Rust China Conf 2020

Shenzhen, China

2020conf.rustcc.cn

使用 Rust 构建高性能时序数据库 CeresDB

熙凯 2020.12
xikai.wxk@antgroup.com
蚂蚁集团技术风险部

目录

背景

1.1 什么是时序数据

1.2 CeresDB 的由来

1.3 CeresDB 的技术栈

时序数据

时序数据

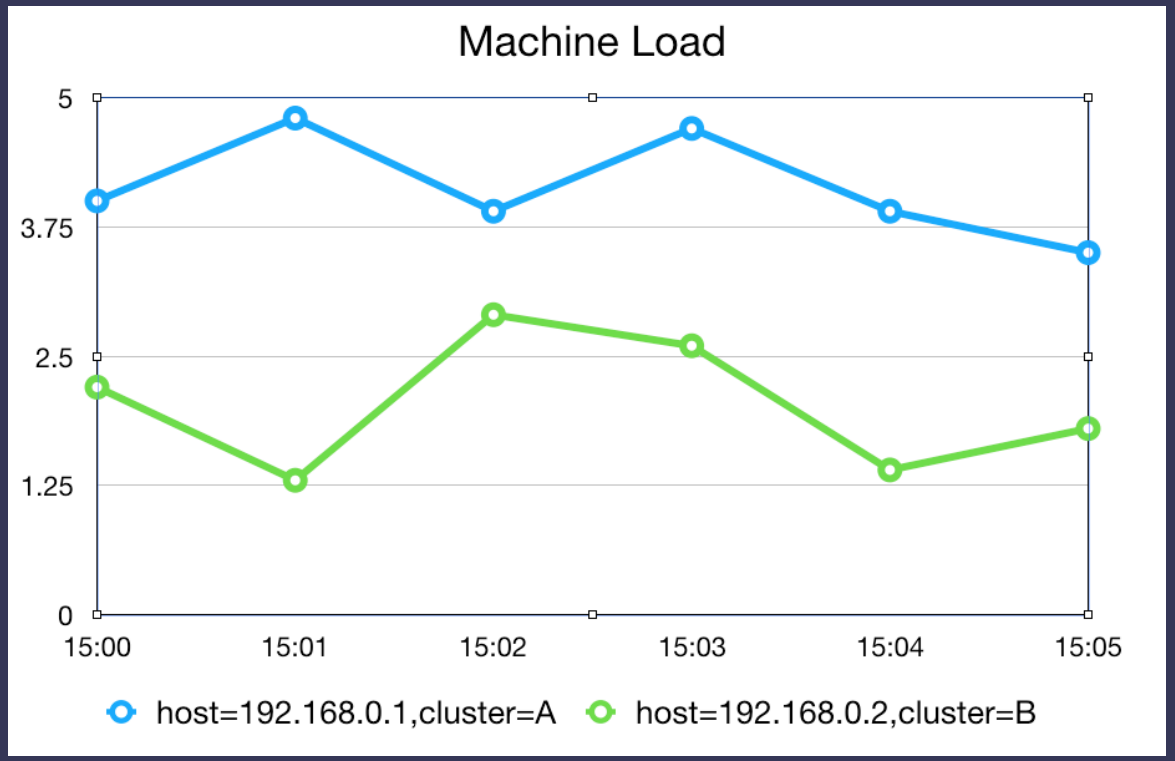
基于时间的一系列数据点的集合。

数据特点

- 写入持续、平稳、高吞吐、少更新
- 写多读少，查询最近的数据

场景

服务健康状态检测、IOT、证券交易等



timestamp	host	cluster	value(load)
2020-06-26 15:00	192.168.0.1	A	4.0
2020-06-26 15:01	192.168.0.1	A	4.8
2020-06-26 15:00	192.168.0.2	B	2.2
2020-06-26 15:01	192.168.0.2	B	1.3

背景

1.1 什么是时序数据

1.2 CeresDB 的由来

1.3 CeresDB 的技术栈

CeresDB 由来

问题

为小规模场景提供高可用的时序存储服务

最初方案

基于 InfluxDB（1.4.x) 来构建集群版本的时序存储服务

优势

- 小规模场景下, 快速构建, 提供时序存储服务

弊端

- 无法满足日益增长的业务需求（**性能**）
- 异常情况难以定位原因（**稳定性**）
- 集群方案过于简陋（**高可用**）

背景

1.1 什么是时序数据

1.2 CeresDB 的由来

1.3 CeresDB 的技术栈

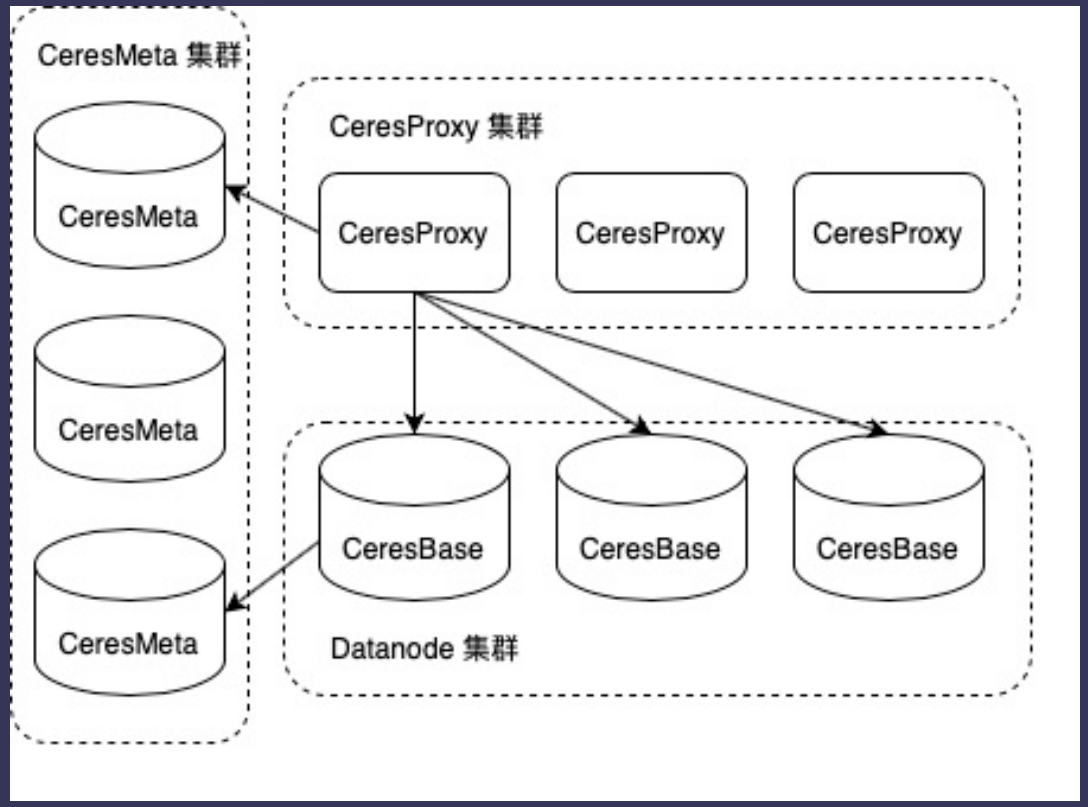
CeresDB 的技术栈

技术栈

CeresMeta(Java): 基于 SOFAJRaft 构建，提供进行集群管控功能。

CeresProxy(Golang): 无状态节点，负责鉴权、协议转换、读写路由等。

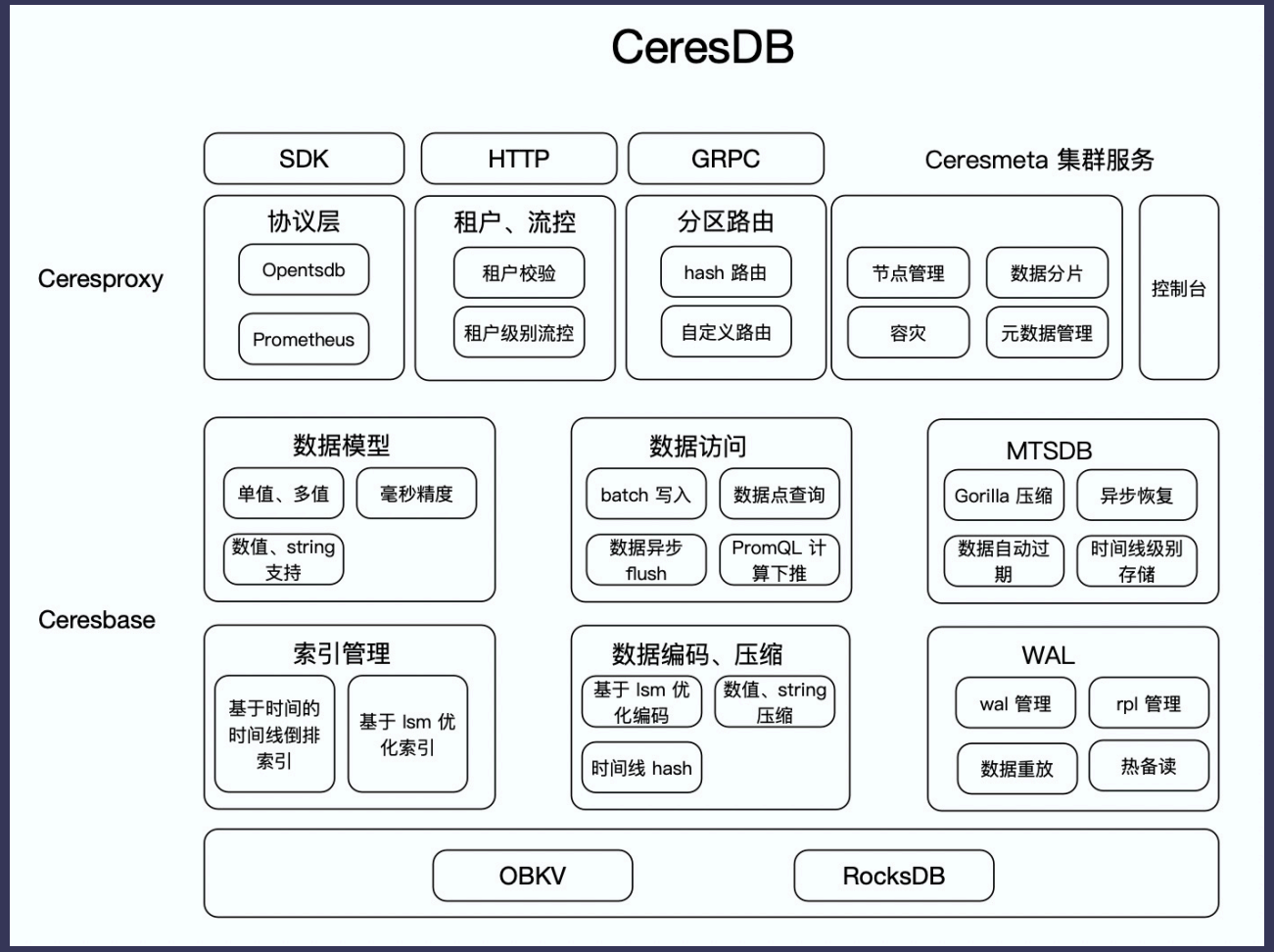
CeresBase(Rust)：存储+计算节点。



为什么使用 Rust?

- 团队背景
- 安全
- 高效

CeresDB 的技术栈



技术特点

- 性能&成本
  - 极致压缩
  - 纯内存时序数据库
  - PromQL 下推计算
- 高可用
  - 基于 OBKV：存储计算分离
- 稳定性
  - Chaos 环境不间断测试
  - 读写动态限流、时间线限制等

目录

经验分享

3.1 编码优化

3.2 查询优化

3.3 内存时序数据库

3.4 踩过的坑

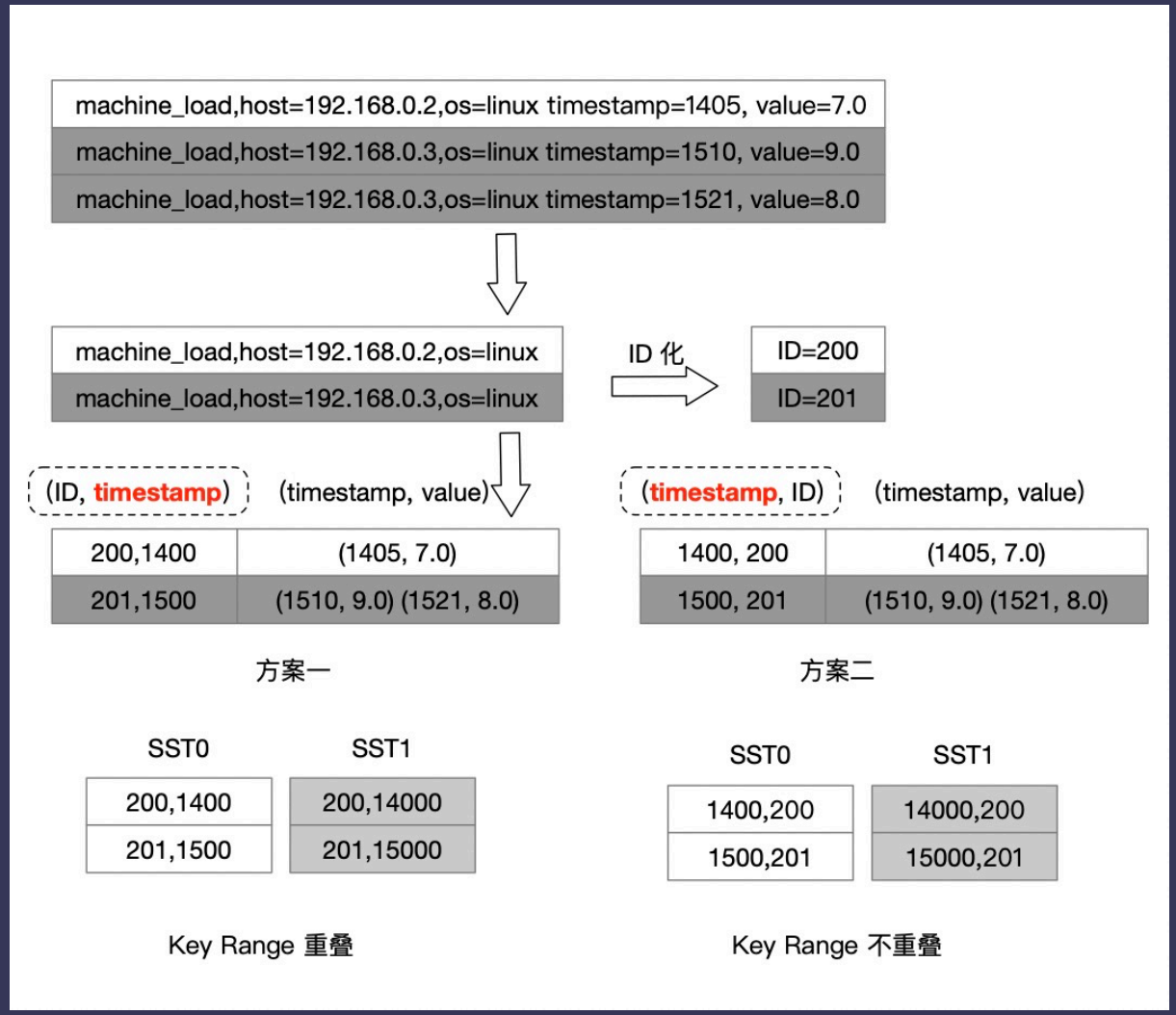
编码优化

背景

- 底层存储都是 LSM 类型的存储
- 写入的时序数据按照时间有序

问题

如何减小对于后续的 Compaction 引起写放大?



经验分享

3.1 编码优化

3.2 查询优化

3.3 内存时序数据库

3.4 踩过的坑

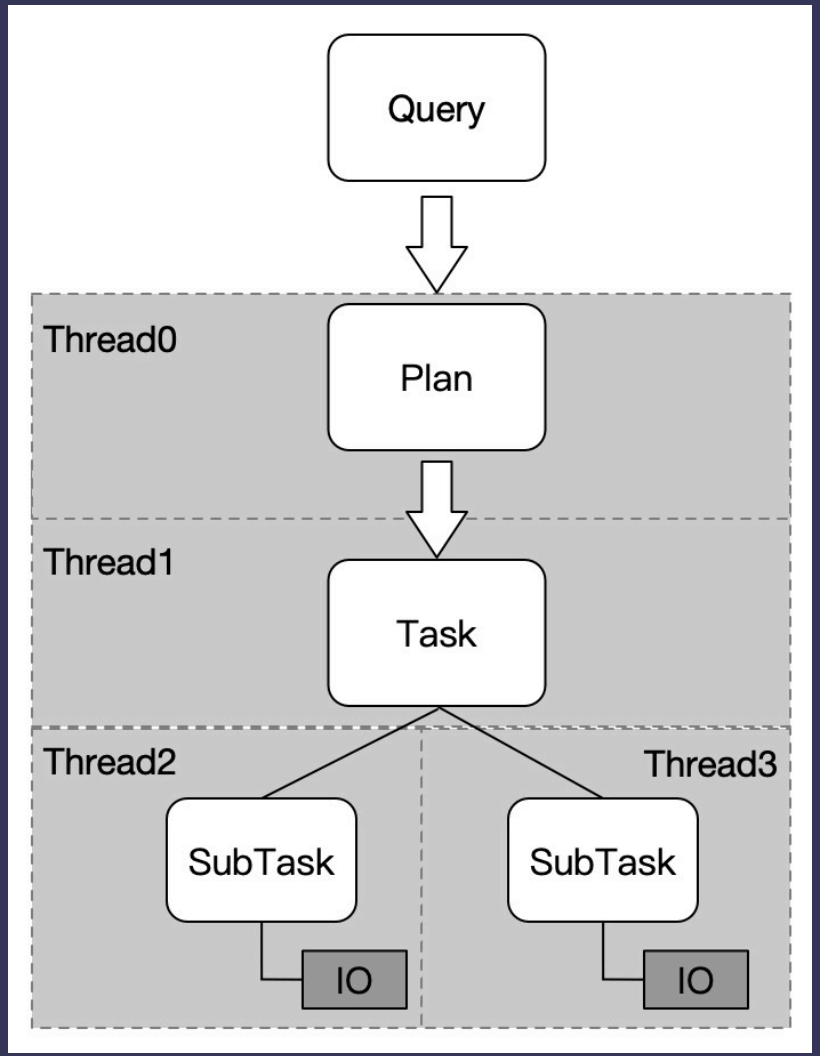
查询优化

背景

- 查询尽量分散到多核，降低延时
- 全量缓存查询涉及的数据
- 不同任务由不同的线程池负责

问题

- 任务拆分，导致 io 分散
- 全量缓存占用过多系统资源
- 上下文切换过多



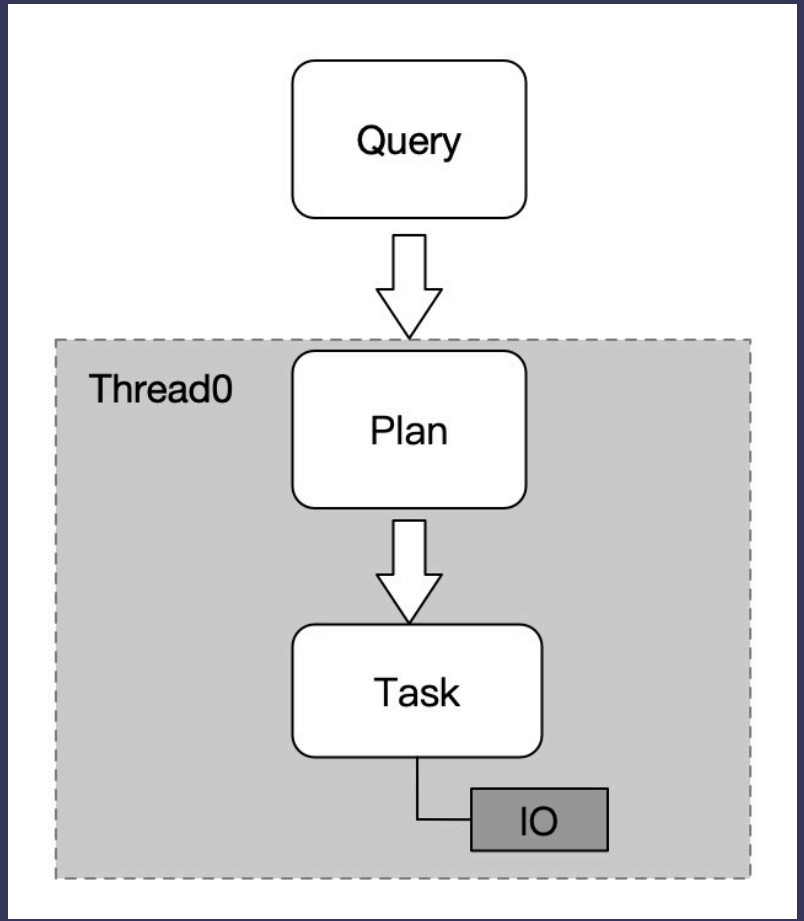
查询优化 cont.

方案

- 避免分散查询，合并查询
- 批处理的方式处理查询过程中的数据
- 所有任务的运行统一线程池

效果

- 线程调度减少，IO 合并，负载降低
- 同样的系统资源，支持更多的并发查询
- 查询延时降低，50～150ms => 10ms（p95）



经验分享

3.1 编码优化

3.2 查询优化

3.3 内存时序数据库

3.4 踩过的坑

内存时序数据库

背景

- 计算存储分离的架构下，定制缓存提升查询性能
- 查询具备时效性，关注最近的热数据，趋势数据
- 数据采样频率固定，但数据量大，需要压缩存储

问题

- 非通用 cache，需定制化——内存时序数据库
- 具备按照时间淘汰数据的策略
- 具备高压缩比

	存储方式	查询方式	压缩	数据过期	复杂度
Cache	数据点	点查	低	点维度	简单
MTSDB	**时间线**	**趋势查**	**高压缩比**	**时间维度**	**复杂**

内存时序数据库 cont.

极致压缩

时间戳: Delta of Delta

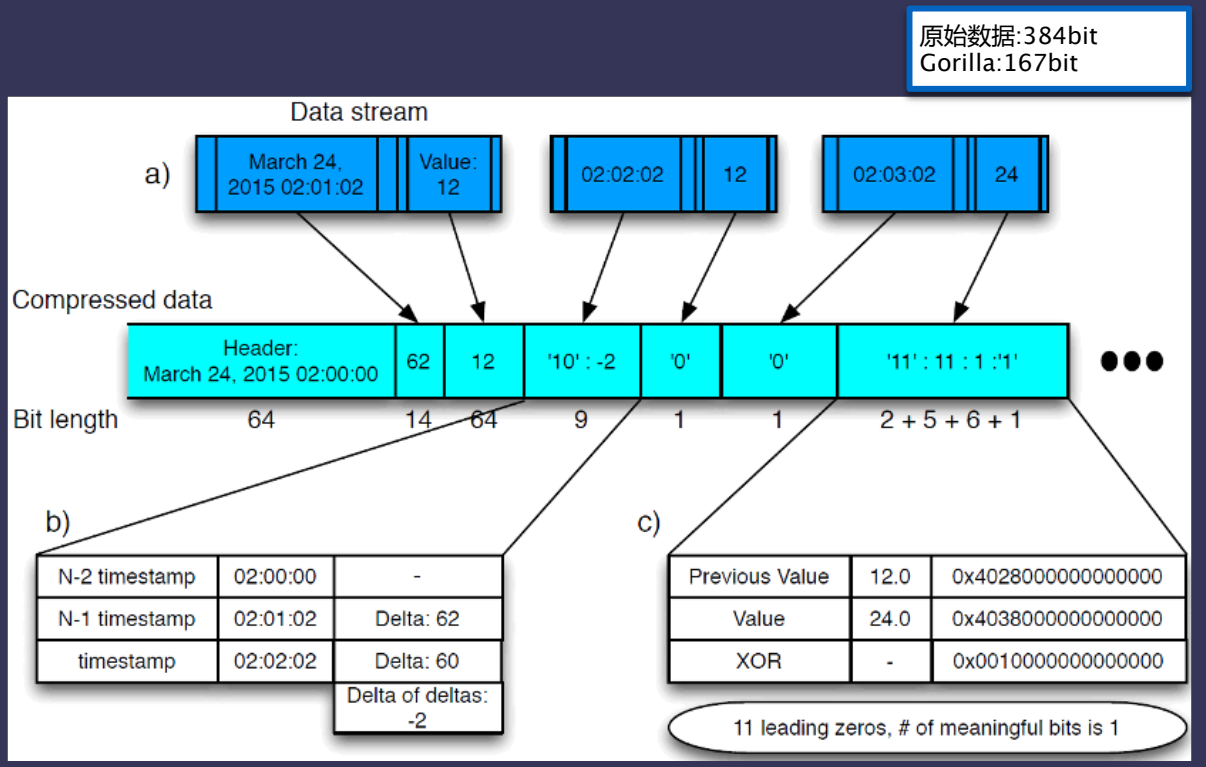
- 时间线上连续的数据点的 Timestamp 可以视作一个等差数列

浮点数: XOR

- 在大多数时间线中，相邻数据点的值变化小

增强

- 支持毫秒精度
- 支持更新

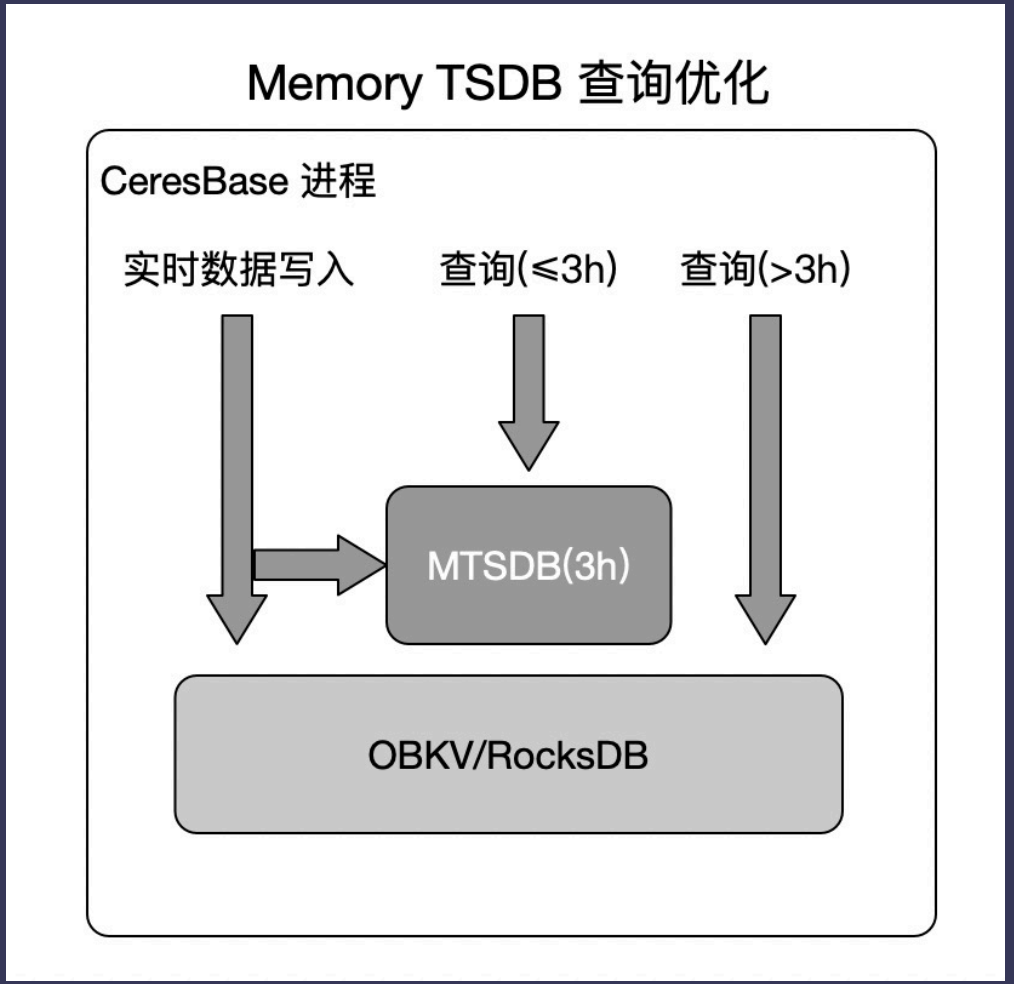


— 来自 Facebook gorilla 压缩算法论文

内存时序数据库 cont.

高效存储

- 按照时间线管理数据、压缩数据
- 按照时间管理数据过期
- 时间线+时间段，提供高效的趋势查询能力
- 新硬件 AEP 扩展缓存时长



经验分享

3.1 编码优化

3.2 查询优化

3.3 内存时序数据库

3.4 踩过的坑

踩过的坑

Lock Scope in Match

- match 紧跟的表达式中的临时变量作用域是整个 match block。
- 当涉及到获取 lock 时，需要注意 lock 并没有释放。
- 除了 match 以外，if let 也具备这个特性。

```rust
use std::sync::RwLock;

const MAX_COUNTER_VALUE: i32 = 10000;

fn main() {
    let bounded_counter = RwLock::new(0i32);
    match *bounded_counter.read().unwrap() {
        MAX_COUNTER_VALUE => (),
        _ => {
            let mut bounded_counter = bounded_counter.write().unwrap();
            if *bounded_counter < MAX_COUNTER_VALUE {
                *bounded_counter += 1;
            }
        }
    };
}
```

踩过的坑 cont.

Match Enum

- match enum 的时候，慎用 _ placeholder。
- 当为新增 enum 类别时，会引起函数语义的错误。

- 例子: is_readonly 的语义在新增 MultiGet 的 RpcRequest 之后会产生错误。
- 建议: 全匹配，不用 _ placeholder。

```rust
#[derive(Clone, Copy, Debug)]
enum RpcRequest {
    Put,
    Get,
}

fn is_readonly(req: RpcRequest) -> bool {
    match req {
        RpcRequest::Get => true,
        _ => false
    }
}
```

```rust
#[derive(Clone, Copy, Debug)]
enum RpcRequest {
    Put,
    Get,
    // new rpc request
    MultiGet,
}

fn is_readonly(req: RpcRequest) -> bool {
    match req {
        RpcRequest::Get => true,
        _ => false
    }
}
```

新增

踩过的坑 cont.

Option Size

- 试图使用 None 来节约内存使用
- 实际上 None 的大小是和 Some 没有区别

- 建议：Option<Box<T>> 的形式来节约内存
- Option<Box<T>> 触发 Option 优化，等于 Box 的大小

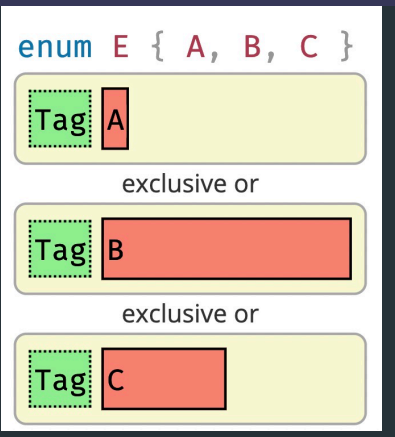
```rust
use std::mem;

#[derive(Clone, Copy)]
struct Data {
    x: i64,
    y: i32,
}

fn main() {
    let data = Data {
        x: 0,
        y: 0,
    };
    let some_data: Option<Data> = Some(data);
    let none_data: Option<Data> = None;
    let box_none_data: Option<Box<Data>> = None;

    println!("some_data size: {}", mem::size_of_val(&some_data));
    println!("none_data size: {}", mem::size_of_val(&none_data));
    println!("box_none_data size: {}", mem::size_of_val(&box_none_data));

    // some_data size: 24
    // none_data size: 24
    // box_none_data size: 8
}
```



—来自 cheats.rs

目录

1. 背景

2. 经验分享

3. **展望**

展望

- SIMD
- Runtime Codegen
- UDF
- 软硬件结合

我们长期招人，欢迎联系！

谢谢！

xikai.wxk@antgroup.com