字节跳动时序存储引擎的探索和实践

字节跳动基础架构研发工程师/陈骁

大纲

- 技术挑战
- 整体架构
- 热存Tsdc
- Khronos
- 未来展望

时序数据模型

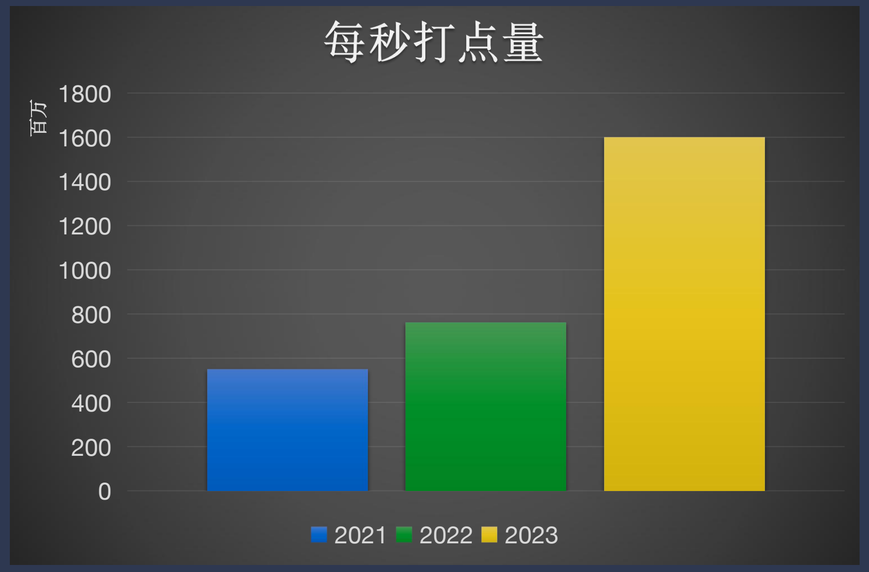
Field Values

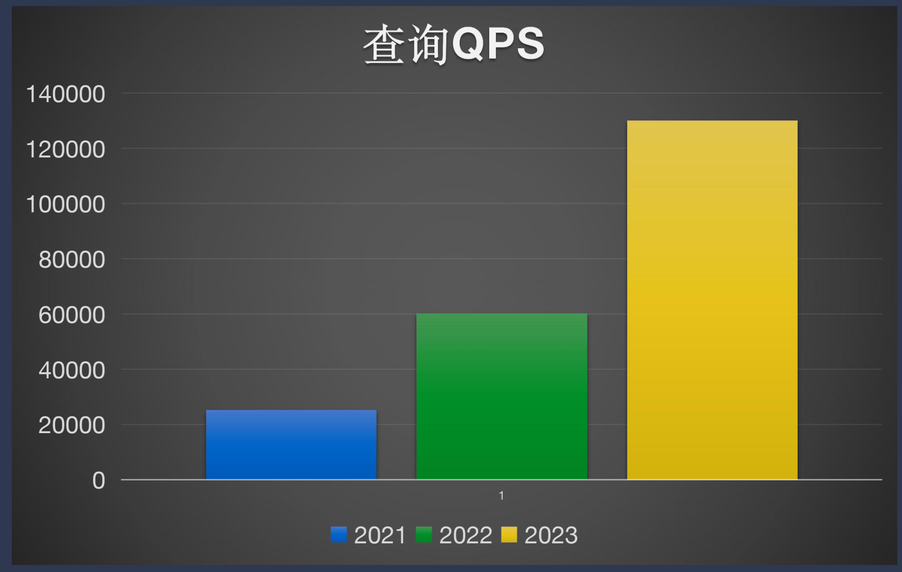
Metric	Tags	Time	CpuUsage	MemUsage
node.stat	idc=1,host=0.0.0.0	2023.04.17 10:30:00	15	30
node.stat	idc=1,host=0.0.0.1	2023.04.17 10:30:00	30	50
node.stat	idc=1,host=0.0.0.0	2023.04.17 10:30:10	31	49
node.stat	idc=1,host=0.0.0.1	2023.04.17 10:30:10	50	50

- 时间序列数据是按照时间序列变化的一组值，反映某个观测值随着时间的变化
- Metric Name + Tags标识一个唯一的时间序列（e.g. 一个观测对象）
- Field Values是具体的度量值，可以有一个或者多个

字节时序数据库使用现状

- 写入点数每秒10亿+
- 查询QPS 100k+
- 指标名数量1亿+
- 活跃时间线1000亿+





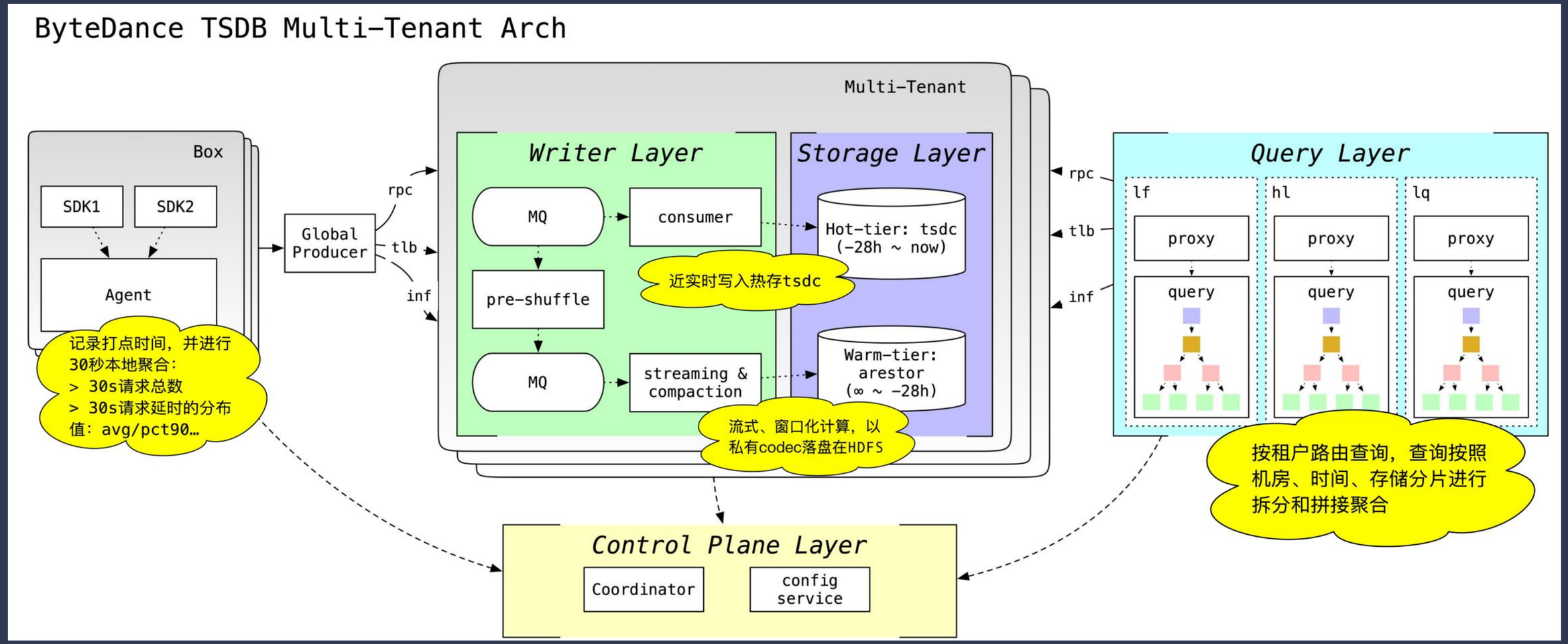
Workload分析和对应的挑战

- 写远大于读，写入量非常大
  - 线性扩展
- 查询以分析为主，点查为辅
  - 面向分析查询优化的同时兼顾点查性能
- 超高维度
  - 在单机亿级活跃维度情况下依然保证写入和查询性能
- Noisy Neighbours
  - 租户间的隔离
  - 防止个别超大metric影响整体可用性

大纲

- 技术挑战
- 整体架构
- 热存Tsdc
- Khronos
- 未来展望

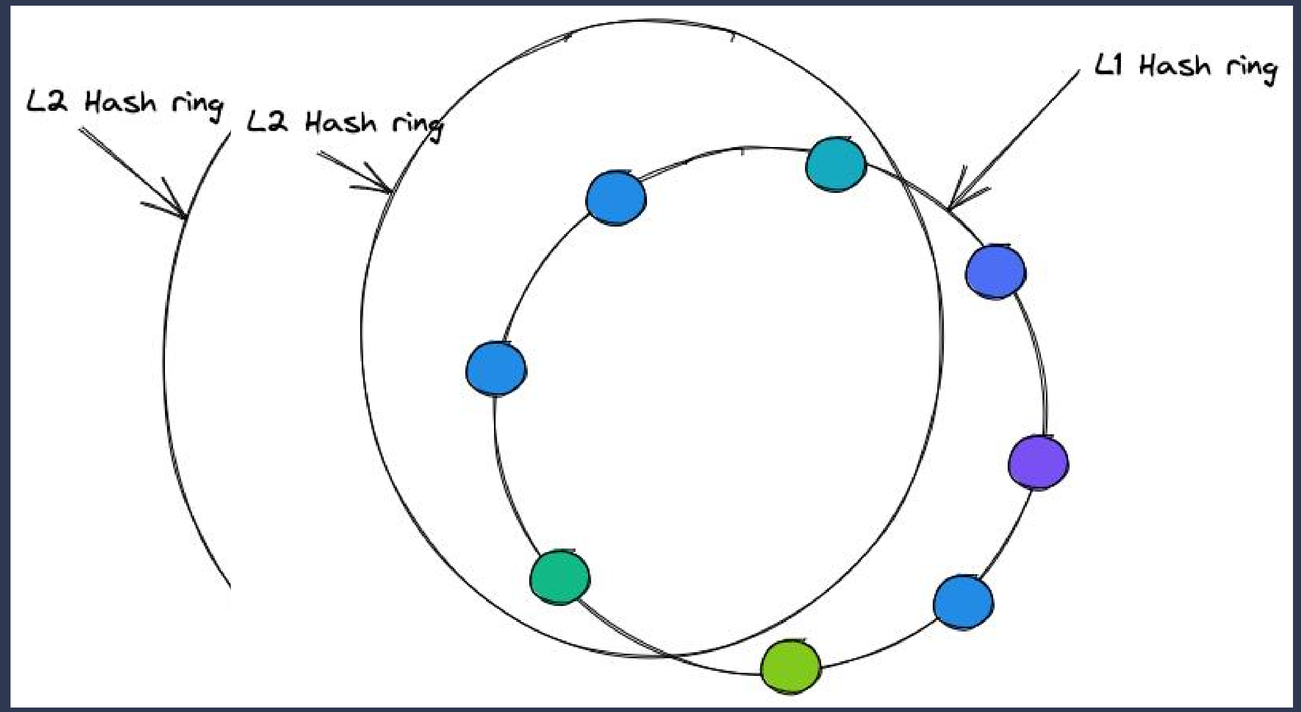
ByteTSD整体架构



如何线性扩展

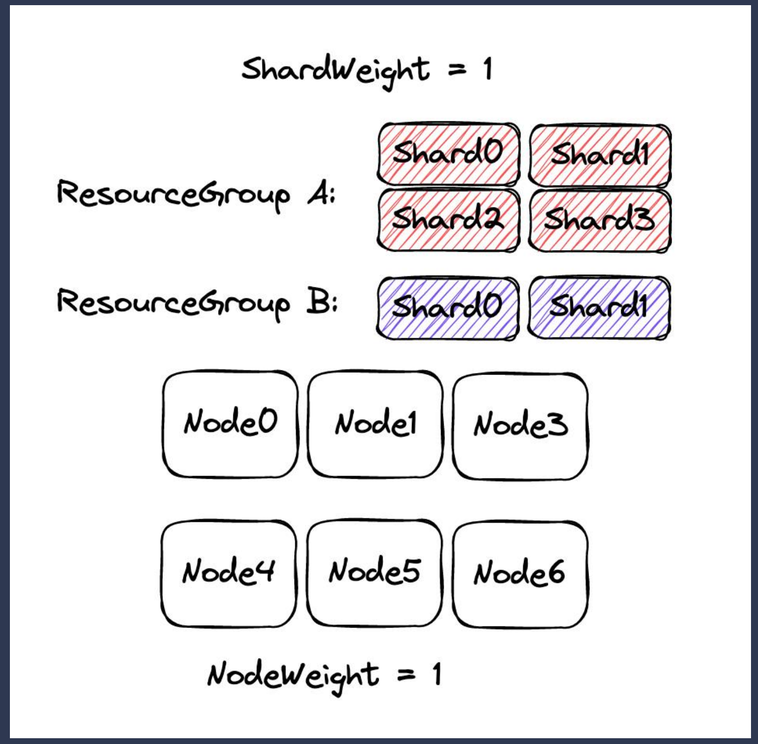
- 二级一致性Hash分区
  - 先按照Metric做一次Hash分区
  - 再按照序列做第二次Hash分区

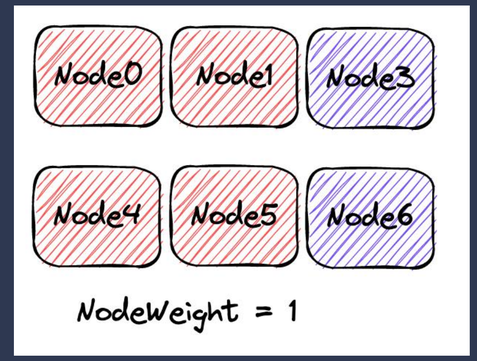
- Metrics级别的动态分区
  - 不同维度的Metric可以拥有不同的二级Hash分片数



如何保证隔离性

- ResourceGroup
  - 被调度的对象
  - ShardWeight表示需要多少资源
- Node
  - 资源的容器，调度的目的地
  - NodeWeight表示有多少资源
- 灵活设置Weight达成理想的资源分布

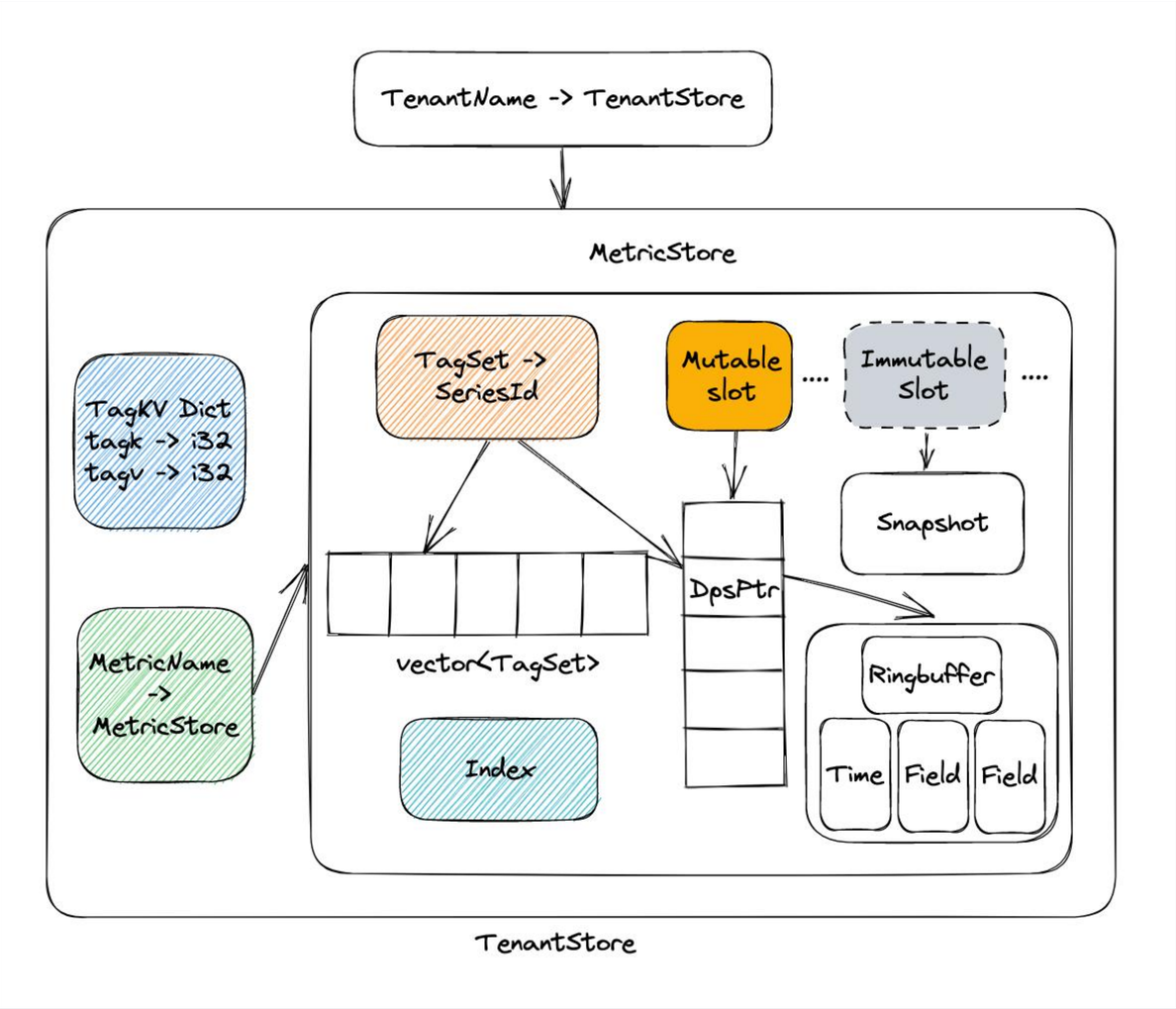




大纲

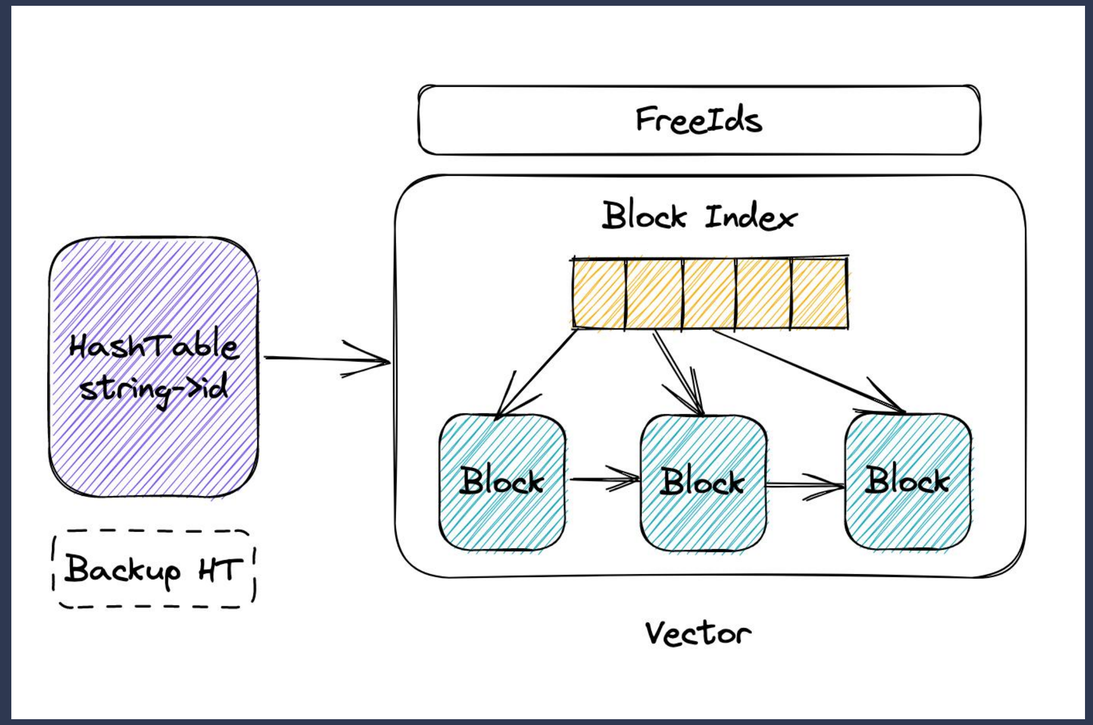
- 技术挑战
- 整体架构
- 热存Tsdc
- Khronos
- 未来展望

深入Tsdc



- 内存存储，提升热数据的读写性能
- 数据按时间分为多个Slot
  - 最近的slot可修改
  - 历史slot落盘释放内存
- 元数据只存一份
  - TagKV字典化
  - TagSet Varint编码后字典化
  - 按需建索引
  - 定时GC

字典结构



- Dictionary = HashTable + Vector
- Vector = BlockIndex + Block
  - O(1)的随机访问
  - 对BlockIndex做快照，实现无锁的遍历
  - 临界区很短，读写互不影响
- 异步Rehash
- 通过Epoch Based Reclamation机制回收内存，避免无锁遍历时访问无效的内存

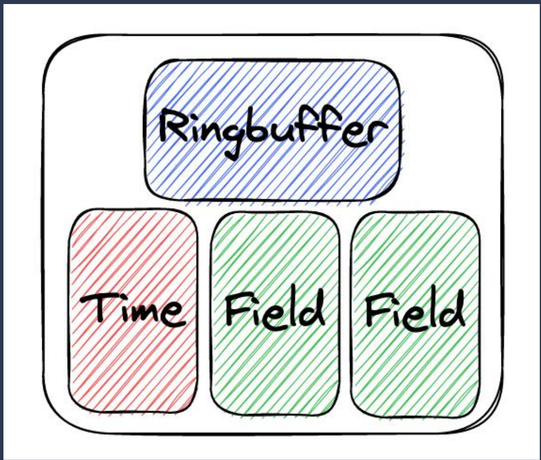
TagKeySet

Metric	Tags	Time	CpuUsage	MemUsage
node.stat	idc=1,host=0.0.0.0	2023.04.17 10:30:00	15	30
node.stat	idc=1,host=0.0.0.1	2023.04.17 10:30:00	30	50
node.stat	idc=1,host=0.0.0.0	2023.04.17 10:30:10	31	49
node.stat	idc=1,host=0.0.0.1	2023.04.17 10:30:10	50	50

TagKeySet: idc,host

- 观察数据特征
  - 大部分序列拥有相同的TagKeys
- 每个序列的所有TagKeys称为TagKeySet
- 直接编码整个TagKeySet
  - TagSet中只存储一个id
  - Encode时只做一次Hash

DatapointSet



Ringbuffer

Compressed Timestamp

- RingBuffer用于处理乱序写入，存储原始数据点
- 数据点划出RingBuffer后，写入TimeBuffer和ValueBuffer
- TimeBuffer使用delta of delta压缩
- ValueBuffer使用Gorilla压缩

乱序写入优化

Question：

- RingBuffer容量有限
- Gorilla压缩算法只能append

Answer：

- 反向Gorilla压缩，能够Popback
- 乱序很久的点不写入ValueBuffer，查询时合并

Ringbuffer

Compressed Timestamp

查询优化

- 支持所有Filter下推，减少数据传输量
  - 包括wildcard和regex，利用索引加速
- 自适应执行
  - 根据结果集大小动态选择查询索引或者Scan
- 并行Scan
- 轻重查询隔离
  - 轻重查询使用不同的线程池
  - 根据维度和查询时长预估查询代价

性能数据

- 实例规格24c 240G
- 平均活跃时间线1.2亿+，总时间线4亿+
- cpu使用率40%左右，内存使用率55%左右
- 平均写入量50w点每秒，单核吞吐8w/s
- 轻查询平均延时500us左右，p99 ms级
- 重查询平均延迟10ms左右，p99 百ms级

大纲

- 技术挑战
- 整体架构
- 热存Tsdc
- Khronos
- 未来展望

现有的问题

- 重启丢数据，运维负担大
- 内存开销大，成本高
- 不支持单实例内单个Tenant多Shard，无法做负载均衡

Khronos

- 冷热存消费两遍数据，成本高
- 三副本消费，数据容易发生不一致

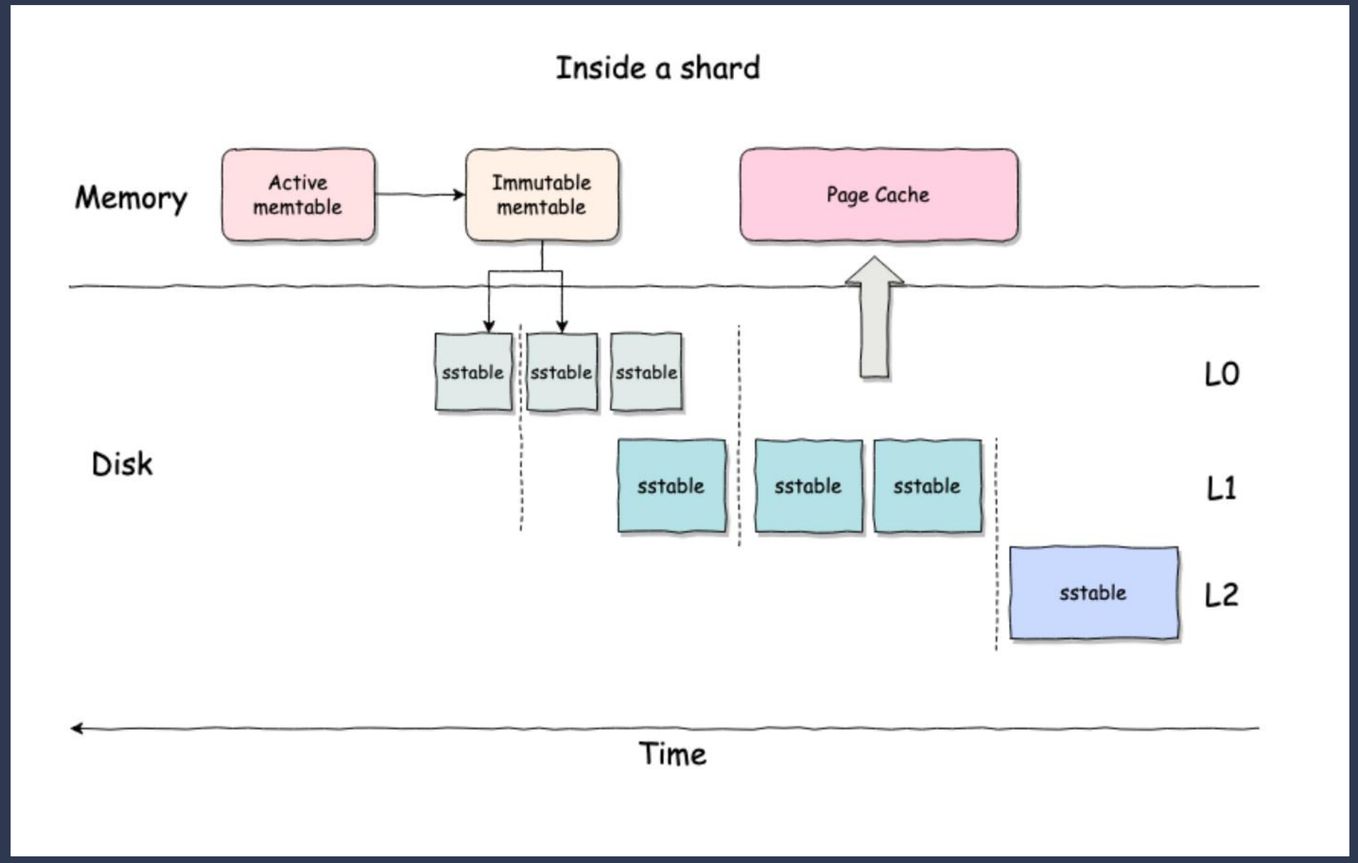
待解决

Khronos存储引擎

目标：

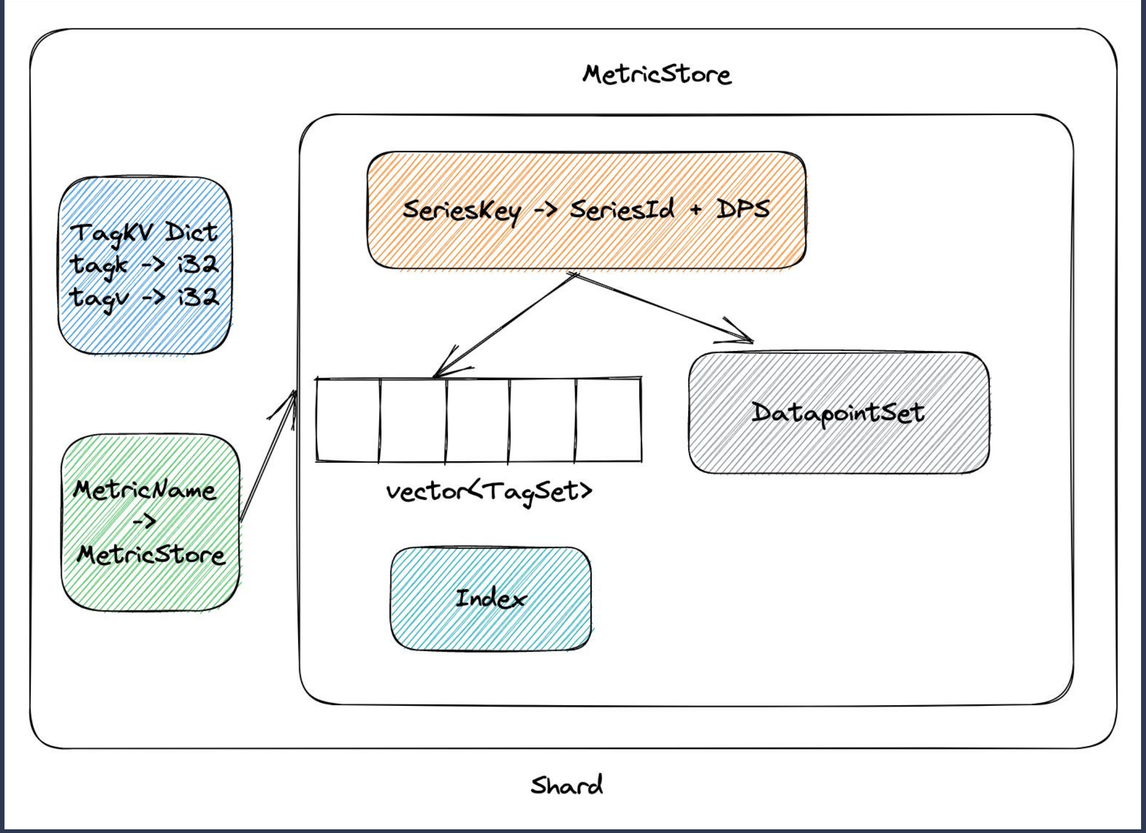
- 降低内存使用，更低成本地支持单实例高维度数据
- 数据全部持久化，提升数据可靠性
- 保持高写入吞吐、低查询时延，提供高效的扫描，同时支持较好的点查性能
- 能够以较低的成本支持较长时间的存储，提供较高的压缩率以及对机械盘友好的存储格式
- 兼容Tsdc，最低成本接入现有集群

Inside a shard



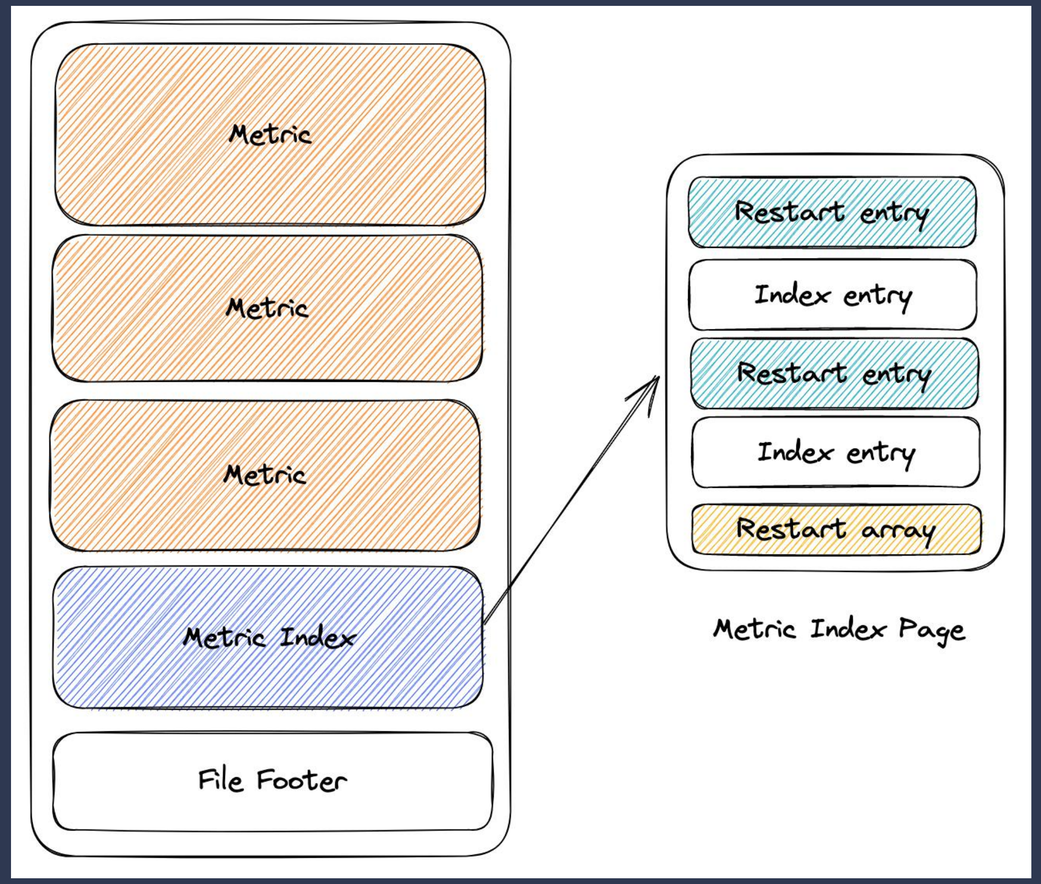
- 每个Shard内部都是一棵独立的LSMT
- 一共分为三层
- 每一层都有一个虚拟的时间分区
  - sstable文件不会跨时间分区
  - Compaction在分区内调度
  - 乱序写入的场景减少写放大

Memtable



- 基本延用了Tsdc的内存结构
- SeriesMap采用有序结构，Compaction依赖Series有序
- SeriesKey = SeriesHashCode + TagSet
  - 节省比较开销
  - 快速拆分range，方便做分区内并行查询

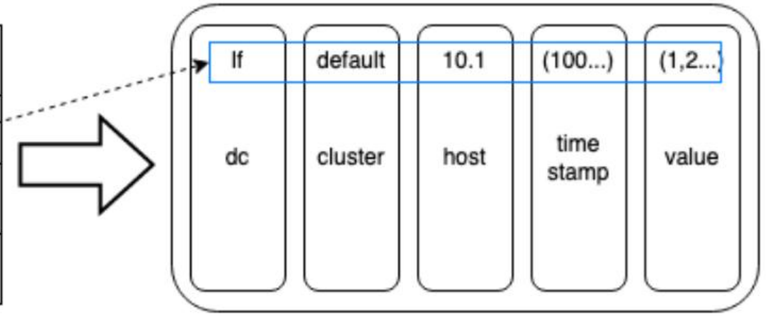
SSTable格式

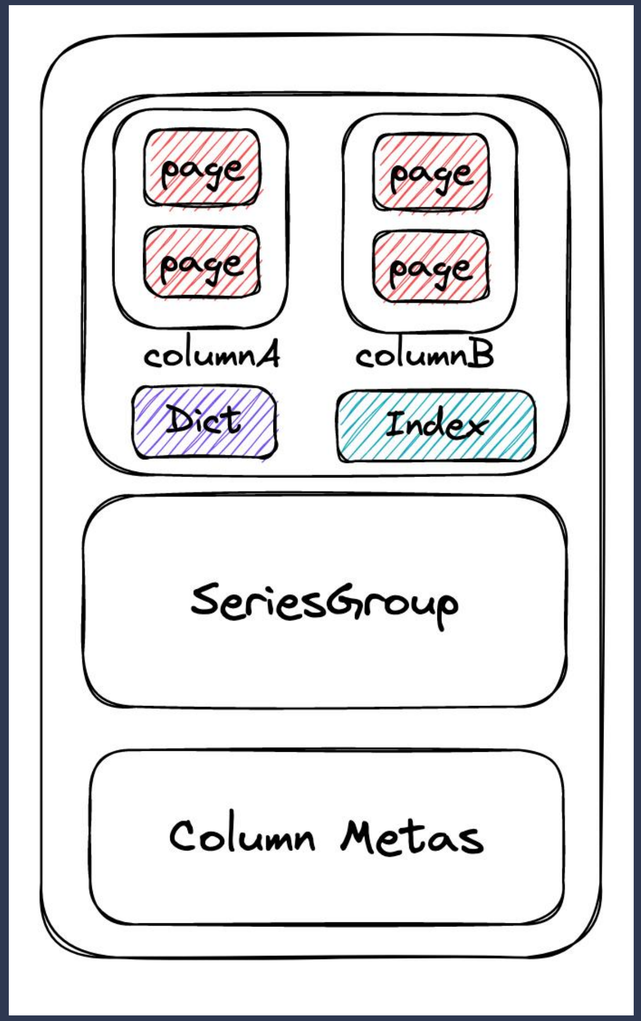


- 由于Metric数量非常多，所以将多个Metric数据混合存储在一个文件中
- 文件尾部有Metric Index指向Metric的位置
  - MetricIndex是一个Btree
  - Page内部使用前缀压缩

Metric格式

dc	cluster	host	timestamp	value
lf	default	10.1	(100,101,102)	(1,2,3)
lf	default	10.2	(100,102)	(4,7)
lf	test	10.3	(100,101,102)	(4,8,9)





- 类Parquet格式，行列混存
- 每行一个序列
- 大Metric会划分为多个SeriesGroup，减少内存占用
- 字典/Raw/Bitshuffle encoding
- Page索引加速查询

Flush优化

- 大量小Metric
  - 存储格式Overhead大
  - write次数太多，性能差
- BufferWrite
  - 预先Fallocate一段空间然后mmap
  - 数据通过mmap写入，减少syscall
- PaxLayout
  - 所有Column写在一个Page
  - 减少IO次数，减少元数据开销

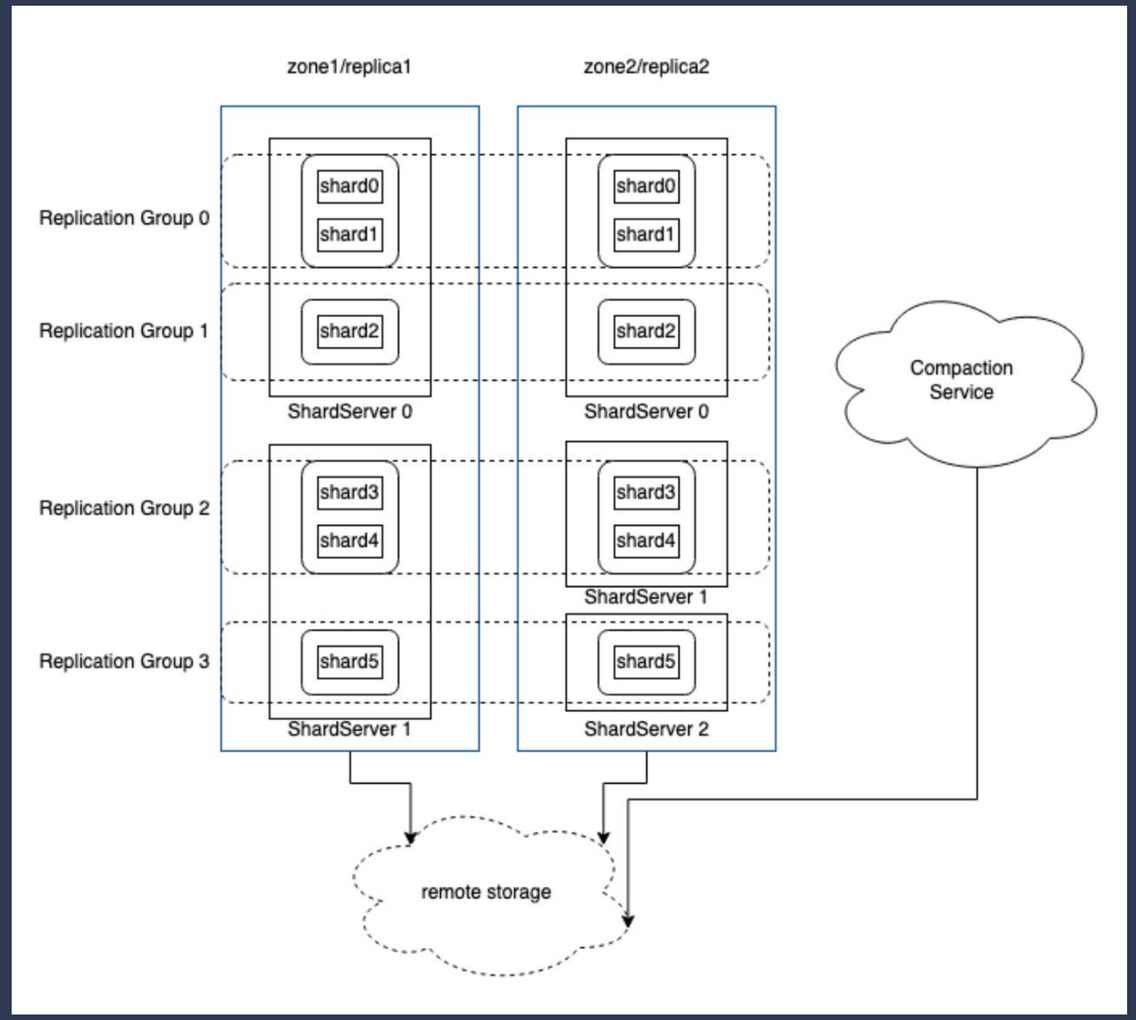
SSTable查询优化

- 延迟投影
  - 先读取带过滤条件的列
  - 每过滤一个列都缩小下一个列的读取范围
  - 最后投影非过滤列
  - 数量级性能提升
- PageCache
  - Cache中缓存解压后的Page，避免重复的解压和CRC校验
  - 更进一步，直接Cache PageReader对象，节省构造开销

大纲

- 技术挑战
- 整体架构
- 热存Tsdc
- Khronos
- 未来展望

存算分离



- 基于分布式存储
  - 提供大容量的存储，融合冷热存
  - 多副本间做复制
  - 分布式Compaction
  - 快速负载均衡

更多功能和优化

- 兼容社区
  - String
  - Bool
  - 元数据查询
- 持续性能优化
  - 更高效的数据传输协议
  - 利用字典编码加速查询
  - 算子下推
  - ……

加入我们

欢迎交流

想一想，我该如何把这些技术应用在工作实践中？

THANKS