WAL来生成数据 2. 流计算恢复的时候，从WAL 中获得检查点（check point）以后的数据来重放流计算过程进行流计算的恢复

时序数据流计算引擎架构—消息队列

- 消息队列架构图

将 WAL 的读取能力外部化，辅以必要的支撑功能，加上消息队列操纵的语义和 API。



时序数据流计算引擎架构—消息队列技术特点

- 不提供 exactly once 的语义保证。在手动提交模式下，提供数据 at least once delivery
- 提供 Vnode 级别数据顺序保证，不提供全局的数据顺序保证
- 能够订阅到时序数据，还能够消费到元数据的变动（创建表、更改表模式、更新标签等）
- 支持多副本环境下运行，Vnode 的 leader 宕机后，待新的leader选举出来后，可以自动切换到新的leader中继续订阅消费
- Driver（客户端）层
  - 提供手动提交和自动提交两种模式
  - 采用单线程执行搭配队列，可以确保每个 Vnode 内数据严格有序到达
  - 客户端采用无锁模式运行

时序数据流计算引擎架构—消息队列用途

taosX 中间件是 TDengine 时序数据整体解决方案中用于跨集群数据同步（复制）的系统。

taosX 中间件可向客户提供企业级的灾备服务和双（多）活解决方案。

时序数据流计算引擎架构—消息队列高可用设计

- 集群多副本下的高可用设计

  1. 只有 Vnode 的 leader 能够接受并处理订阅请求
  2. 订阅者消费点位（offset） 存储在 Vnode，并通过 sync 模块在 leader/follower 之间同步
  3. Leader 宕机以后，transporter 层自动转入循环重试等待状态，待剩余 followers 中重新选举出新 leader，然后 transporter 层自动将消费请求重定向到新 leader，该过程对订阅客户端（driver）透明
  4. 新leader 在WAL中定位到（offset）指定的位置，然后开始返回数据

- 消费点位在服务端的多副本可靠存储和leader崩溃后的重试充分利用现有框架的能力，不需要重新实现（步骤 2 和步骤 3）。
- 分布式存储和计算基础架构支持下，消息队列的实现逻辑不需要关注集群和分布式的场景。

时序数据流计算引擎架构—消息队列语法

- 消息队列的订阅，对象可以是普通表、超级表或则整个数据库

```
create topic [topic_name#1]
    as
    select <col_1>,<col_2>
    from [table_name]
```

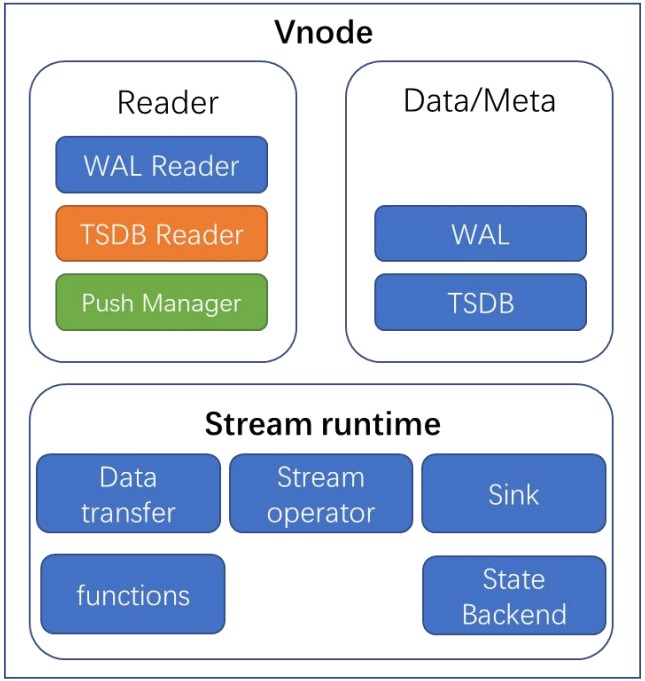
- 创建一个名为 topic_name#1，该topic 的源是 table_name
- 订阅该表，则该表任何**数据写入**会推送到该topic 的订阅者

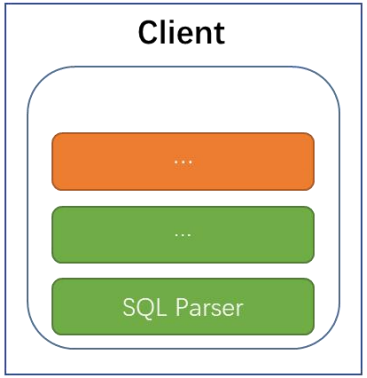
```
create topic [topic_name#2]
    as
    <Db_name>
```

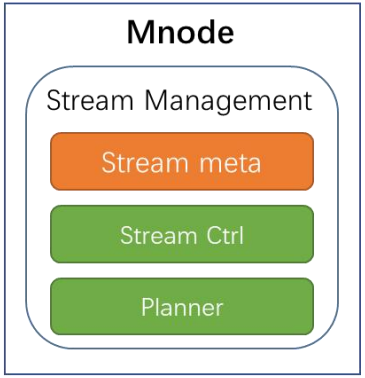
- 创建一个名为 topic_name#2，该topic 的源是 db_name
- 订阅该数据库，则该数据库的**任何变动**会以元数据+时序数据的形式返回。

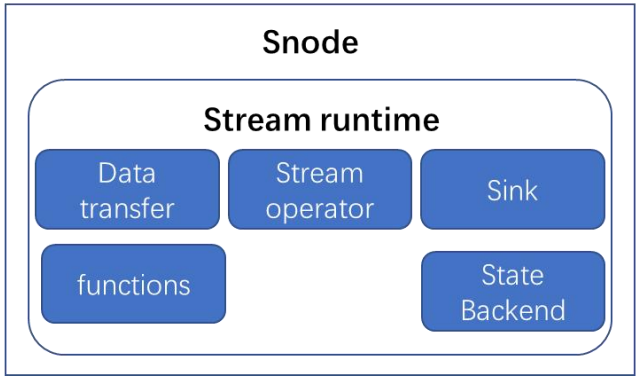
时序数据流计算引擎架构—流计算引擎

- 流计算引擎的整体框架



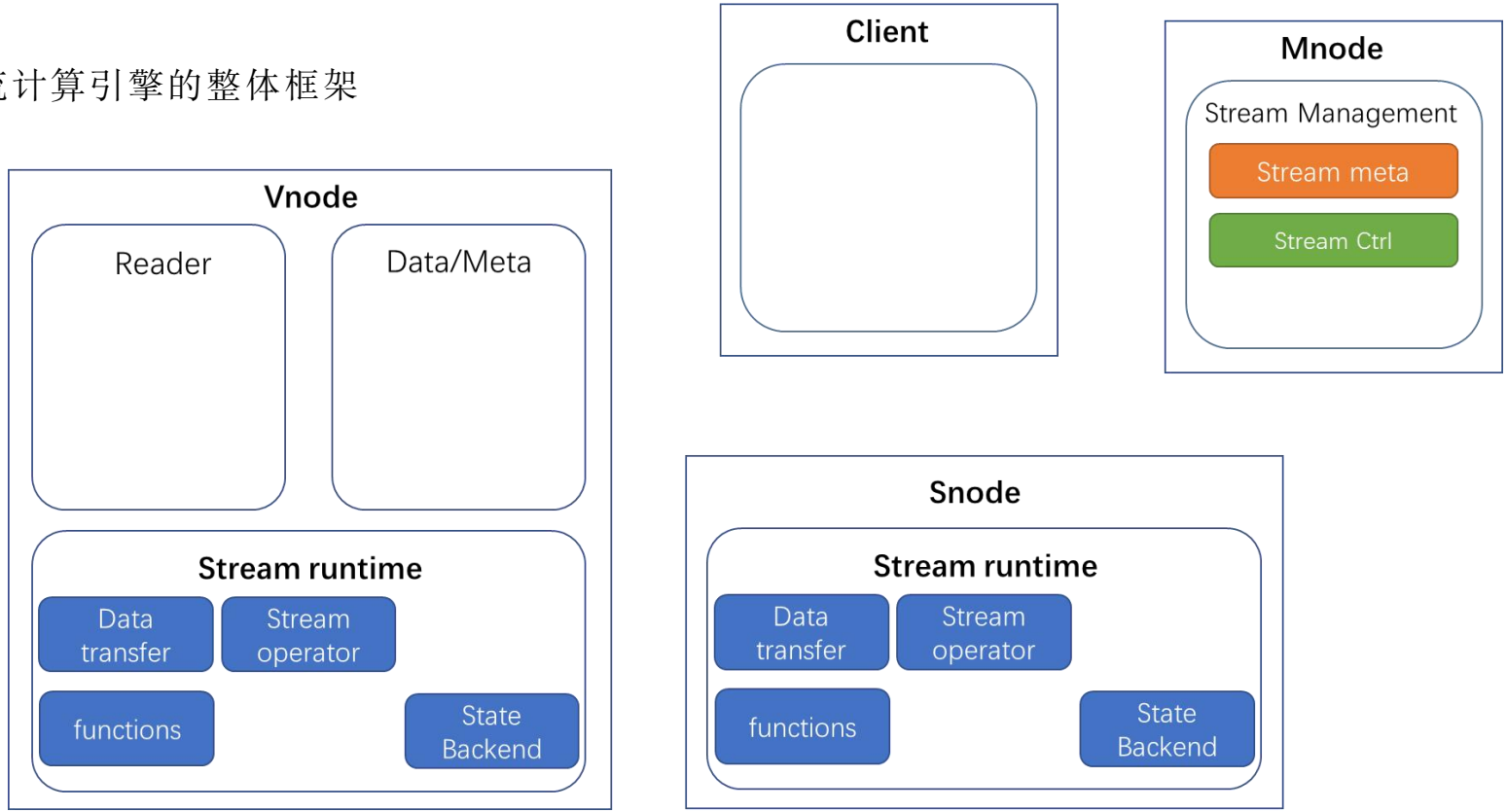






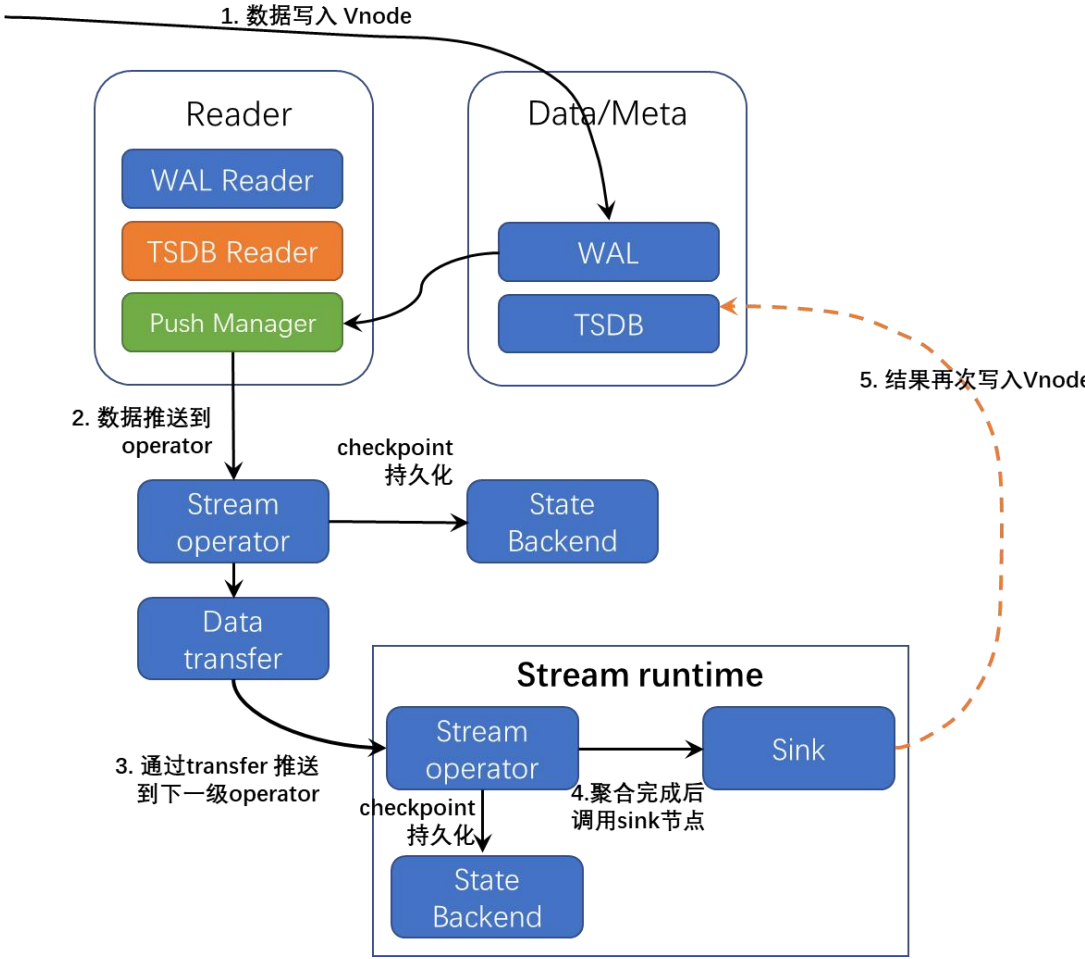
时序数据流计算引擎架构—流计算引擎

- 流计算引擎的整体框架



时序数据流计算引擎架构—流计算引擎

- 流计算引擎的Stream Runtime
  - 数据写入Vnode
  - Push Manager 复制写入消息，然后将其引用放入每个流计算任务的输入队列
  - 执行器执行完成以后，推送到下一级stream task进行聚合处理
  - 最后聚合的结果回写TSDB（有可能是本地也有可能是远程的 TSDB）



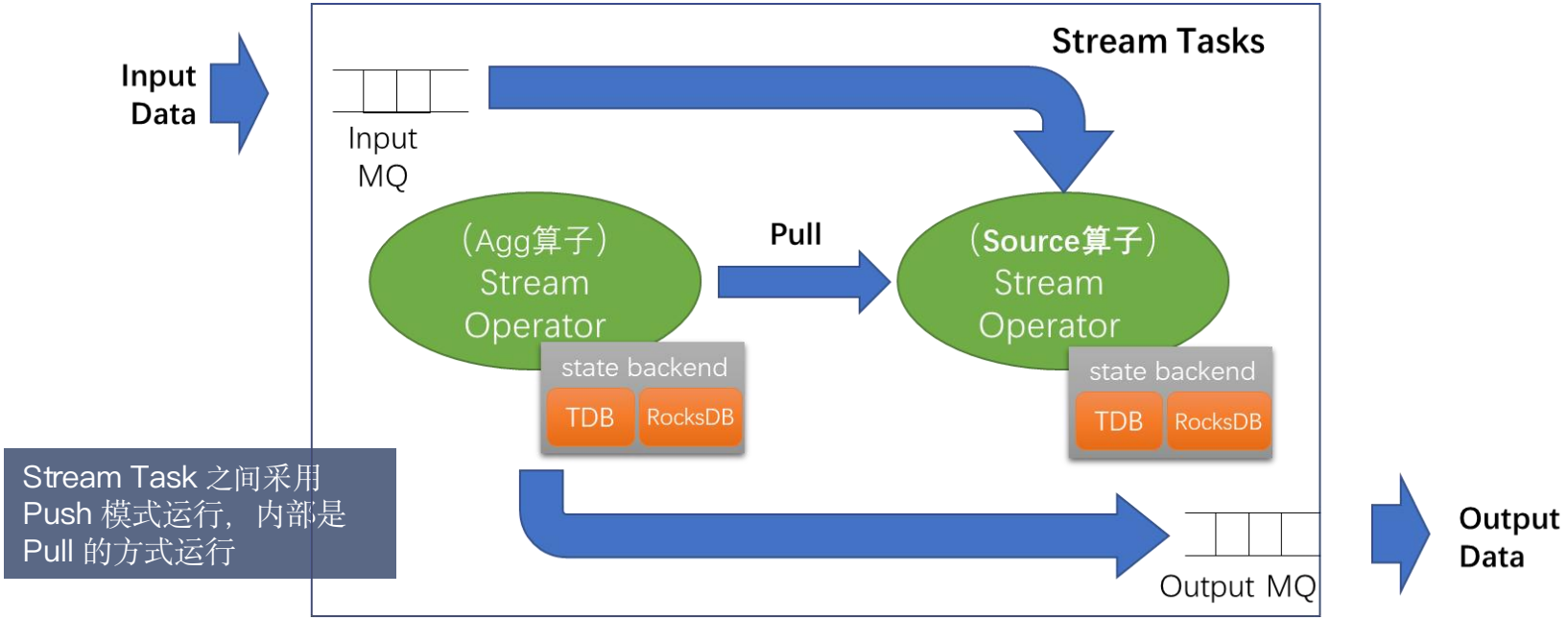
时序数据流计算引擎架构—流计算算子

- Stream operator
  - 总计 34 个算子，部分算子共享使用，例如：Project Operator 等
  - 5个窗口相关的算子提供流计算版本
- 流算子具有不同的计算（推送）结果的策略
  - AT ONCE：数据写入立即更新，实时推送更新结果
  - WINDOW CLOSE：数据写入，等待时间窗口关闭后，才会推送结果
  - MAX DELAY：在流事件中断或延迟不确定的情况下，窗口即使未关闭，经过一段时间后，仍然推送结果

流算子	普通查询算子
StreamIntervalOperator	IntervalOperator
StreamPartitionOperator	PartitionOperator
StreamSessionAggOperator	SessionAggOperator
StreamStateAggOperator	StateAggOperator
StreamFillOperator	FillOperator

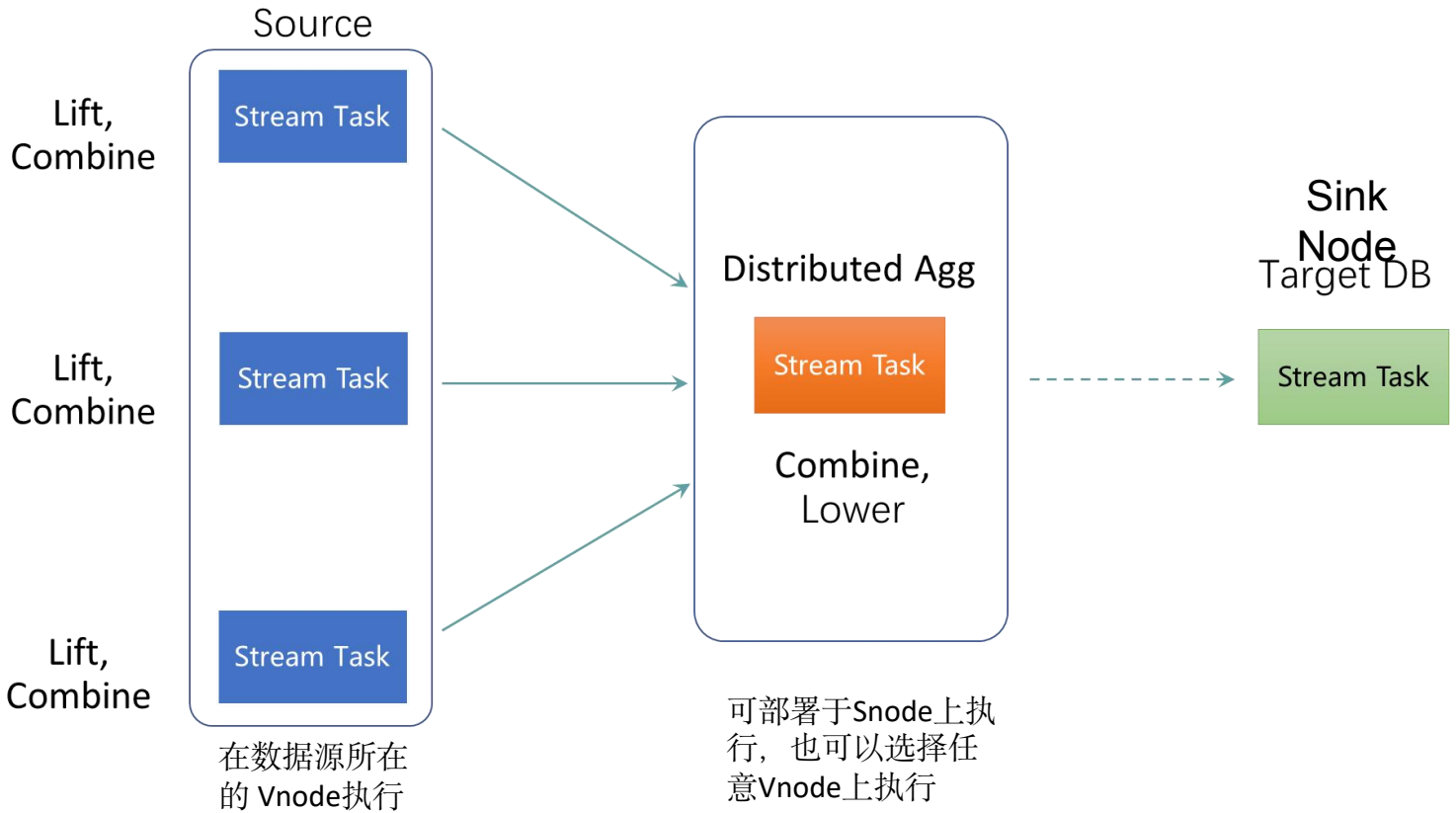
时序数据流计算引擎架构—执行任务

- 在 TDengine 中，普通查询使用向量化的 Pull 模式执行
- Task 内部，复用了数据库普通查询的算子整体架构， 因此流计算的算子之间也采用了 Pull 模式调用
- Task 之间，采用 input/output 队列来保证数据处理严格顺序性

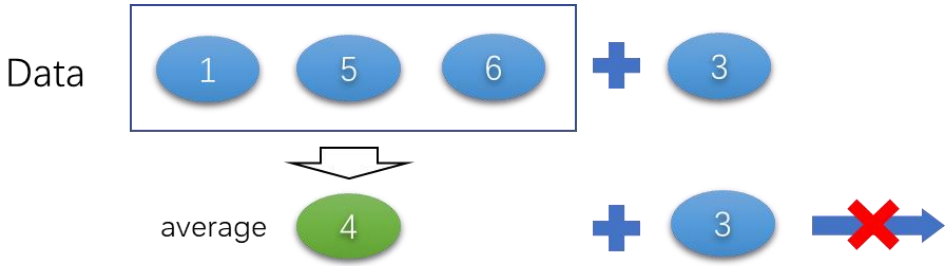


时序数据流计算引擎架构—两级任务执行模式

- 两级执行序列



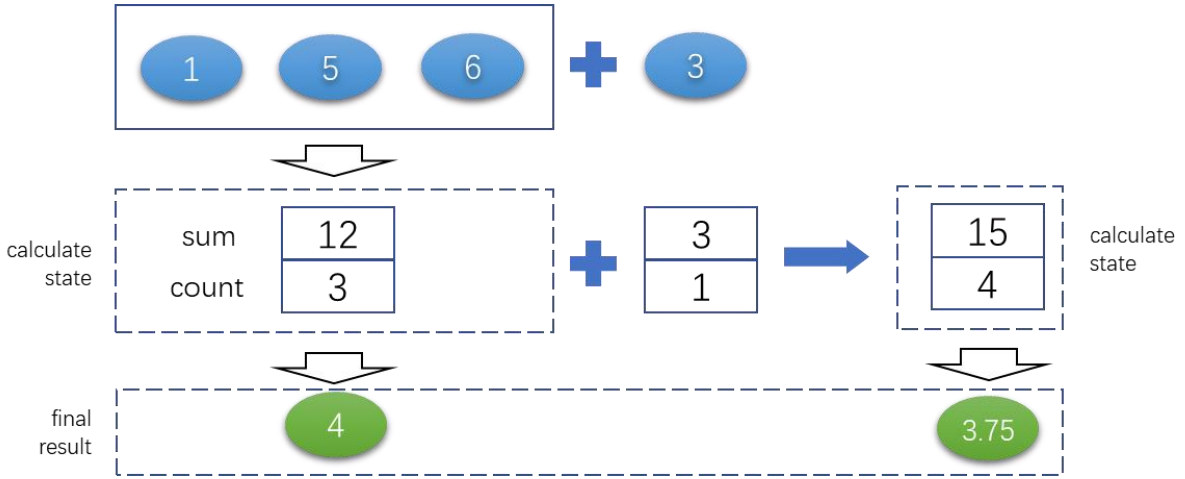
时序数据流计算引擎架构—增量计算



非增量计算

增量计算

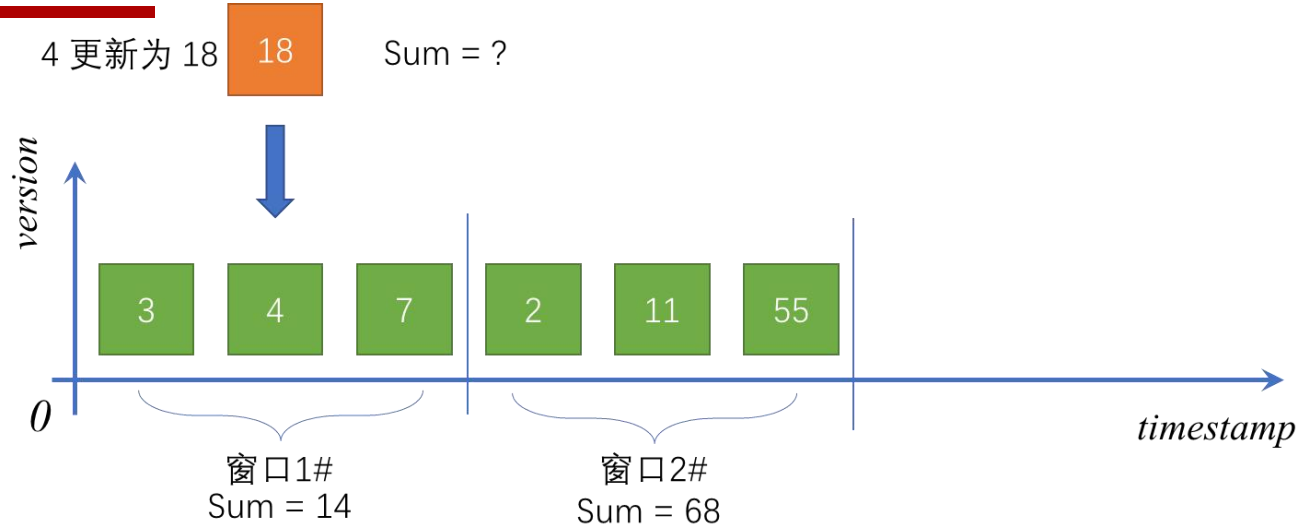
批查询中实现的所有的函数
均采用增量计算的计算模式



时序数据流计算引擎架构—检查点

- 检查点机制依赖于状态存储系统提供的能力
- Stream operator消费完 Input 队列后，会在最后自动插入ck。通过调用state storage 的后端 commit 机制进行增量式状态存储
- TDB是 TDengine 内置的 B+Tree 结构的K/V存储系统
- 不同 Vnode 运行的 stream task 生成的 checkpoint 和 commit 操作相互独立

时序数据流计算引擎架构—更新处理



- 对于每个窗口，建立 bloom filter。
- 首先通过 bloom filter 判断是否是更新的数据，如果不是，直接增量计算。
- 如果是更新，调用 TSDB 读取时间窗口范围内的所有数据，重新进行计算。
- 对于部分函数（sum/avg/stddev），定义了invert函数，将该计算转化为：

  Result – oldValue + new_value.

时序数据流计算引擎架构—SQL语法

- SQL 语句

```
CREATE STREAM [IF NOT EXISTS]
<stream_name>
  [TRIGGER at_once]
  [WATERMARK 300s]
    INTO <dst_table_name>
    AS
      SELECT _wstart AS start, min(c1)
      FROM <source_table_name>
      PARTITION BY tbname
      INTERVAL(10s)
```

- 定义流的 Source：

  FROM `source_table_name`

- 定义流的 Sink：

  INTO `dst_table_name`

- 窗口结果更新以后，立即推送聚合结果：

  TRIGGER `at_once`：窗口结果更新以后立即推送结果

  WATERMARK `300s`：容忍 300s 乱序（窗口会依据事件时间延迟关闭 300s）

- PARTITION BY TBNAME：超级表中的每个子表各自聚合，写入 `source_table_name(超级表)` 的不同子表中

- 定义聚合的时间窗口：

  `INTERVAL(10s)`：长度为 10 秒钟的翻转时间窗口

时序数据流计算引擎特点

- 架构特点
  - 写入时序数据库中的数据均可作为计算数据源
  - （写入）事件驱动的计算行为，非定时任务方式执行
  - 分布式执行，充分利用集群资源
  - 二次聚合的计算资源弹性扩展，与存储解耦
- 功能层面
  - 轻量级，面向时序数据，非通用场景的流计算处理
  - 与普通时序数据查询相同的时间窗口聚合计算
  - 支持与普通查询相同的聚合函数
  - 提供消息队列能力，在Vnode级别，按照数据写入的顺序重放数据序列
  - 计算过程兼容数据乱序和数据更新
  - 计算结果需要回写数据库，此时也可以下一级流式计算的新数据源
- 用户层面
  - 与普通查询相同的用户界面，使用类 SQL 语言定义流计算行为
  - 支持 UDF，通过UDF拓展流计算的功能

时序数据流计算引擎特点

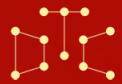
- 用户层面
  - 与普通查询相同的用户界面，使用类 SQL 语言定义流计算行为
  - 支持 UDF，通过UDF拓展流计算的功能
- 架构特点
  - 写入时序数据库中的数据均可作为计算数据源
  - （写入）事件驱动的计算行为，非定时任务方式执行
  - 分布式执行，充分利用集群资源
  - 二次聚合的计算资源弹性扩展，与存储解耦

时序数据流计算引擎特点

- 功能层面
  - 轻量级，面向时序数据的非通用场景的流计算处理
  - 与普通时序数据查询相同的时间窗口聚合计算能力
  - 支持与普通查询相同的聚合函数
  - 提供消息队列能力，（在Vnode级别）按照数据写入的顺序重放数据序列
  - 计算过程兼容数据乱序和数据更新
  - 计算结果需要回写数据库，此时也可以下一级流计算的数据源

目录

CONTENTS



时序数据流计算引擎演进路线

- 功能方面
  - 提供可配置的检查点机制
    - 检查点的生成可以通过配置来控制
  - 查询执行的动态优化策略
    - 将普通查询中的计算优化策略引入到流计算中，提升计算效率，降低计算开销
  - 支持非时间窗口形式的聚合计算
  - 流计算的结果可 sink 到其他的系统或应用中
- 架构方面
  - 支持多级 stream task 调度执行
    - 拓展两级 task 执行的架构，能够扩展流计算的能够支持的计算的能力和嵌套计算能力
    - Source 层的流计算任务可以与数据源（Vnode）解耦合，实现彻底的计算与存储分离
  - 将任务内部流计算算子之间的处理流程采用 Push 模式重构

总结

- 在TDengine 3.0中，首次整合并提供了消息队列的功能和轻量级流计算引擎。这也是在分布式数据库系统中搭建流数据处理系统的首次尝试
- 充分利用分布式数据库已有的高可用/多副本同步模块、SQL解析及计划生成、事务处理模块、内置计算函数、以及算子框架等基础模块，极大地降低消息队列和流计算系统开发的难度和复杂程度
- 内置的消息队列和流计算系统均只面向时序数据提供服务。针对时序数据的应用特点，在系统功能特征和应用边界上，进行了取舍
- 虽然是非通用的消息队列和流计算引擎，但是极大地拓展了 TDengine 时序数据库的服务能力，可有效简化用户时序数据整体部署的复杂程度和应用维护的难度
- 上述介绍的消息队列系统和流计算引擎在开源社区和商业用户中均得到了使用，后续我们将继续完善其功能，提升其性能

总结

www.github.com/taosdata/TDengine

TDengine 3.0 于 2022.08 发布，核心代码全部开源

社区版开源

2019.07.12

21,100+ Star

4,600+ Fork

3,600+ Issue

集群版开源

2020.08.03

谢谢观看

THANKS FOR WATCHING