DTCC

数，造，未，来

第十二届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2021

2021 年 10 月 18 日 - 20 日 | 北京国际会议中心

IT168.com ChinaUnix ITPUB

爱奇艺高性能KeyValue数据库研发与应用

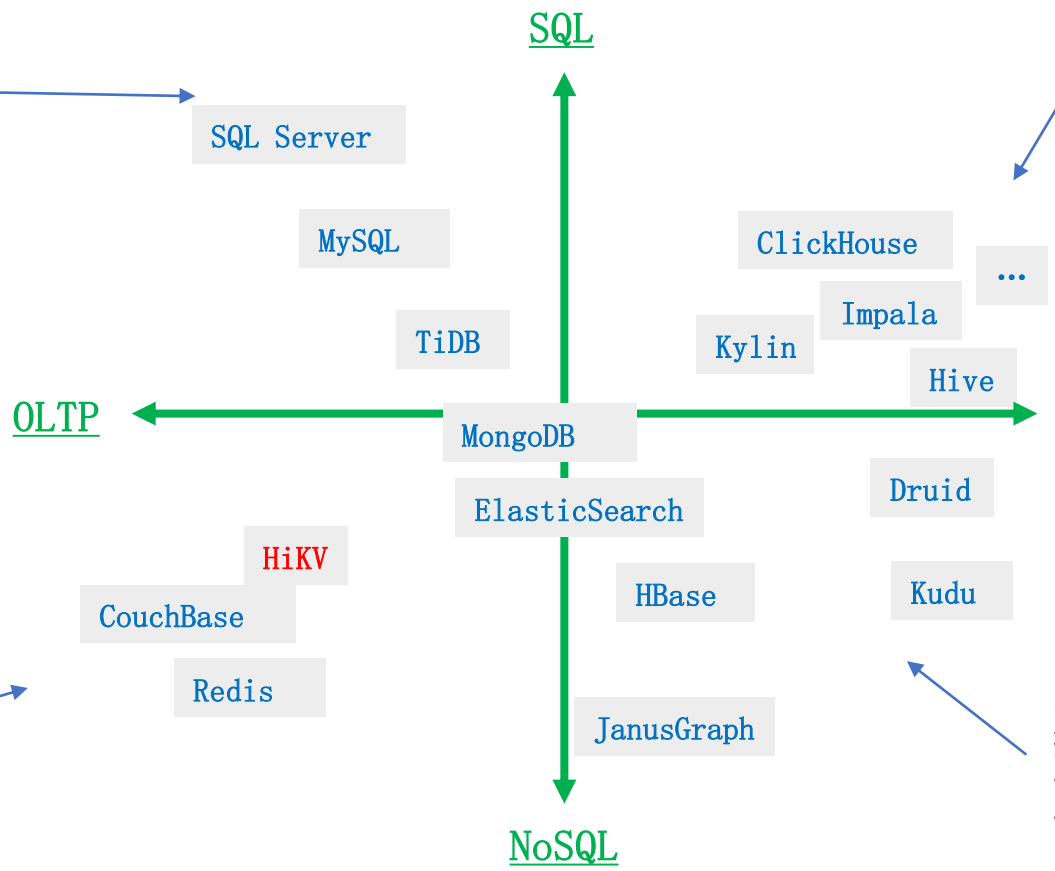
郭磊涛@iQIYI
https://www.iqiyi.com/

一系列“问号?”

- 爱奇艺用到了哪些数据库？
- 为什么要研发KeyValue数据库？
- 有哪些关键技术？
- 应用效果如何？
- 未来如何规划？

爱奇艺数据库服务

支持SQL和事务
高性能
高可靠
较难扩展
用于交易系统关键数据存储等



一般支持(类)SQL，不支持事务
扩展性好（PB+）
响应时间较长（秒~分钟级）
用于大数据分析

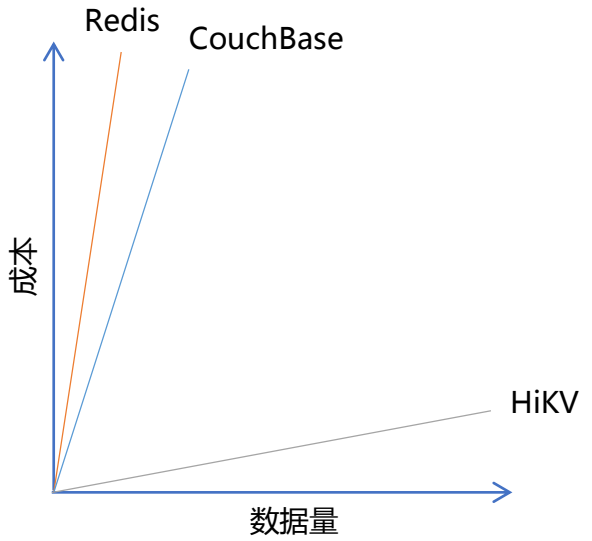
简单KV为主
高性能
基于内存/SSD，成本高
一般用做Cache

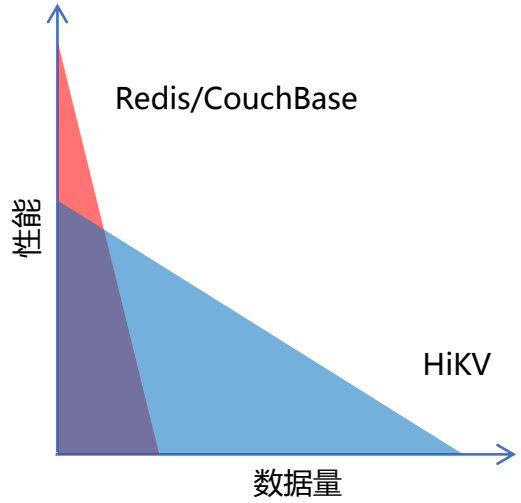
多列或灵活的schema定义
扩展性好（TB~PB）
性能较好（响应时间毫秒~秒级）
一般用于较大规模数据存储&查询

HiKV是什么？

- 爱奇艺基于Scylla二次开发的Key-Value数据库
  - Scylla : C++重写的Cassandra，高吞吐，低延时
  - HiKV : High Performance Key-Value Database

- 特点：**大容量+高性能**
  - 大容量：单集群百TB数据量
  - 高负载：单节点10w+ QPS
  - 低延时：读写延迟 P99<10ms
  - 部署：多数据中心、在线扩容
  - 最终一致性



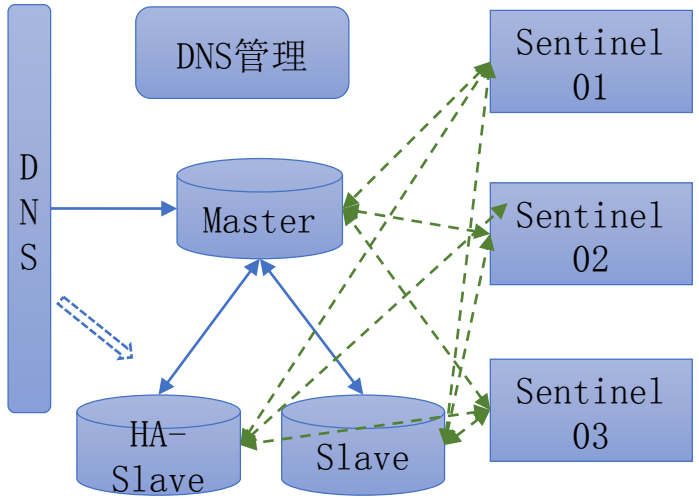


提纲

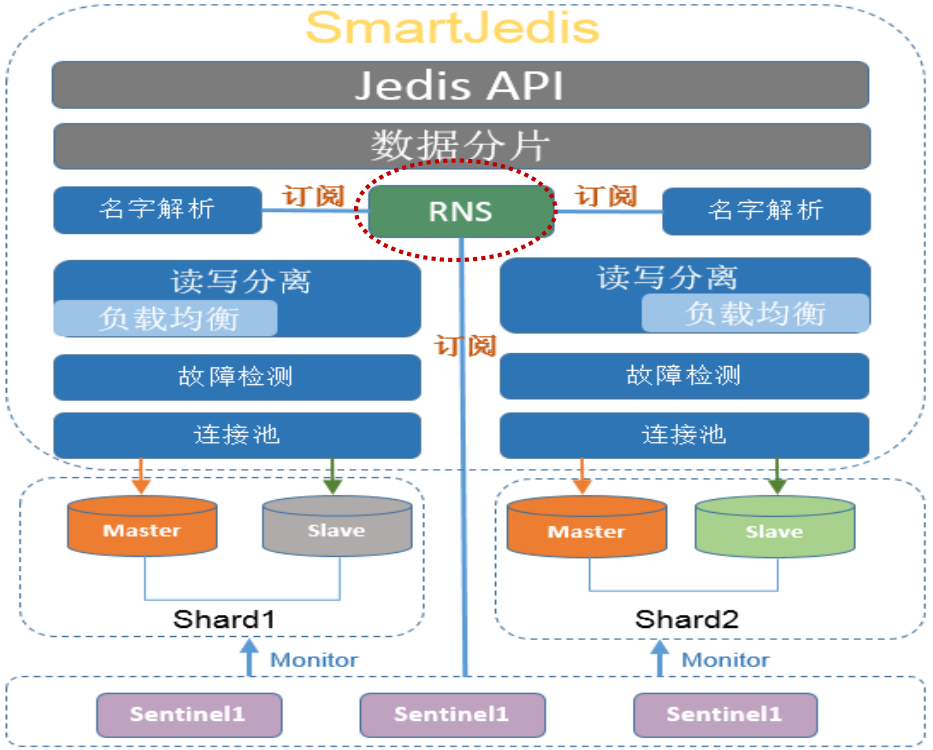
- **HiKV 研发动机**
- HiKV 关键设计思路
- HiKV 在爱奇艺的应用
- HiKV 的未来规划

Redis 适合大容量/低延时应用场景吗？

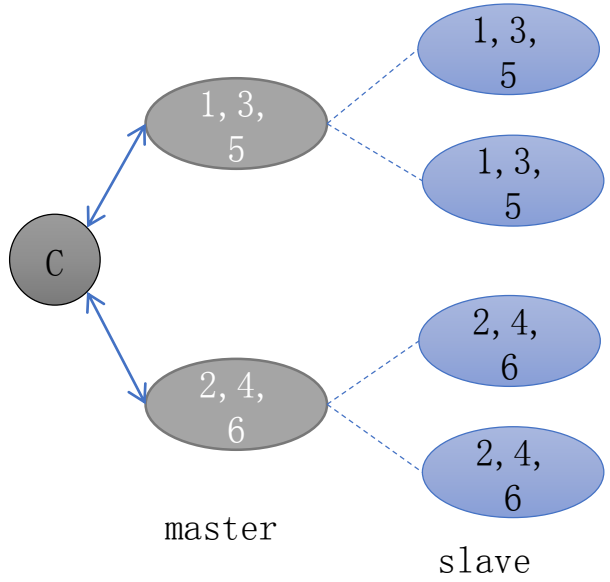
Master-Slave集群



Client-side sharding



Cluster

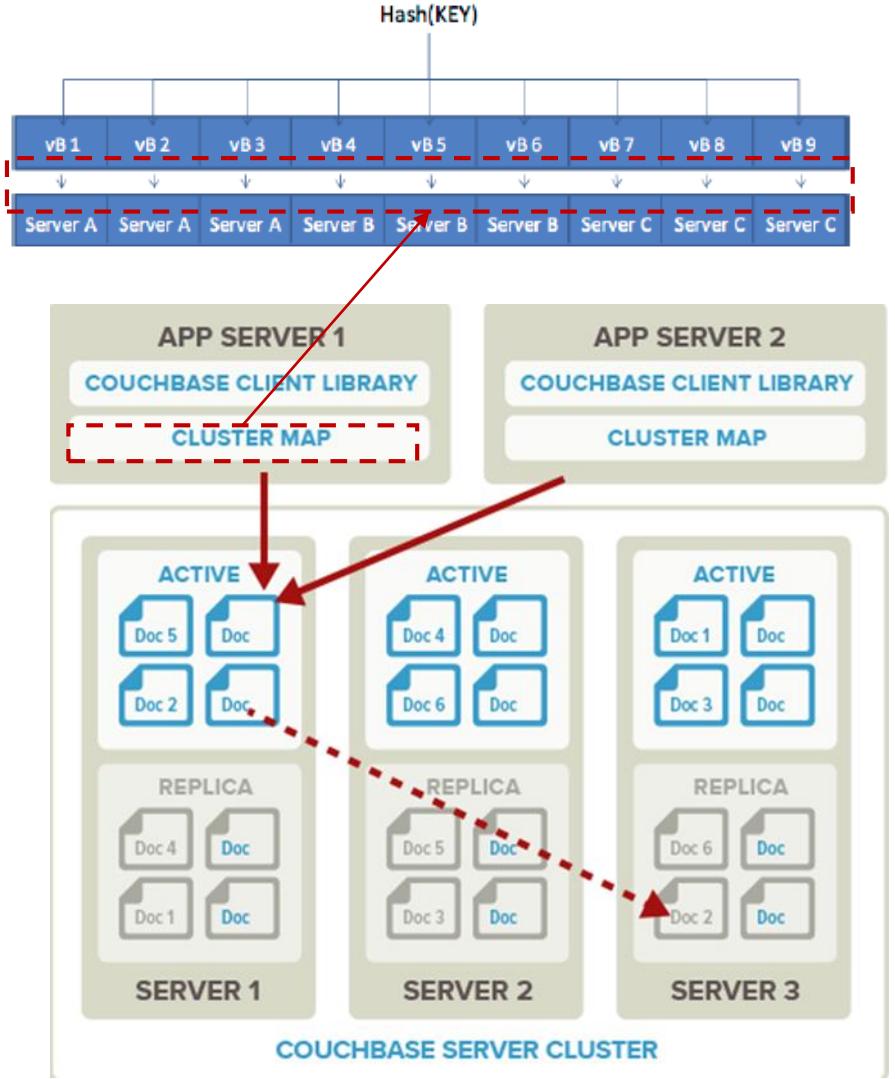


Redis扩展性差，无法存储百TB级数据量

CouchBases适合吗？

分布式高性能NoSQL数据库

- 3种bucket（等价于Database）
  - Memcached：KV, 不持久化，无副本
  - Ephemeral：无持久化，有副本（v6.0）
  - **Couchbase**：JSON，持久化，有副本，Rebalance
  - 1 Bucket 包含 1024 vbucket
- 支持N1QL（类SQL）
- 容量：扩容方便，目前线上**TB级**
- 性能：与key/value size相关
- XDCR：支持跨数据中心集群间同步



Couchbase易扩展，“数据量<可用内存”时性能极高。但是，存储成本过高！

大容量开源NoSQL系统

Couchbase

选择Scylla、mongoDB、Couchbase进行性能对比

大容量开源NoSQL系统 – 测试场景

服务器

服务端：3台物理机
CPU: 2 x Intel Xeon Gold 5118 @ 2.30GHz, 24 Cores 48 Threads in total
Memory: **192GB**
SSD: 8 x INTEL SSDSC2KB96 (960GB)

客户端：3台物理机
CPU: 2 x Intel Xeon Gold 6148 @ 2.40GHz, 40 Cores 80 Threads in total
Memory: 512GB

数据集

小数据集（Small）
1亿条 X 1KB长度 X 3副本
数据量<内存

大数据集（Large）
5亿条 X 1KB长度 X 3副本
数据量>内存

负 载

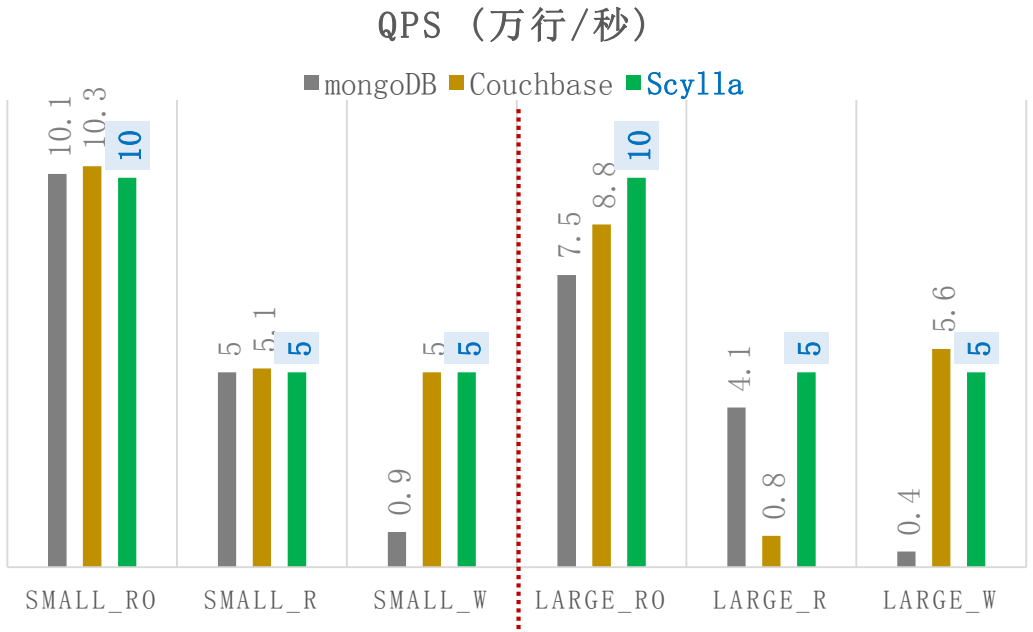
只读（RO）
加载数据集后，10w Read + 0 Write

随机读写（RW）
加载数据集后，5w Read + 5w Write

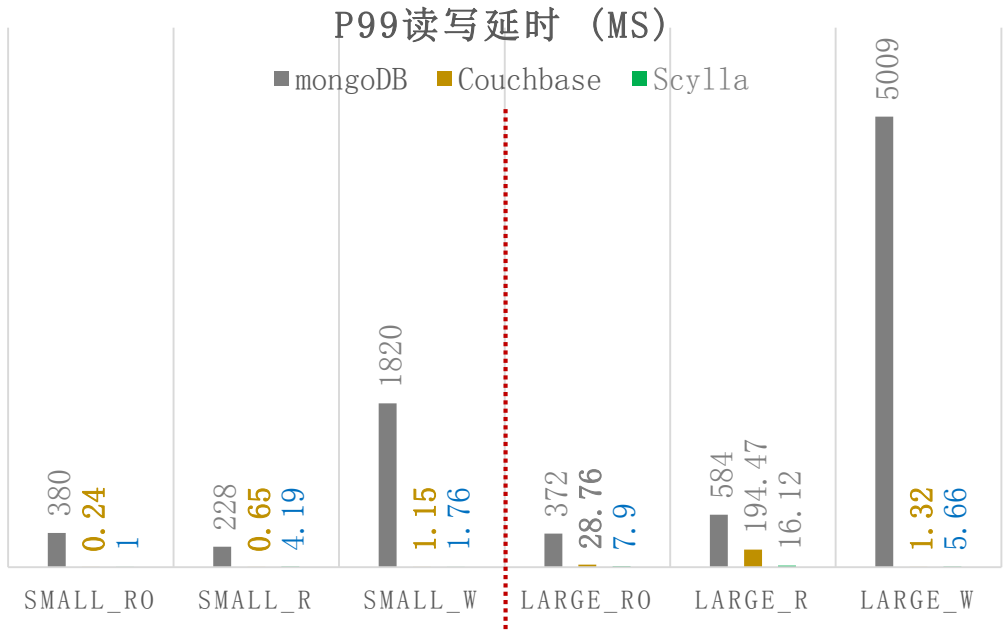
版 本

MongoDB 4.2.0
Couchbase 6.0.0
Scylla 2.0.4

大容量开源NoSQL系统 – 测试结果



- 仅有 **Scylla** 满足所有场景QPS要求
- **mongoDB** 在cache dirty比例达到20%后，写请求开始排队（qw），**写**QPS急剧下降
- **Couchbase** 在大数据量读写场景下，数据异步持久化造成**读**QPS急剧下降

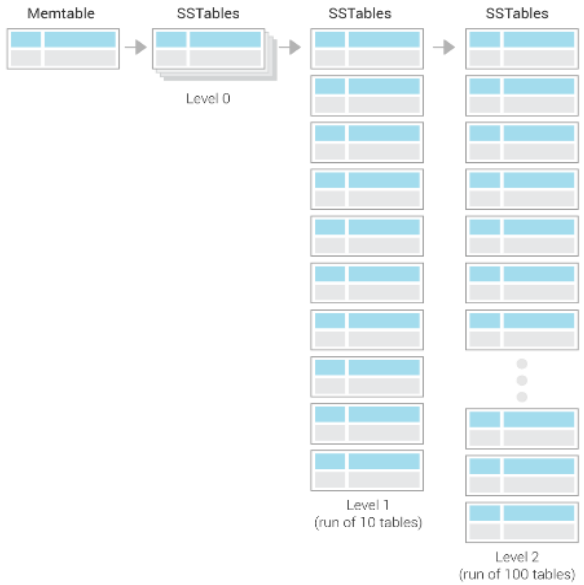


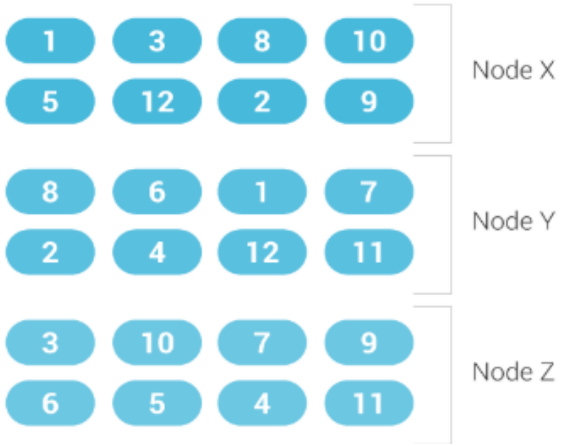
- 均不满足读写延时需求（P99<10ms）
- **Couchbase** 在小数据量，数据均缓存在内存，延时最优。大数据量读写时，读延时较大
- **Scylla**大数据量读写时，读延时最优。压测2小时后，开始出现10s+超时

Scylla最接近我们的需求，但高负载下长尾延时明显。是什么原因？

Scylla 工作原理

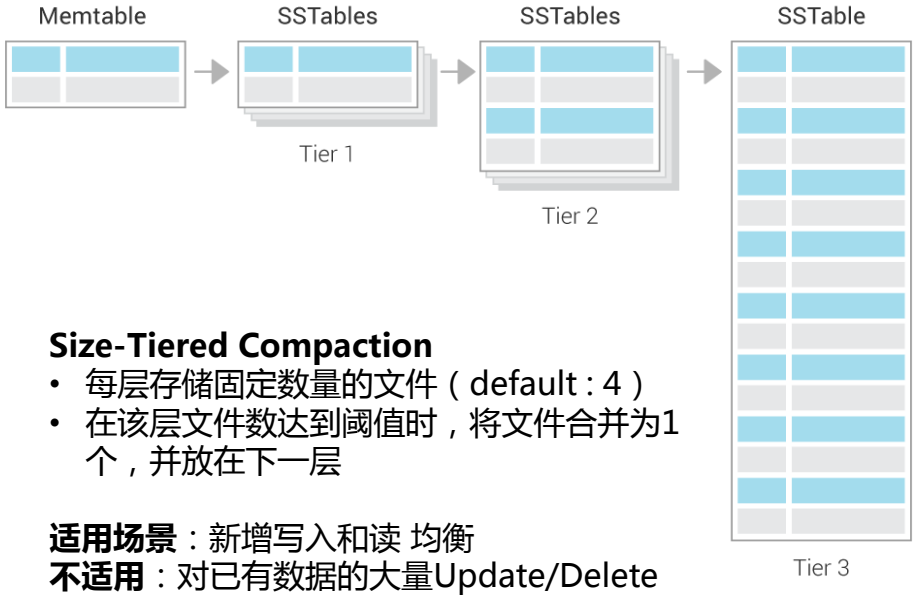
```
CREATE TABLE loads ( machine, cpu, mtime, load ,
PRIMARY KEY ((machine, cpu), mtime) )
WITH CLUSTERING ORDER BY (mtime DESC);
```





Coordinator → **Murmurhash3** (PartitionKey) → **Token** (64bit) → **Token Range** → **VNode** → **Physical Node** → **Commit Log** / **Mem Table** → **SSTable** → **Compaction**

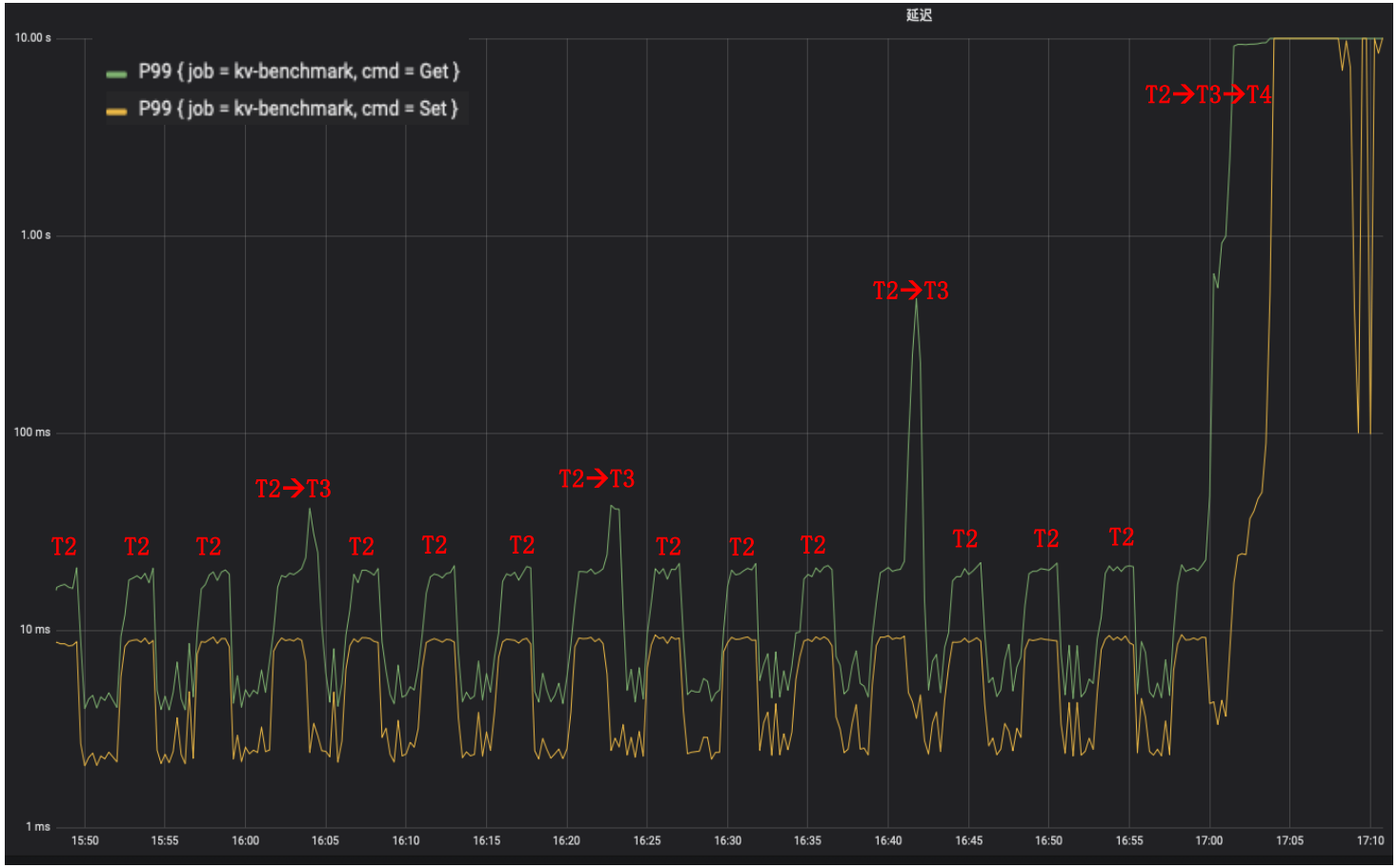
Scylla 延时抖动的原因



Size-Tiered Compaction
- 每层存储固定数量的文件（default : 4）
- 在该层文件数达到阈值时，将文件合并为1个，并放在下一层

适用场景：新增写入和读 均衡
不适用：对已有数据的大量Update/Delete

N：flush到盘上的文件数
写放大：O(logN)
读放大：O(logN)
空间放大：2x



基于LSM (Log Structured Merge) Tree的存储方式，会造成“写/读/空间的放大”

优化存储引擎，以减小“写/读/空间放大”

带来的性能抖动和空间浪费？

提纲

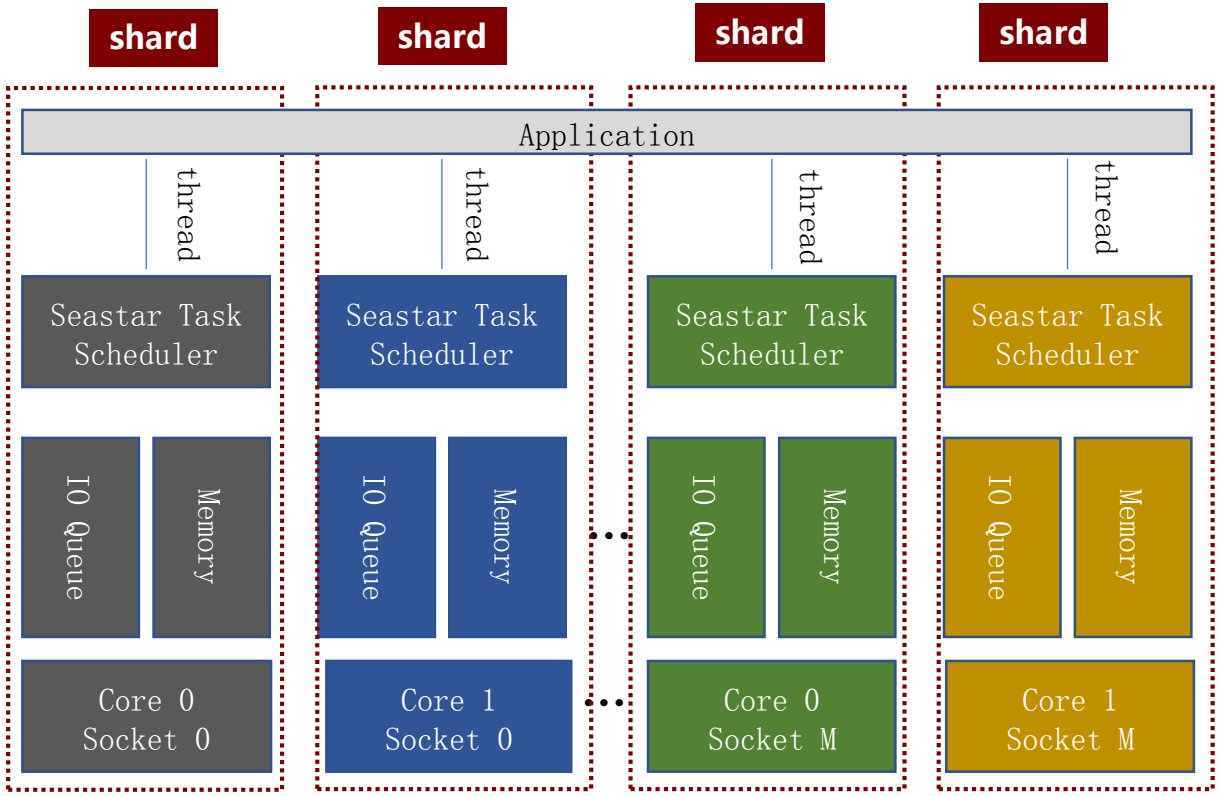
- HiKV 研发动机
- **HiKV 关键设计思路**
- HiKV 在爱奇艺的应用
- HiKV 的未来规划

HiKV 设计思路

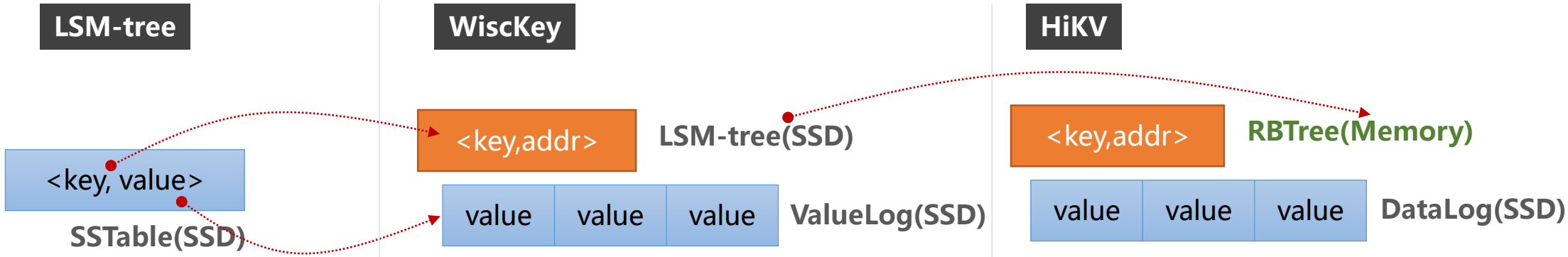
HiKV = Scylla + WiscKey存储引擎

- 继承**Scylla**的优势，发挥硬件的性能
  - Seastar Shared-nothing架构
  - CPU和I/O调度，用户态 I/O 队列
  - NUMA friendly
  - 查询缓存
  - 兼容Cassandra，一致性哈希分片
  - 无单点+多副本+多数据中心+多活

- 参考**WiscKey**的设计，减小写/读放大，降低长尾延时



HiKV 设计思路 – 数据存储方式



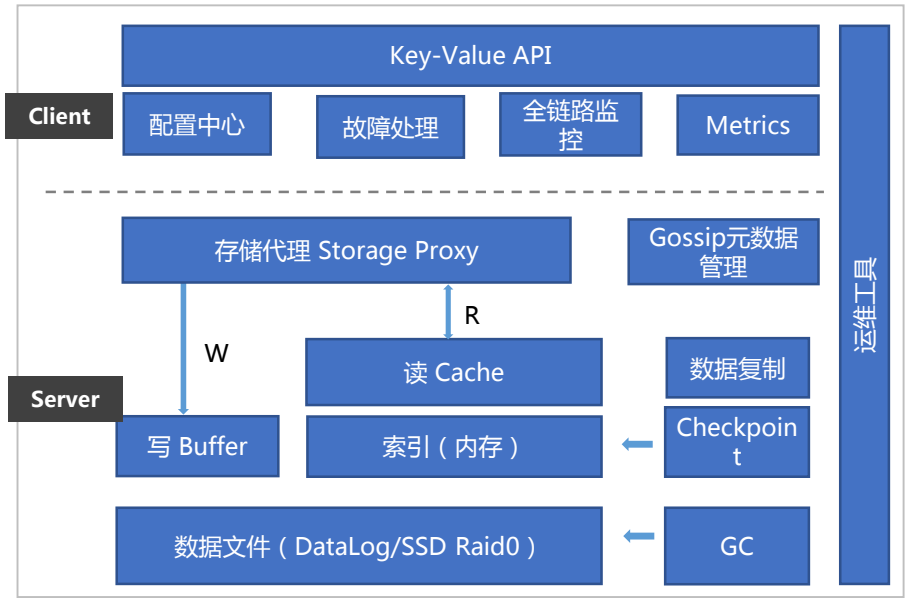
- 优化写：批量顺序写盘
- 优化读：Compaction（排序+GC）
- 缺点
  - Key与Value混存，检索效率低
  - 读/写放大
  - 浪费存储

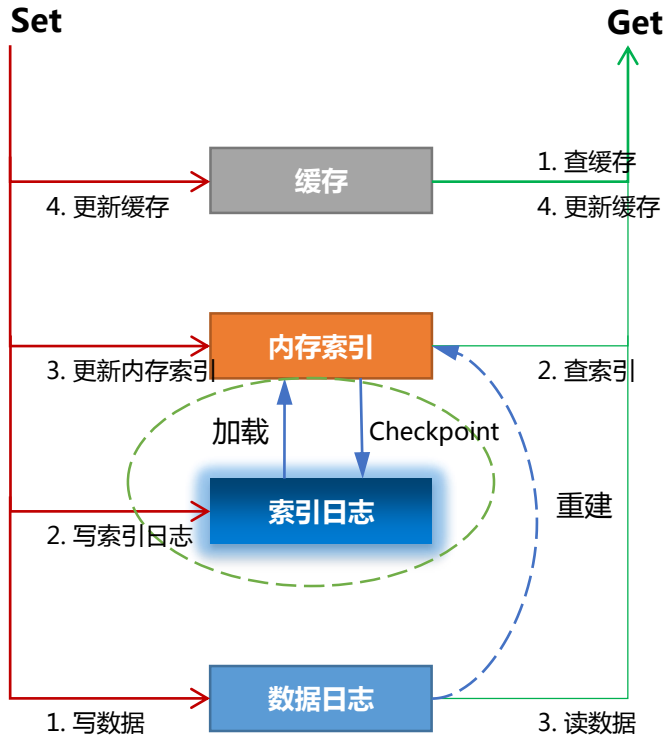
- 减小“读写放大”：索引（Key）与数据（Value）分开管理
- 优化写：Value批量顺序写盘
- 优化读：Key排序+缓存，支持Range
- 空间优化：ValueLog 轻量级GC
- 缺点
  - 索引（Key）读放大

- 消除索引（Key）的读放大
  - LSM-tree → RBTree/Hashtable
  - 读：1次盘IO
- 提高索引检索速度：SSD → Memory
- 索引持久化：索引日志 Checkpoint
- 空间优化：DataLog轻量级GC+手动全量清理
- 缺点
  - 不支持range查询

HiKV 整体架构

- 数据模型
  - <Key, Value>
  - Value 为单/多列，支持类型、TTL
- 支持的操作
  - Get/MGet
  - Set/MSet
  - Scan （ token range ）
- 分布式复制
  - Coordinator负责读写
  - 任何一个节点都可以是Coordinator
  - Coordinator通过Gossip共享元数据
- 高可用
  - 多副本（ Hinted Handoff/Read Repair ）
  - 多数据中心
- 可配的一致性级别(Consistency Level)
  - QUORUM/LOCAL_QUORUM
  - ONE/LOCAL_ONE
  - ALL/...





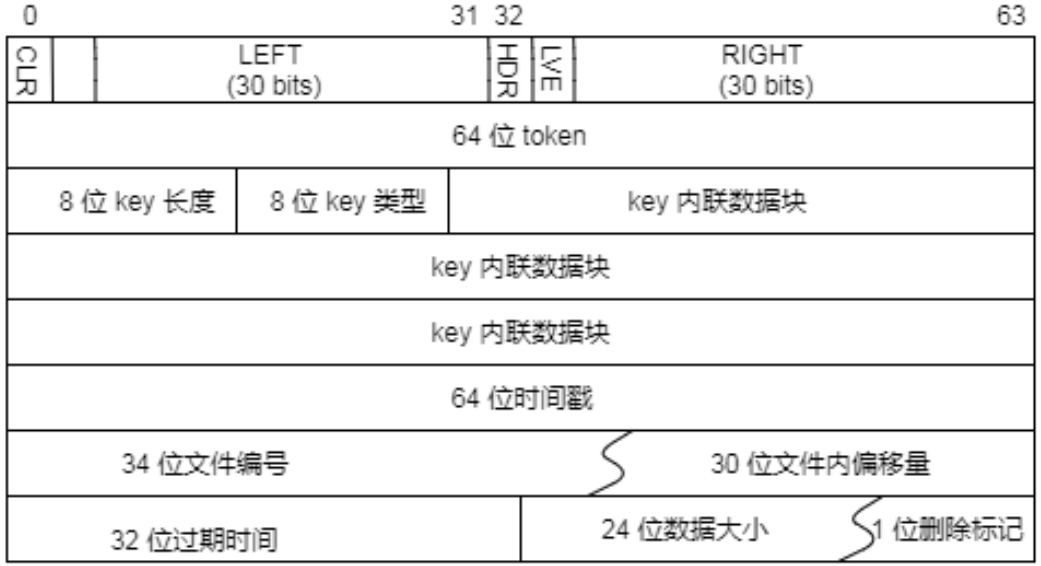
HiKV 单节点内读写路径

如何设计索引内存结构，以便在内存中可以记录所有数据的索引？

HiKV 索引内存结构

参考Aerospike的设计

- **每条数据 ⇔ 1条索引信息**
  - SET → 索引指向最新的Value
  - Delete/TTL → 删除索引
- 索引在内存中组成 **RBTree**
- 索引记录长度：**64 Byte**



- 物理机内存：512GB
- 索引数据内存占比：90%
- 单条数据记录长度：1KB

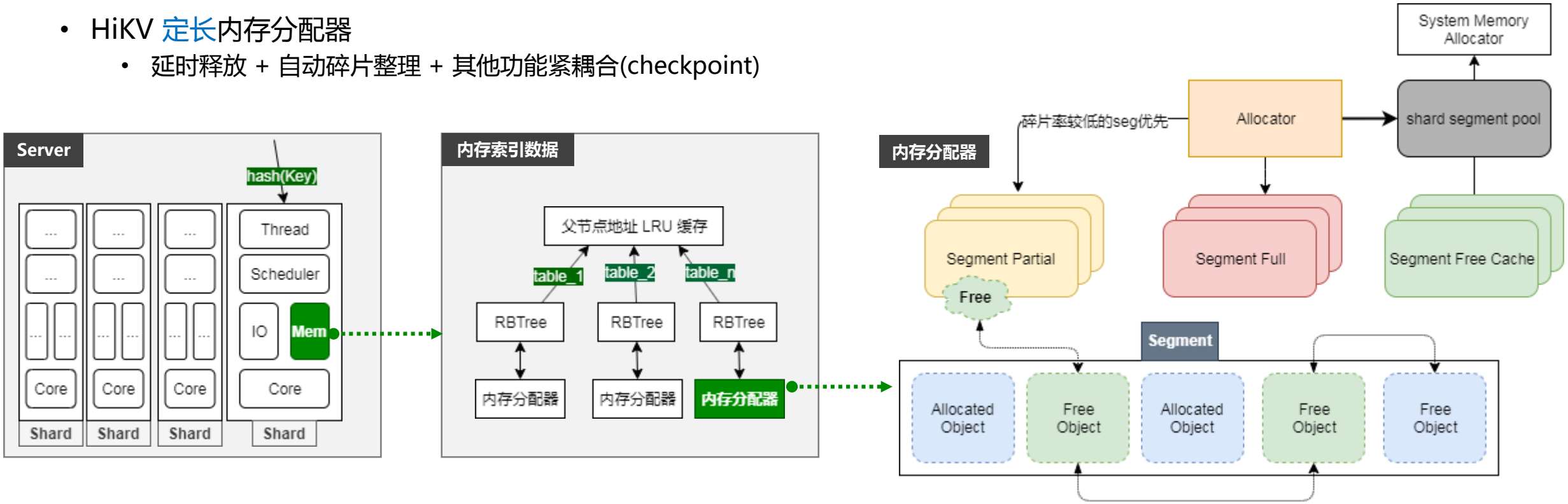
单台物理机
存储容量

- HiKV 2.0：**7.2** TB

内存如何管理？ 使用Seastar的内存分配器LSA（Log-Structured Memory Allocator）？

HiKV 内存管理

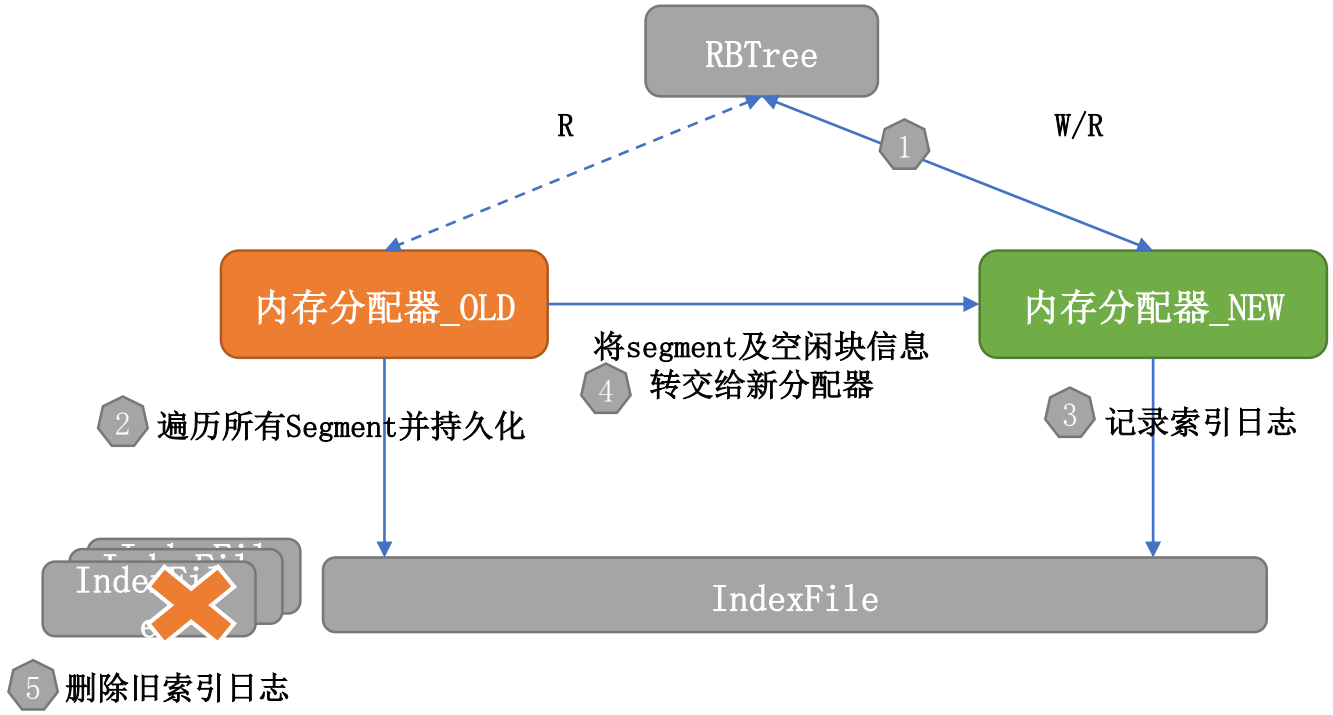
- Seastar LSA
  - 延迟释放 + 自动碎片整理 + 变长 ➔ **索引记录长度需要96Byte，单机存储量下降33%**
- HiKV 定长内存分配器
  - 延时释放 + 自动碎片整理 + 其他功能紧耦合(checkpoint)



索引数据如何进行 持久化 及 快速加载？

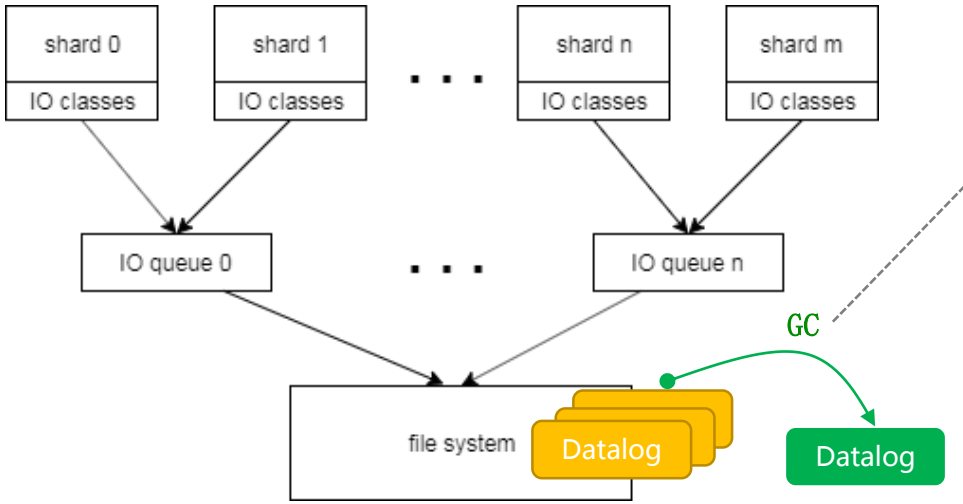
HiKV 索引数据Checkpoint

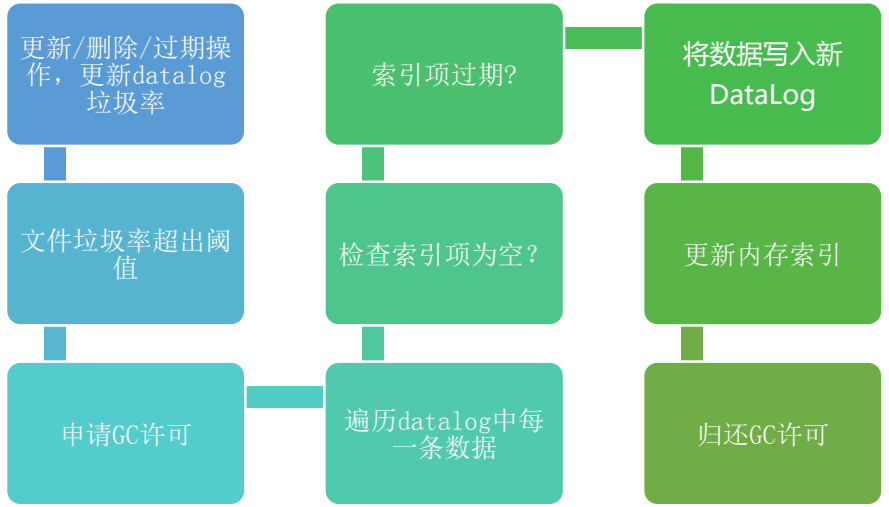
- 将某时刻内存索引数据持久化，并删除之前的IndexLog，加快节点启动速度



HiKV 数据文件GC

- Datalog中的垃圾数据
  - 删除的数据
  - 被覆盖的旧数据
  - 过期的数据
- Datalog GC
  - 将垃圾率超出阈值的Datalog重写，释放存储空间
  - 控制GC Thread并发度和优先级，避免影响正常操作

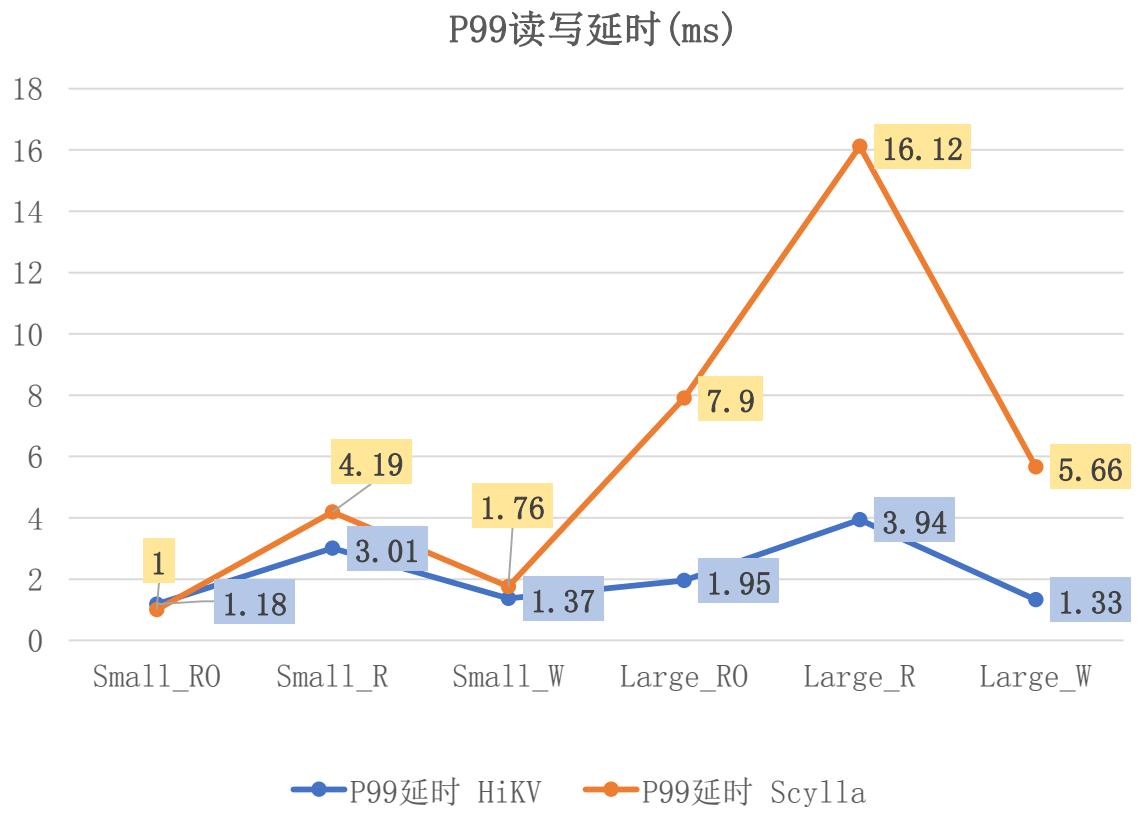


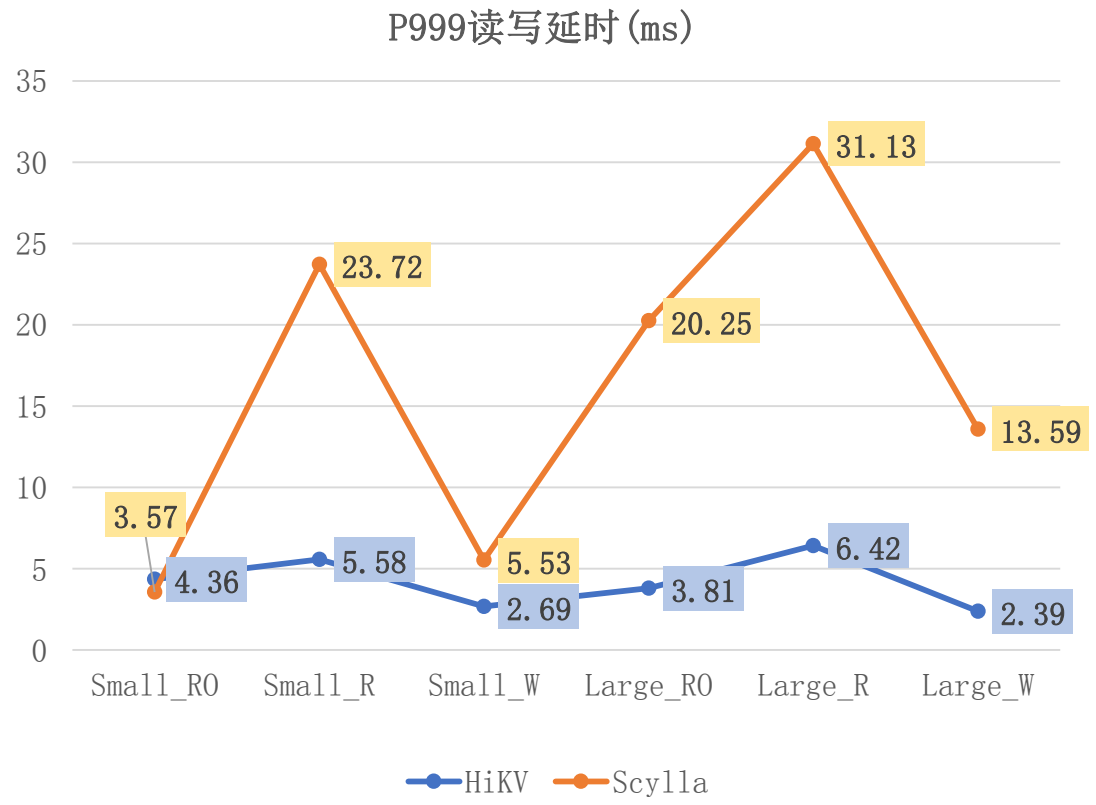


提纲

- HiKV 研发动机
- HiKV 关键设计思路
- **HiKV 在爱奇艺的应用**
- HiKV 的未来规划

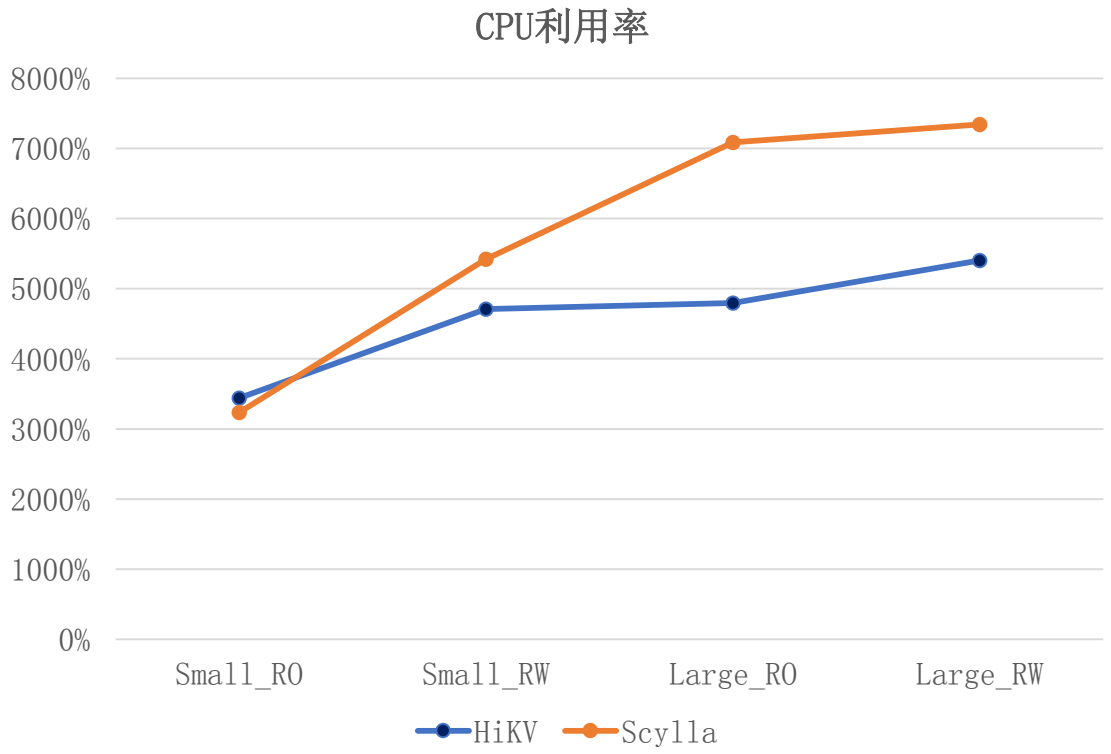
性能比较：HiKV vs. Scylla

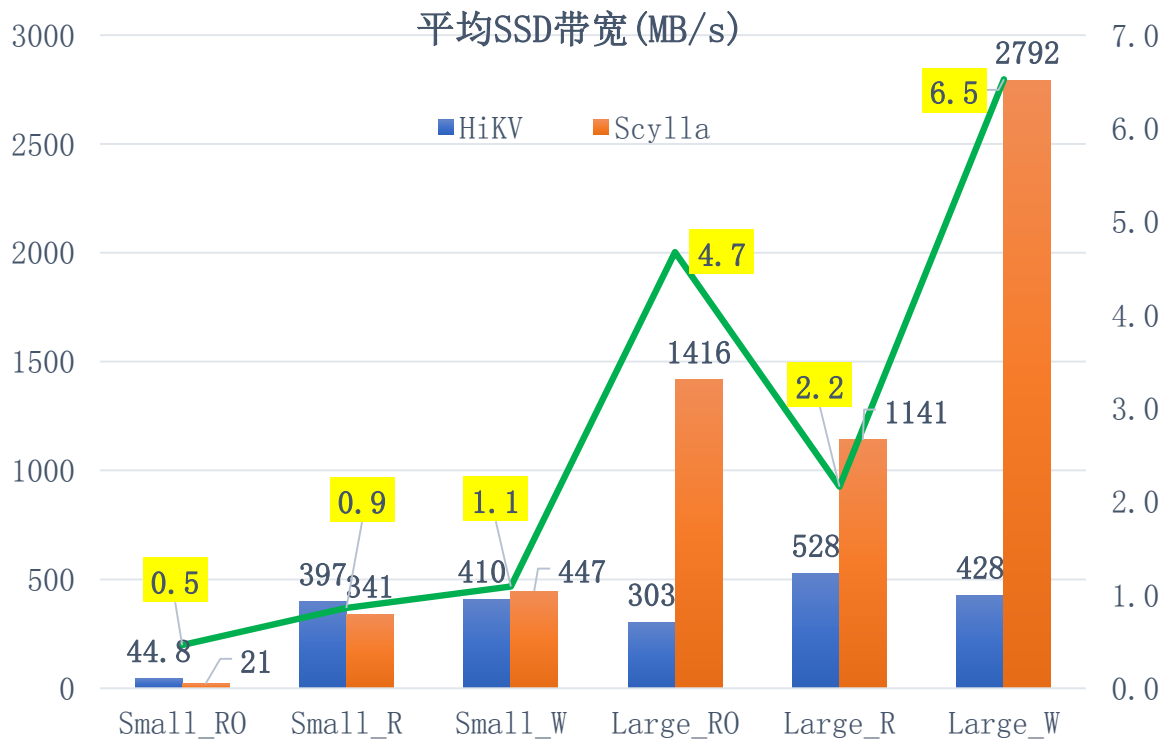




小数据量下，延时相当；大数据量下，HiKV 读写延时稳定，且明显优于Scylla

性能比较：HiKV vs. Scylla





HiKV CPU利用率较Scylla低，且极大降低了读写IO放大

HiKV 研发历程

2017Q4
立项，封装Scylla

2018Q2
发布**v1.0**，支持扩容、多列、视图等

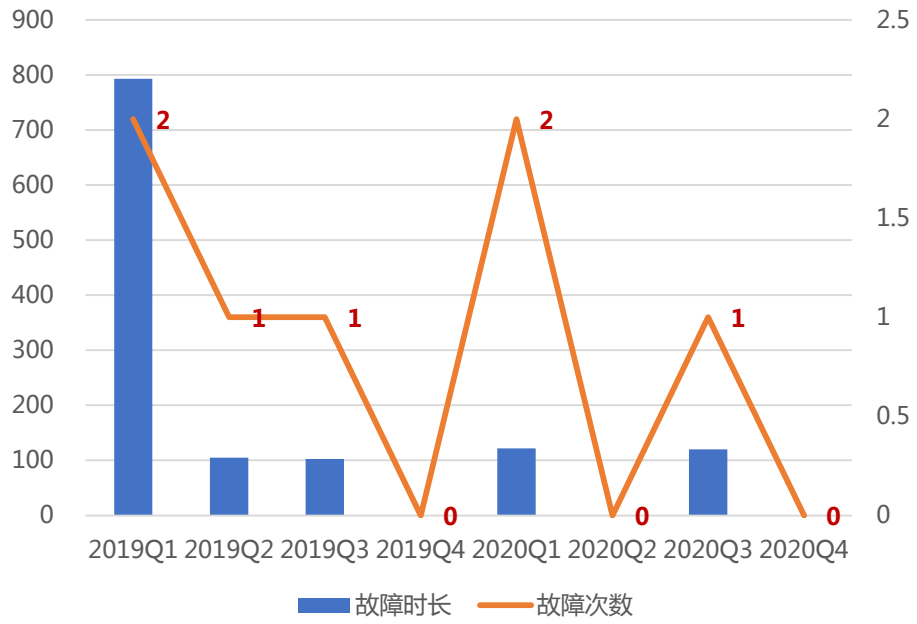
2018Q3
v1.2，支持缓存、GC、定长内存分配器等

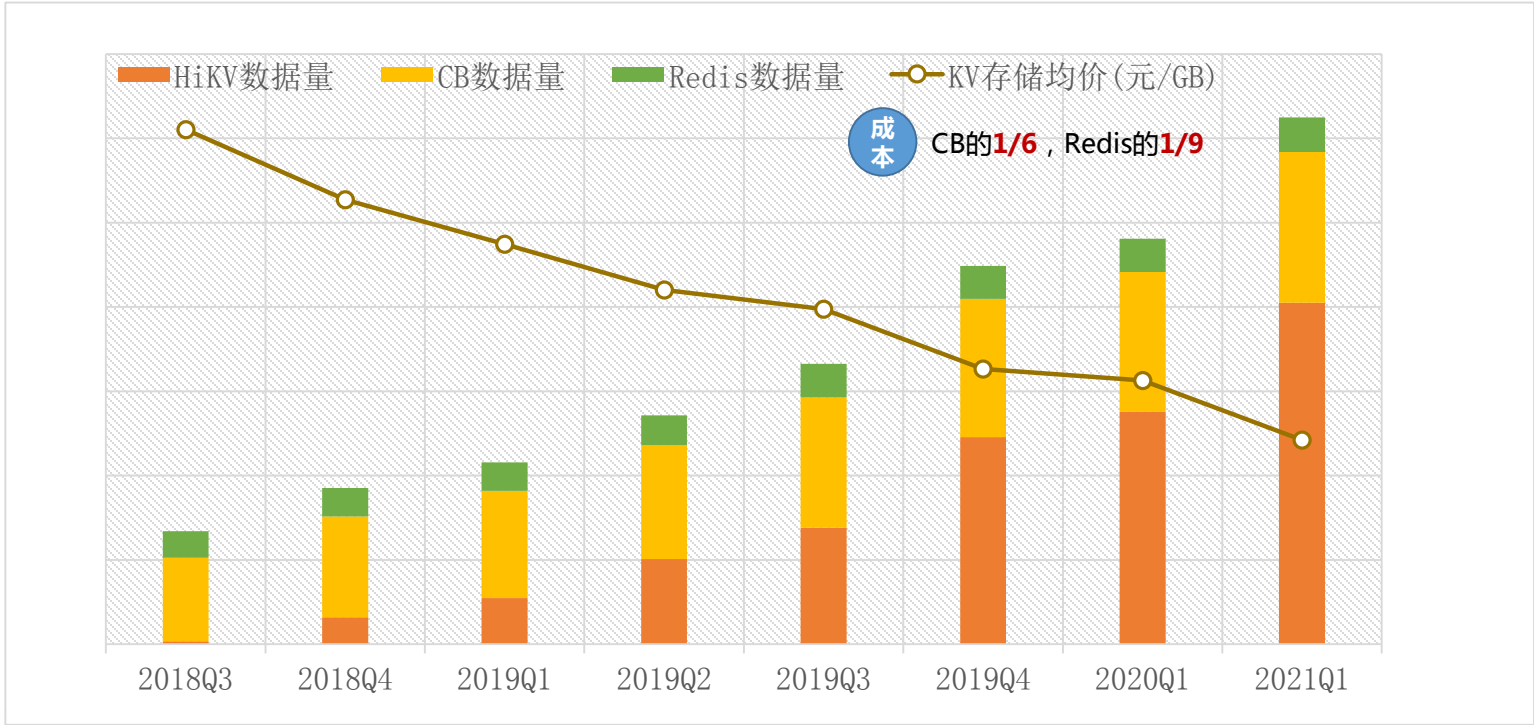
2018Q4
v1.5，内存索引结构优化至72Byte

2019Q2
v2.0，内存索引结构优化至64Byte，checkpoint、写保护等

2020Q1
V2.2，可用性治理，全年故障时长缩短75%

2021Q3
V3.0，存储计算分离HST发布





HiKV 在爱奇艺的应用实践

小KV，高吞吐

- 存储架构：HiKV 单集群65台服务器, 2地, 6DC
- 数据：用户特征、兴趣、短域名、点击历史等
- 数据特征：短数据为主，访问量大，延迟要求高
- 改进措施：GC并发控制、负载均衡优化、客户端压缩、多租户
- 部署方案：11个库22个应用共用集群
- 服务性能：QPS峰值3.5M，P99延迟2~10ms

大KV

- 存储架构：CB + HiKV + 后端兜底服务
- 数据特征：JSON格式，平均长度 8.2KB，访问量低
- 改进措施：客户端压缩，数据量13TB -> 1.1TB

高可用

- 存储架构：CB + HiKV + 双集群 + 2地 3机房部署
- 数据特征：数据量较大，直接影响用户，对可靠性要求高
- 部署方案：采用3机房部署，单个机房故障不影响服务

- 服务器规模 500
- 数据量近 1PB
- 线上单集群最大规模 近100台服务器
- 基本100%覆盖S级业务

提纲

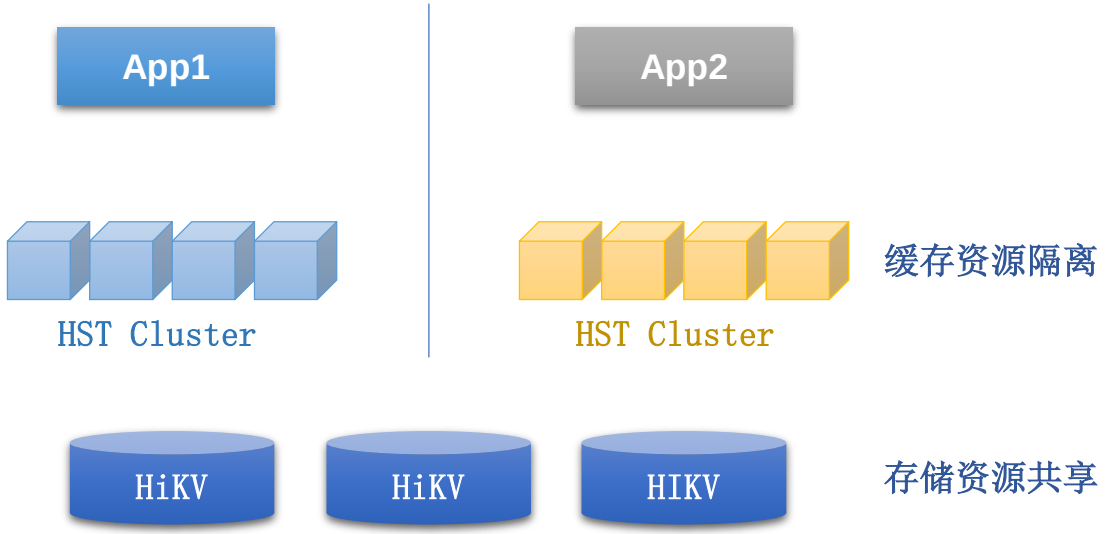
- HiKV 研发动机
- HiKV 关键设计思路
- HiKV 在爱奇艺的应用
- **HiKV 的未来规划**

HiKV 3.0展望：当前的痛点

	优点	缺点	风险	成本	敏捷
服务、存储一体化	• 部署简单易维护 • 业务接入成本较低	• 运维风险较高 • 扩容时间较长	● ●		
全内存索引	• 每次访问最多一次IO	• 服务启动时间较长 • 内存开销较大，与缓存形成竞争资源	●	 ●	
CPU分片架构	• 数据访问无锁	• 前后台任务无法并行 • CPU资源相对紧张		● ●	
多副本	• 数据安全 • 灵活的一致性和可用性策略	• 一致性场景下，多一次节点间通信 • 内存资源成本较高		● ●	
多业务共用大集群	• 管理成本低	• 业务之间没有物理隔离 • 缺乏业务粒度的监控	●		 ●
基于ScyllaDB开发	• 继承了ScyllaDB的架构优点	• code base 升级困难 • bugfix 反向移植困难			● ●
C++	• 性能优秀	• 内存安全性无保障 • 构建系统和依赖管理复杂	●		 ●

HiKV 3.0 解决方案： HiKV Service Tier

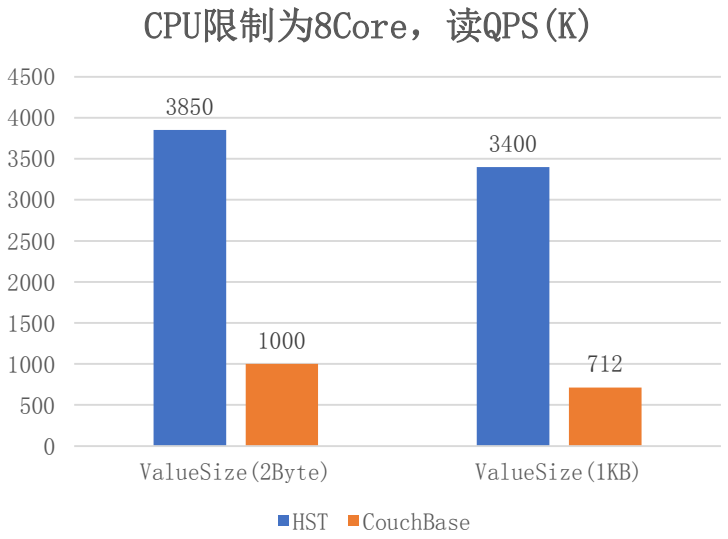
- 服务与存储分离
  - 独立伸缩、快速扩容
  - 物理隔离：各业务使用独立缓存集群
- Loading Cache
  - 数据一致性
  - 降低业务接入成本
- 单副本
  - 响应快、成本低
- Rust语言
  - 媲美C++的性能
  - 内存安全性
  - 现代的构建系统和依赖管理能力



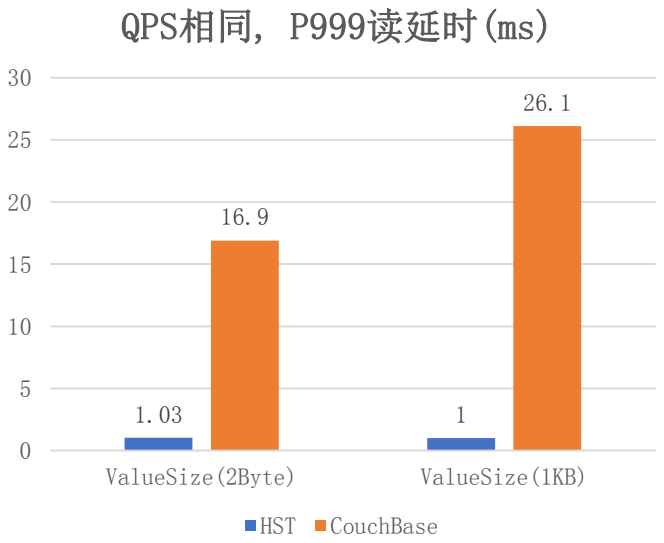
HiKV 3.0：HST性能

测试环境：2 × Intel Xeon Gold 5218 (2.30GHz, 16核32线程)，384G

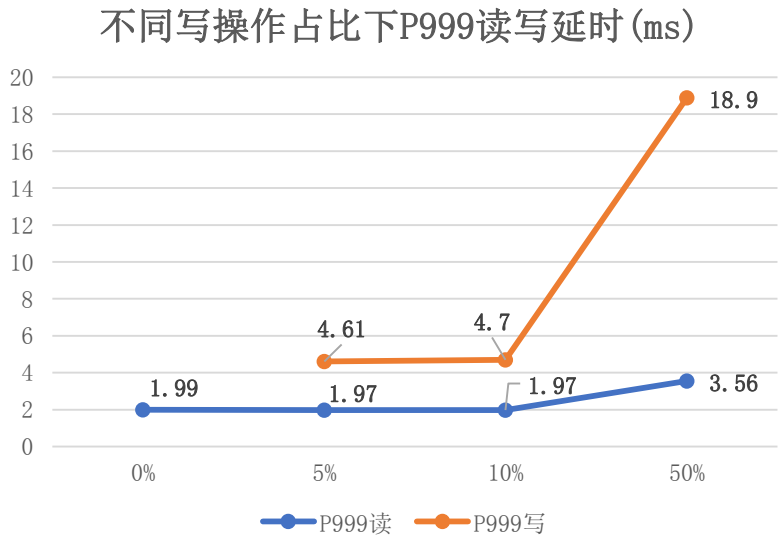
- couchbase-server-7.0.1, 3台物理机，3节点 × (无副本 + 8core)
- hst-server 0.4.1-alpha.4, 3台物理机，3节点 × (8分片 + 8core)，后端6节点HiKV集群



在相同 CPU 利用率限制下，Couchbase 能支持的 QPS 仅为 HST 的 20%～26%



在相同 QPS 限制下（1000/800K），Couchbase 读延迟高于 HST，长尾延迟现象更加突出



相比HiKV读延时缩短至50%，写延时增加明显

HiKV 3.0：进展与规划

已支持的特性

- 多线程支持，性能随线程数线性增长
- 基于 hash map 的缓存实现
- 从 HiKV 加载未命中数据
- 支持连接复用：CQL 4 协议 + 自定义请求/响应类型
- 数据淘汰策略：LRU
- 支持的命令：GET，SET，TTL，DEL
- 监控系统：VictoriaMetrics + Grafana + Hubble
- 集群管理接口
- Rust驱动和命令行客户端
- Java驱动
- 性能测试工具：hst-bench，YCSB

后续规划

- 更多数据结构：Hash，List，Set
- C/C++驱动
- 运行时配置变更
- 一致性和可用性配置
- 云原生支持

- **开源计划 (准备中)**
  - HiKV&HST 代码开源
  - 深入技术系列分享

Q&A

爱奇艺技术产品团队

简单想，简单做


THANKS
时序
开源
自研
文档
Spark
HANA
SQL
OLAP
Oracle
Hadoop
MPP
DB2