SACC 2016 第八届中国系统架构师大会
SYSTEM ARCHITECT CONFERENCE CHINA 2016
架构 创新之路

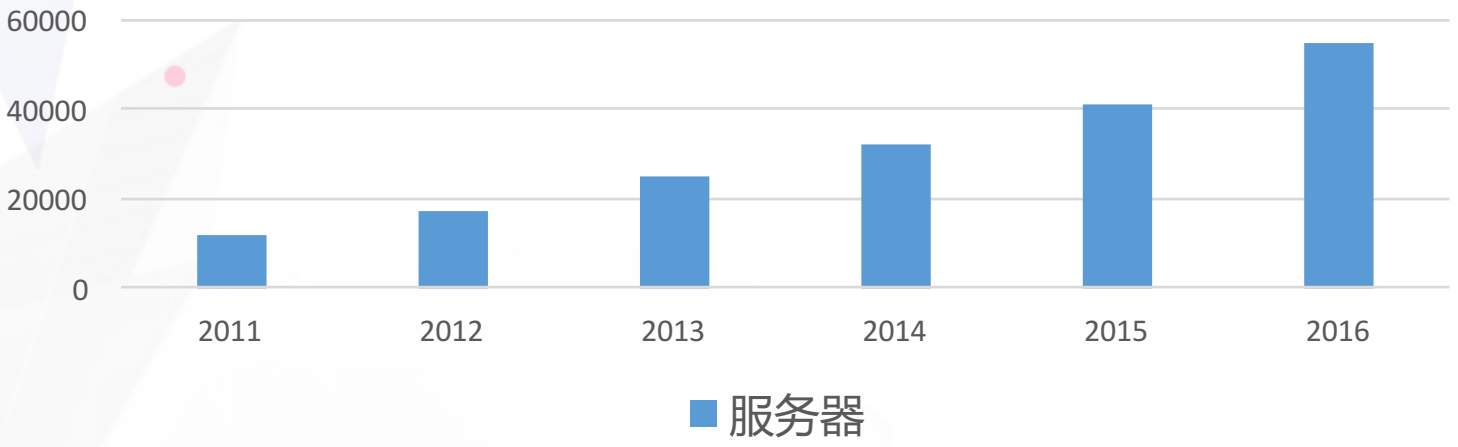
微信运维监控 ——

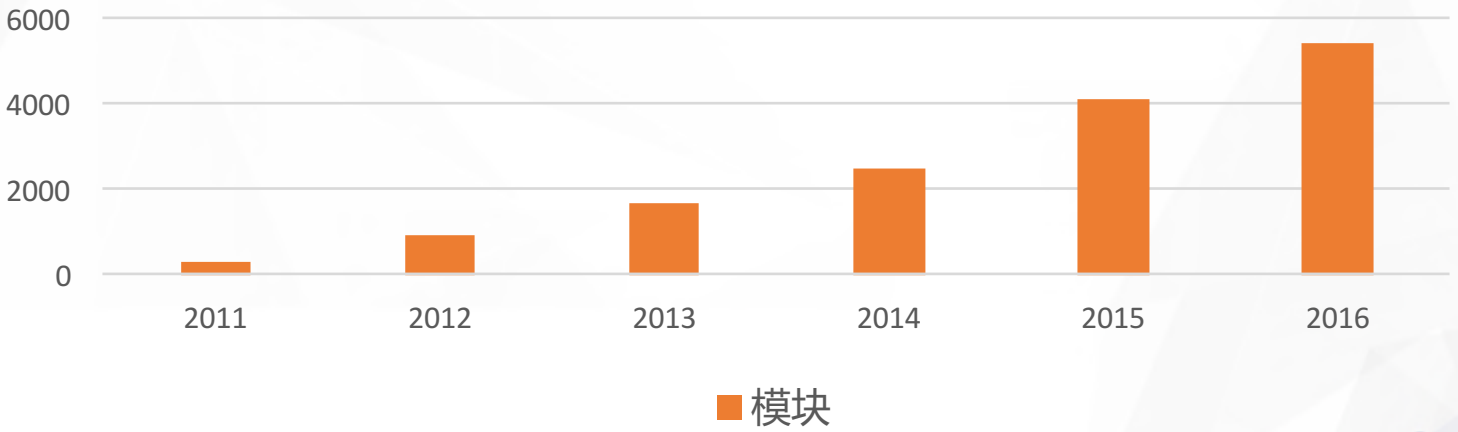
海量监控数据上报及存储设计实践

微信运维中心 陈晓鹏

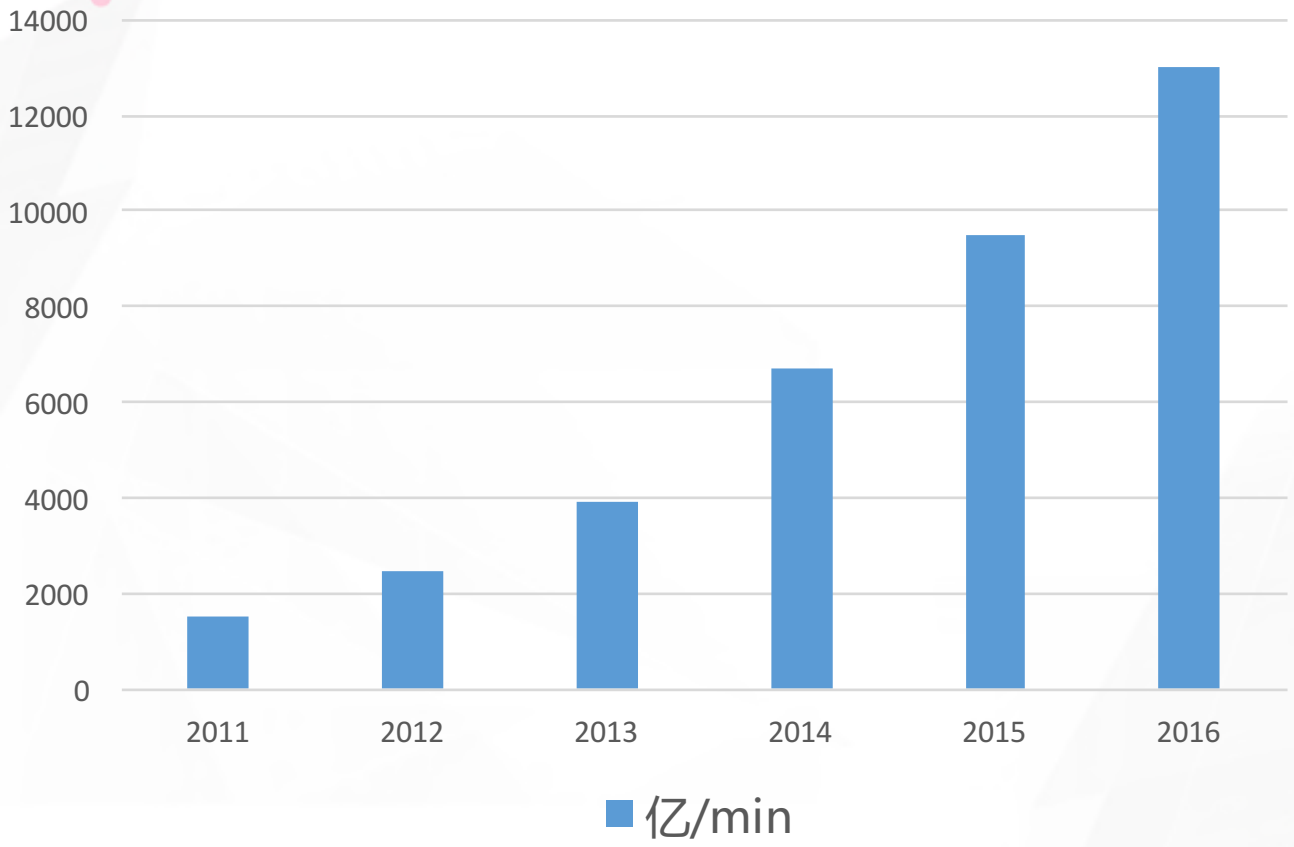
如果运维监控数据处理太慢...

微信后台系统现状


60000
40000
20000
0
2011
2012
2013
2014
2015
2016
服务器


6000
4000
2000
0
2011
2012
2013
2014
2015
2016
模块

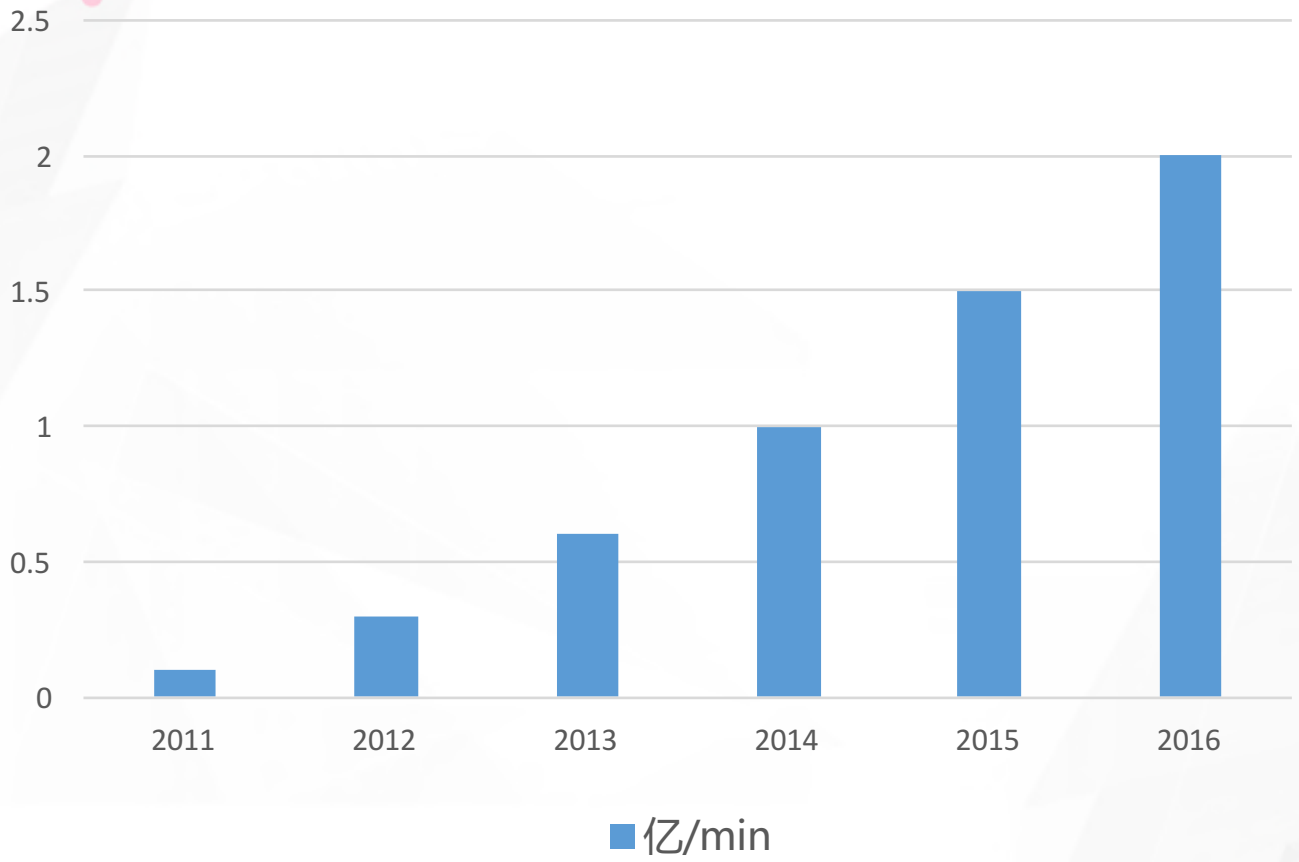
微信监控日志上报量 —— 1万亿/min



- 存储容量不够
- 统计延迟严重

每分钟万亿级监控数据，如何上报、汇总？

微信监控数据存储量 —— 2亿/min



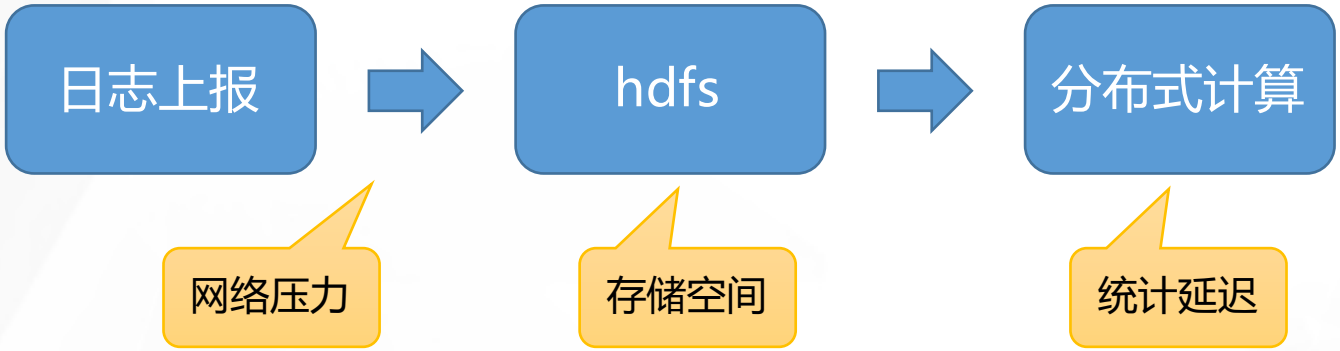
- 如何进一步提高TSDB的读性能？

每分钟亿级时间序列数据，如何实现存储，

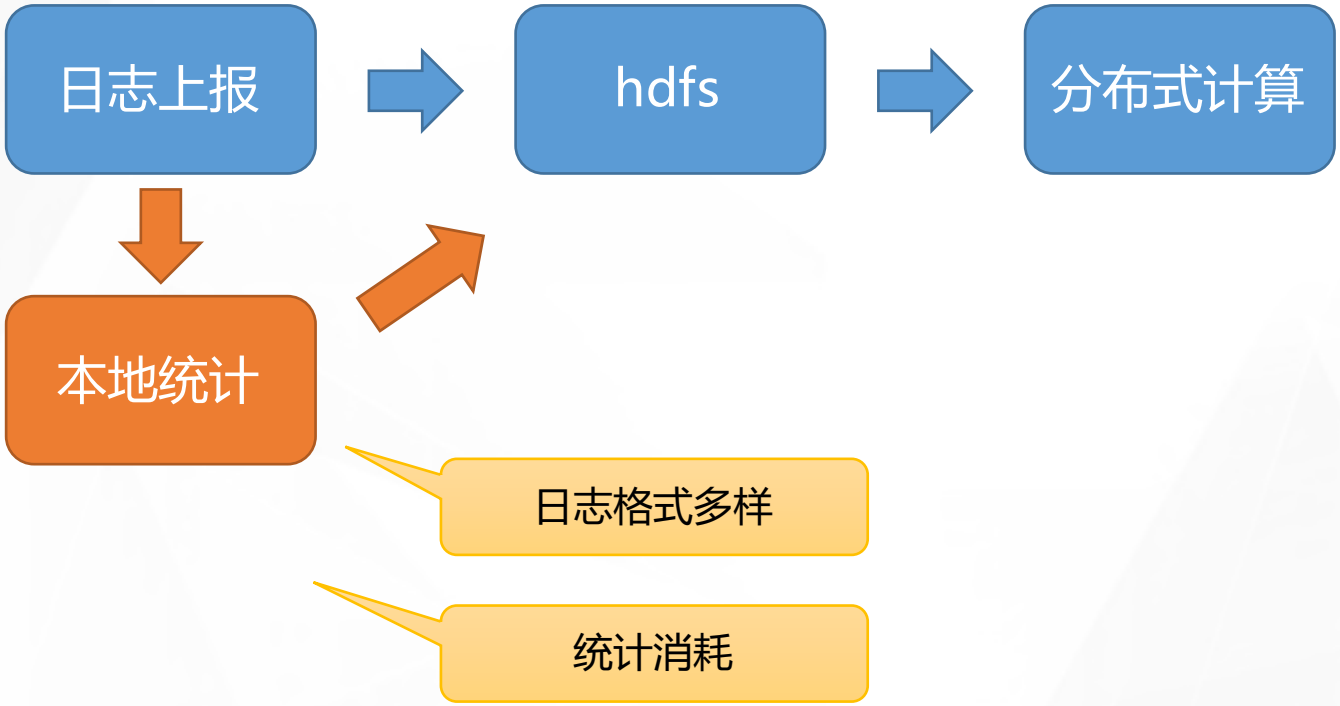
如何保证N年跨度的数据读取性能？

微信轻量监控数据上报框架

常见监控数据上报/汇总方案



常见监控数据上报/汇总方案



放弃日志这种上报形式！

微信后台轻量数据上报框架

- 数据分类
  - 业务数据 —— 复杂、高延迟
  - 监控数据 —— 简单、低延迟
- 简化监控数据
  - 统一数据格式
  - 简化计算规则 —— 累加、平均

统一数据格式

- ID Key Value：
  - ID： 0 ~ 128k-1
  - Key： 0 ~ 127
  - Value： uint32_t

轻量上报/汇总框架

- type __sync_fetch_and_add (type *ptr, type value)
- bool __sync_bool_compare_and_swap (type *ptr, type oldval, type newval)
- 支持计算规则：累加、设置新值、设置最大值

轻量上报/汇总框架

- CPU消耗极小
- 可实现秒级、实时数据采集
- 可简化数据存储、汇总

微信高性能监控数据存储

监控数据（时间序列）特点

- 数据格式：

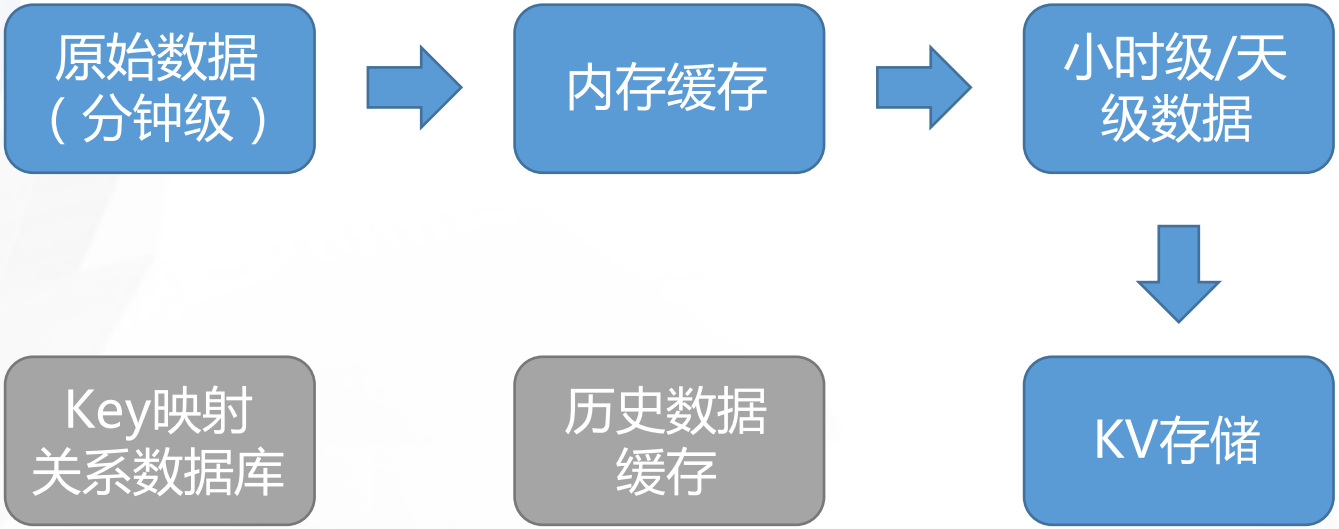
  Key + Time + Value

- 数据量大：

  1天 = 1440min， 2亿/min = 2880亿/天

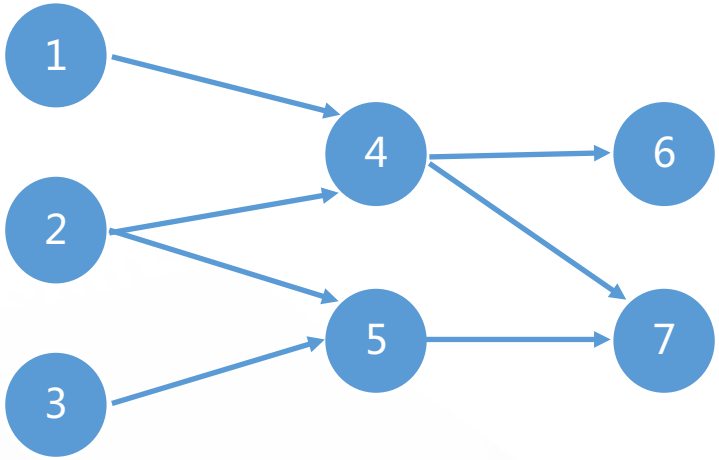
- 大量读取历史数据

时序数据库设计思路



- 历史数据查询性能不足
- 数据缓存命中率不高
- 复杂关系数据查询效率低

多维度Key的作用



- 当4失败总数上升时，如何定位故障机器、接口？

A模块 A1机器 → B模块 B2机器 X1接口

调用数

失败数

耗时

提高历史数据查询性能

- 定制能快速查找历史数据的index：
  - 分表、按时间分区
  - 每个分区独立生成index

旧版微信监控数据存储

```
CREATE TABLE `idkey_0` (
  `time` int(10) unsigned NOT NULL,
  `key1` int(10) unsigned NOT NULL,
  `key2` int(10) unsigned NOT NULL,
  `key3` int(10) unsigned NOT NULL,
  `v0` bigint(20) unsigned DEFAULT '0',
  `v1` bigint(20) unsigned DEFAULT '0',
   ...
  `v718` bigint(20) unsigned DEFAULT '0',
  `v719` bigint(20) unsigned DEFAULT '0',
  PRIMARY KEY (`key1`,`key2`,`key3`,`time`),
  KEY (`key3`,`key2`,`key1`,`time`)
) ENGINE=MyISAM
PARTITION BY RANGE (time)
(PARTITION P20150327 VALUES LESS THAN (14274144
 PARTITION P20150330 VALUES LESS THAN (1427673600) ENGINE = MyISAM,
...
 );
```

分表

Key * n

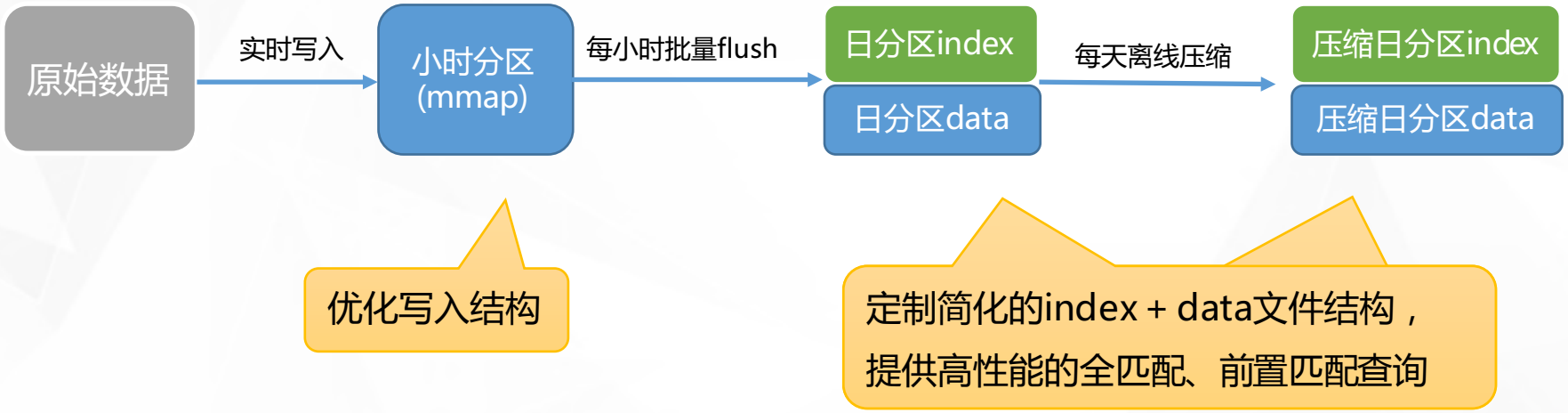
Value * 720

多index，支持全匹配、前置匹配

分区

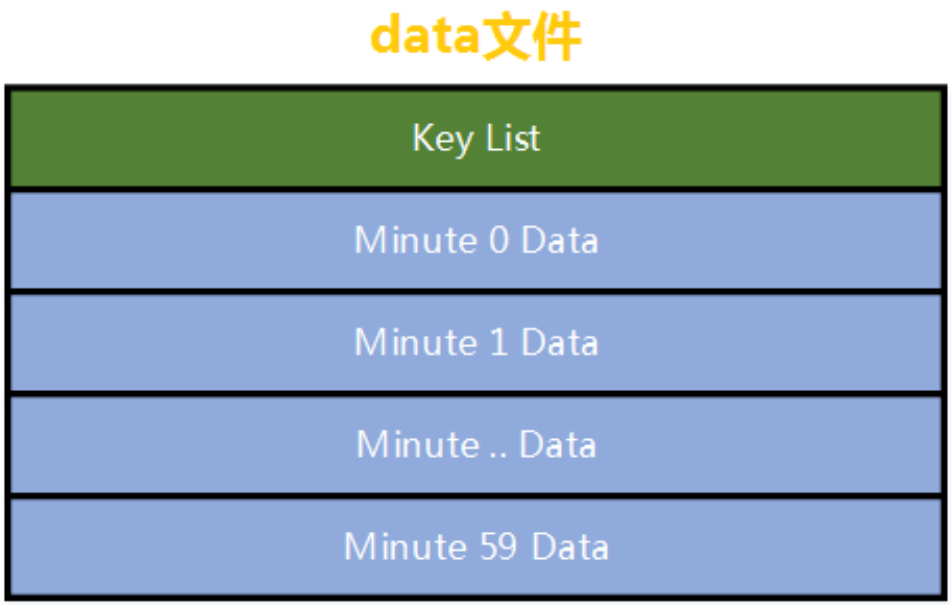
写200w/min，读100w/min

微信新版存储方案


原始数据
实时写入
小时分区
(mmap)
每小时批量flush
日分区index
日分区data
每天离线压缩
压缩日分区index
压缩日分区data
优化写入结构
定制简化的index + data文件结构，
提供高性能的全匹配、前置匹配查询

小时分区（mmap）结构

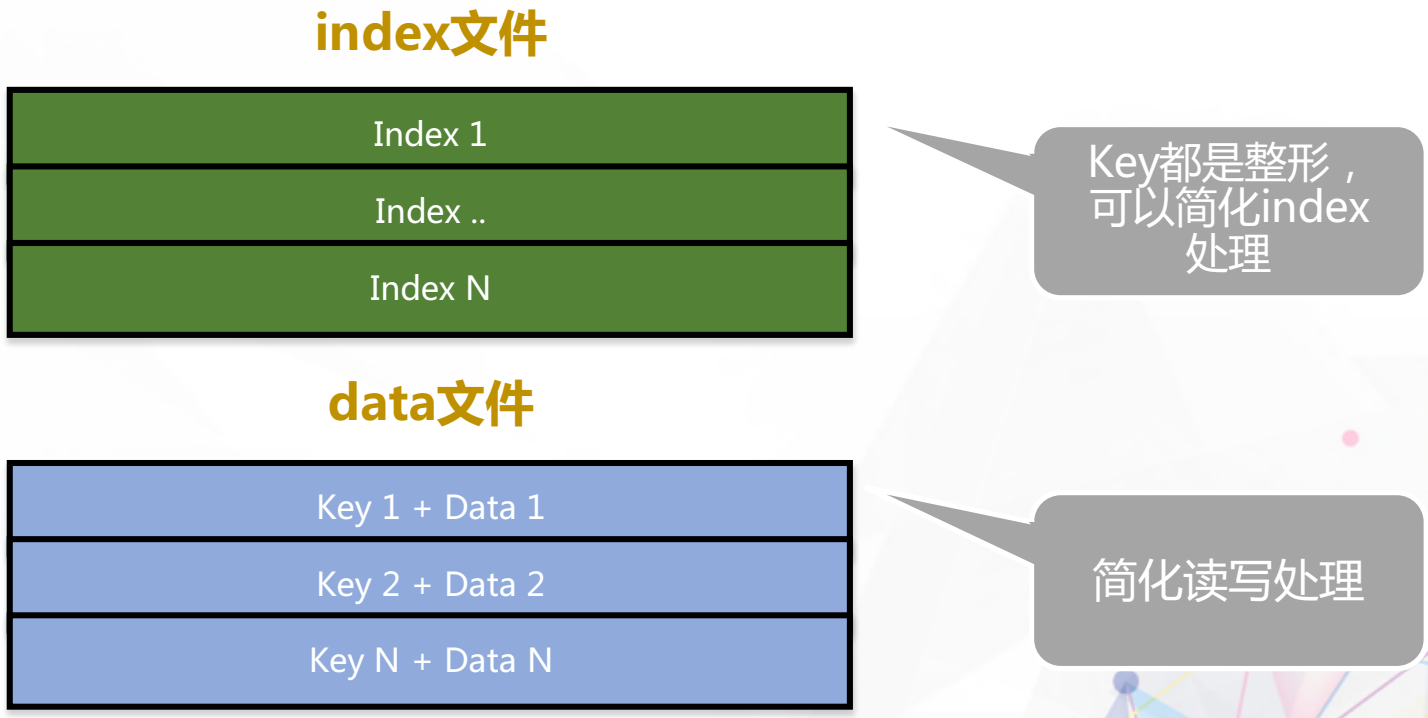
- 1个meta文件 + N个data文件组成
- 每个data文件大小固定，使用mmap映射



不同key同一分钟的数据相邻，减少单次flush数据量。

日分区（压缩/未压缩）文件结构

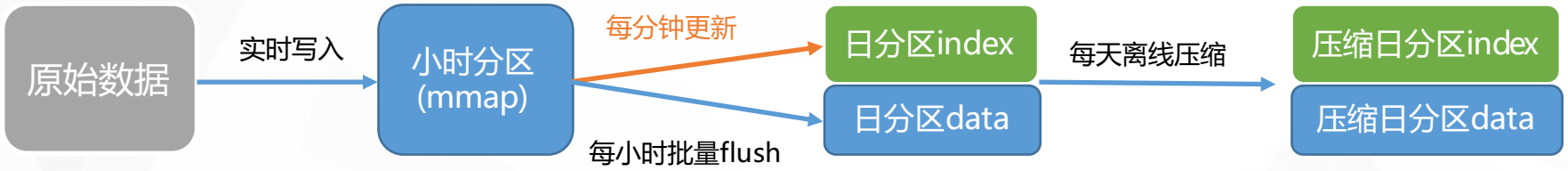
- 1个meta + 1个index + 1个data文件



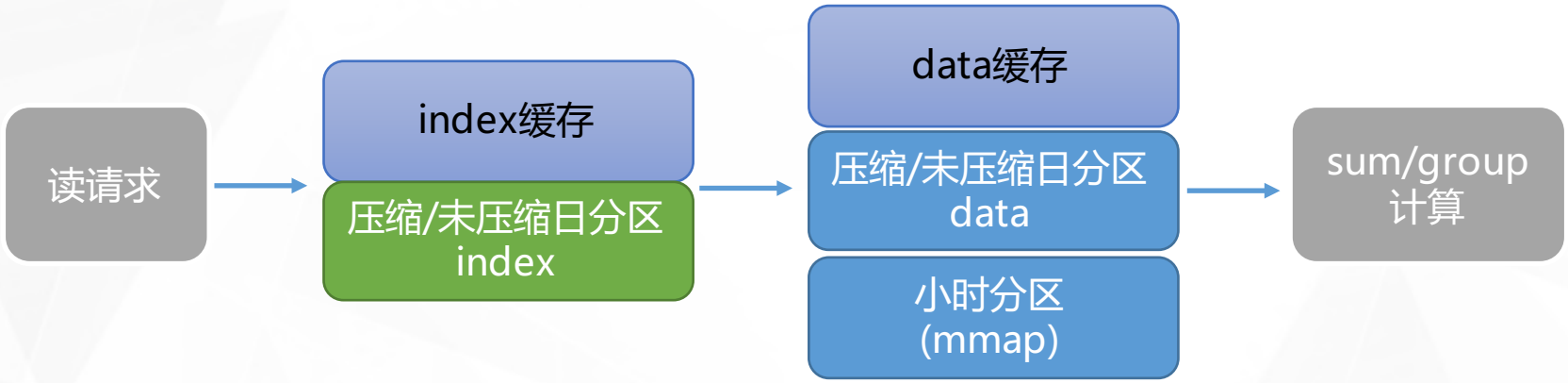
index算法 —— 折中查找

- 全匹配查询：

- 前置匹配查询：


原始数据
实时写入
小时分区
(mmap)
每分钟更新
每小时批量flush
日分区index
日分区data
每天离线压缩
压缩日分区index
压缩日分区data

数据读取流程


读请求
index缓存
压缩/未压缩日分区
index
data缓存
压缩/未压缩日分区
data
小时分区
(mmap)
sum/group
计算

单机性能

- 写性能：

  小时分区(mmap)： 1kw/s（6亿/min)

  未压缩日分区： 35w/s（12.6亿/hour)

- 读性能（单分区100w条记录）：

  未压缩data性能： 30w/s

  压缩data性能： 110w/s

THANKS

SequeMedia 盛拓传媒 | ChinaUnix | ITPUB