腾讯云 | 云+社区 intel

「云+社区沙龙online」

海量挑战：腾讯云ES可用性及性能优化实践

张彬

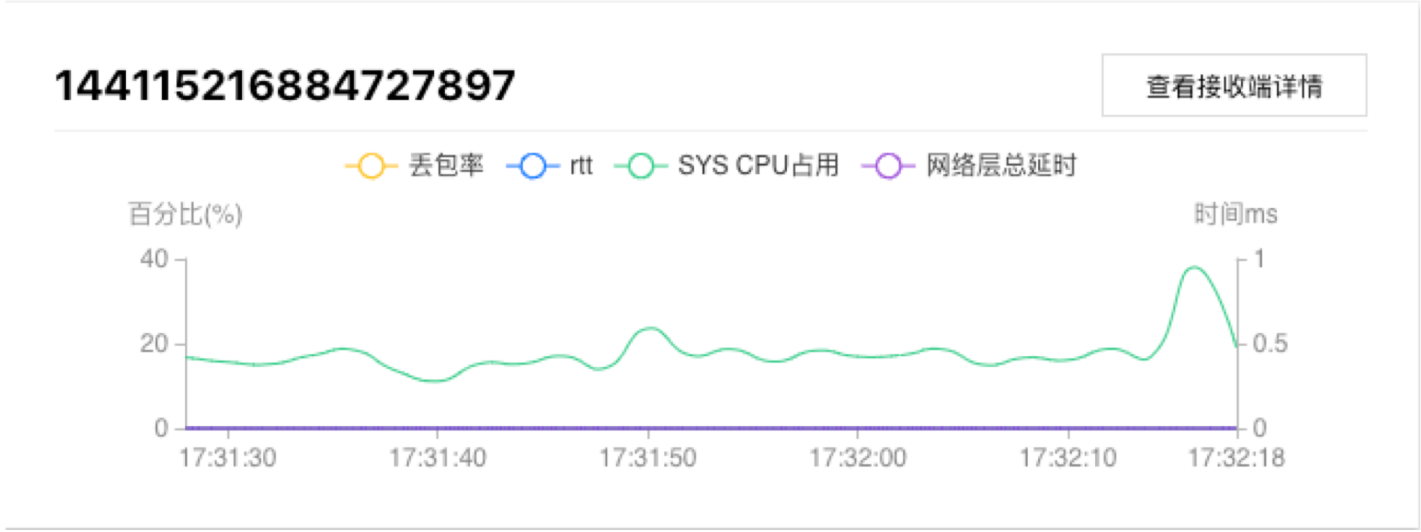
腾讯云专家工程师

关注「云加社区」

回复「加群」进直播交流群

目录

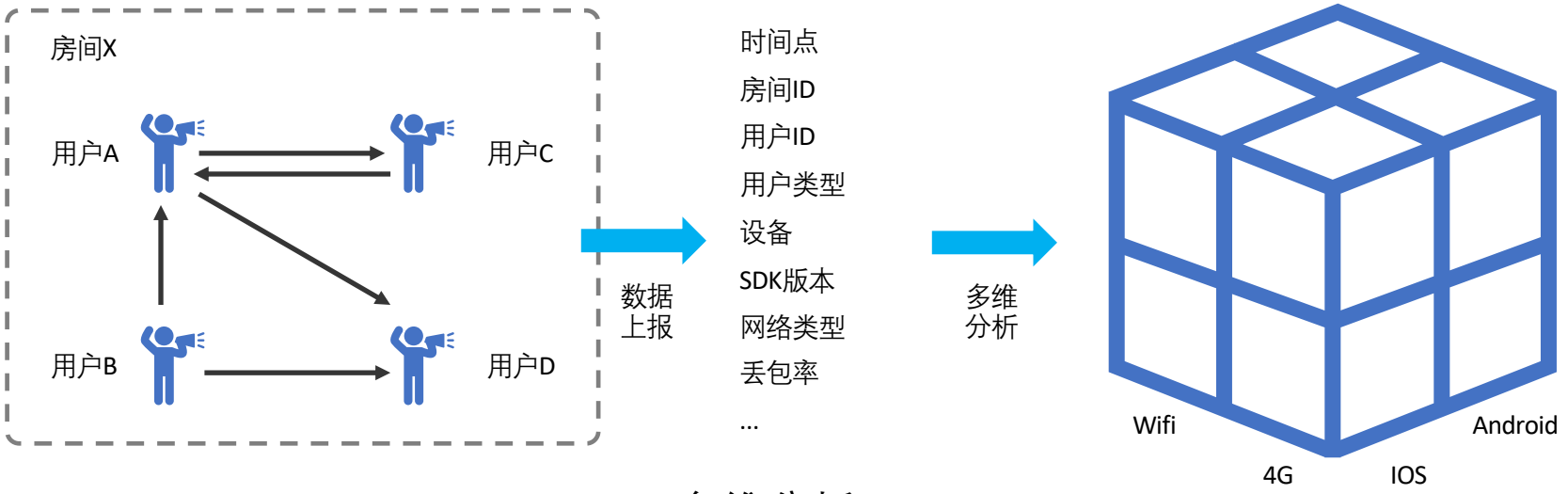
一、ES在腾讯会议中的应用—腾讯会议质量分析系统



会议质量数据查询

- CPU占用率
- 丢包率
- 网络延时
- 网络切换
- etc...

质量保障团队根据腾讯会议用户报障查看分析问题
通过多维分析技术，结合看板，阀值预警等提前发现问题



多维分析

多维分析

- 某版本问题
- 某设备类型问题
- 某地域网络延时大

一、ES在腾讯会议中的应用—痛点挑战

可用性

集群经常雪崩
SLA难以保障，达不到4个9

性能

写入慢，数据堆积
数据同步延迟高
平均查询延迟达不到要求

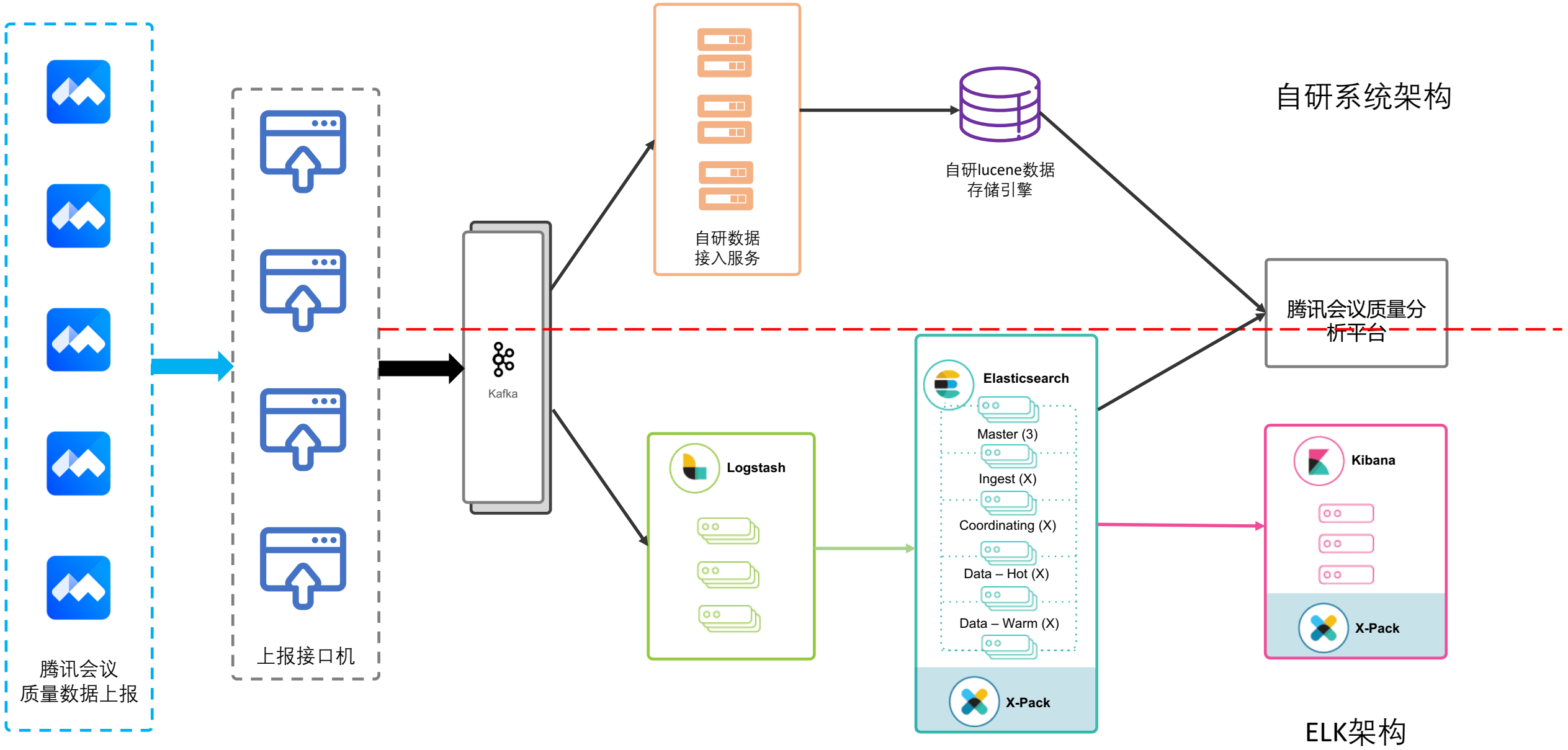
扩展性

扩展性差
人工运维
数据分布不均

易用性

是否能快速切换
导入导出组件
自运维能力

一、ES在腾讯会议中的应用—系统架构图



一、ES在腾讯会议中的应用—更多场景应用

数据分析

bilibili

蘑菇街 我的买手街

TAL 好未来

富途证券 FUTU5.COM

搜索

猎豹移动

聚美优品 JUMEI.COM

英孚教育 English First

鳳凰網 IFENG.COM

运维监控

知乎 有问题 上知乎

享物说

RIOT GAMES

mobike 摩拜单车

数据库加速

NIO 蔚来

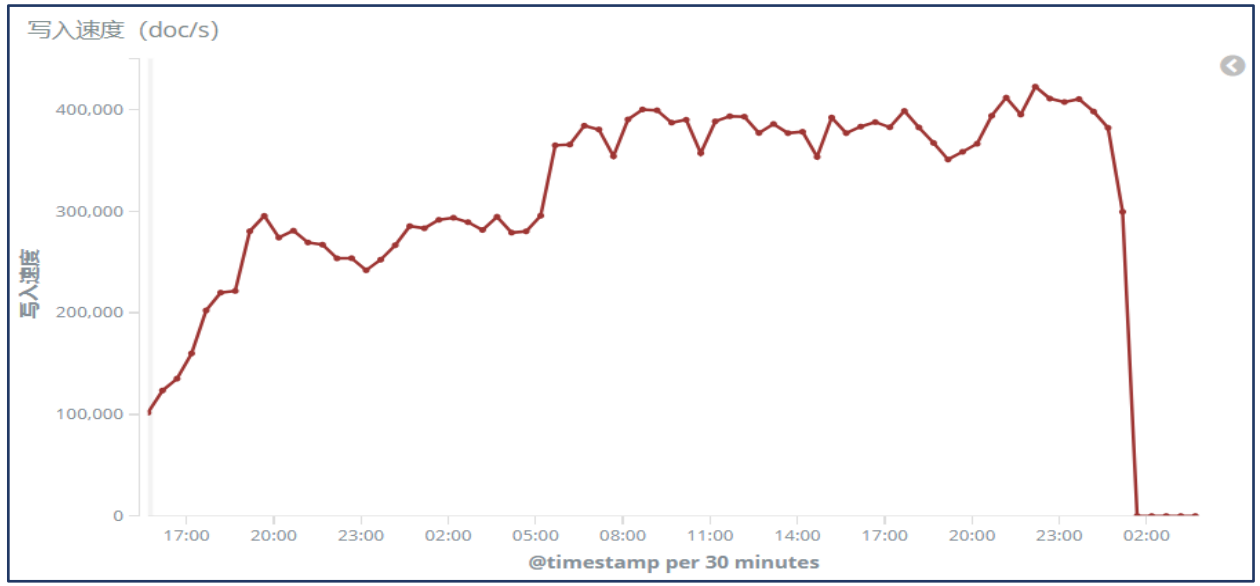
丙晟科技 BEYOND SCIENCE

乐逗游戏 iDREAMSKY GAMES

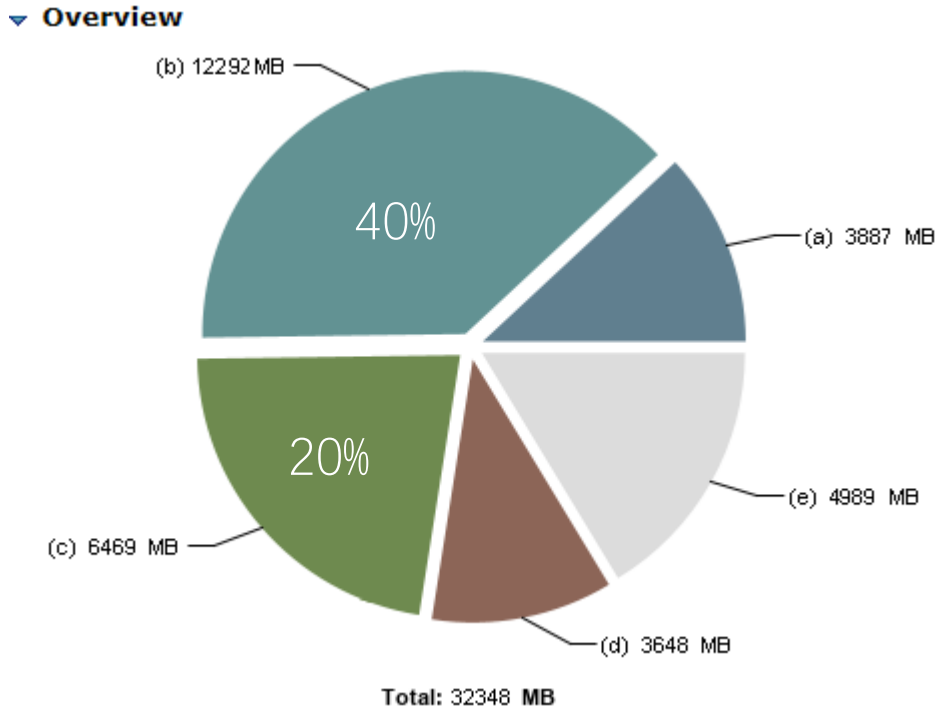
阅文集团 CHINA READING

目录

二、 高可用及性能方向的优化—高并发请求



内部某日志集群，写入量一天突增5倍



集群现象：

- 多个节点 **Old GC** 卡住脱离集群
- 集群状态 **RED**（数据主副本分片均不可用）
- 集群停写

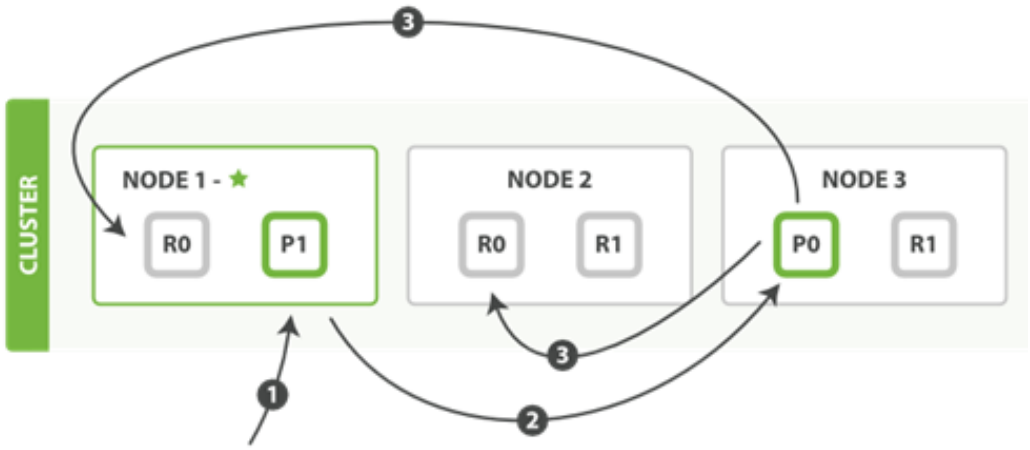
JVM内存分布：

- 反序列化**前**的批量写入请求 **20%**（Netty 缓存）
- 反序列化**后**拆分的分片级批量写入请求 **40%**

二、 高可用及性能方向的优化—高并发请求

场景分析

High Level 写入流程

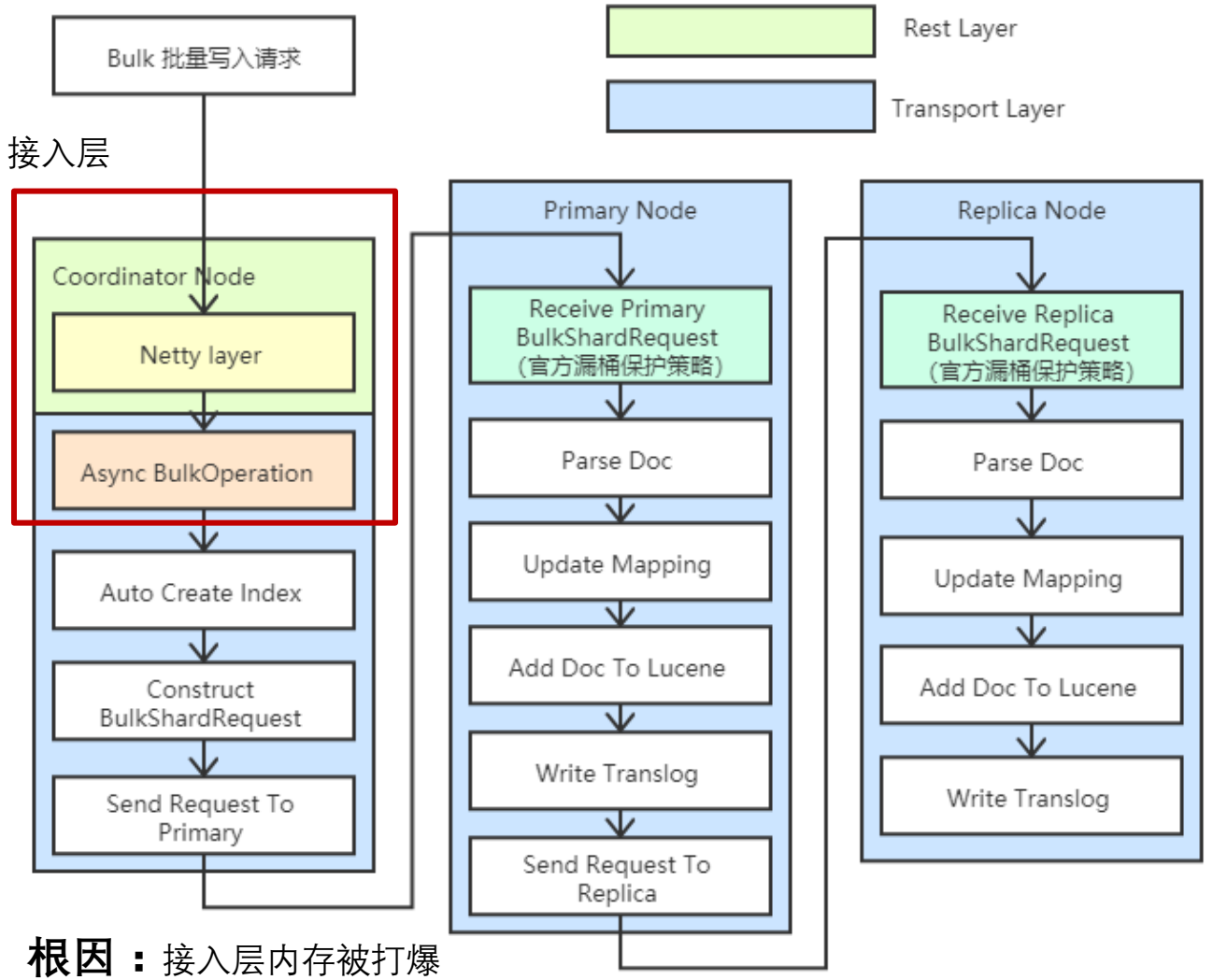


❶ 用户写入请求发送至任意节点

❷ 请求转发至**主分片**所在节点处理

❸ 主分片处理完并发转发给**从分片**处理

Internal 写入流程分析

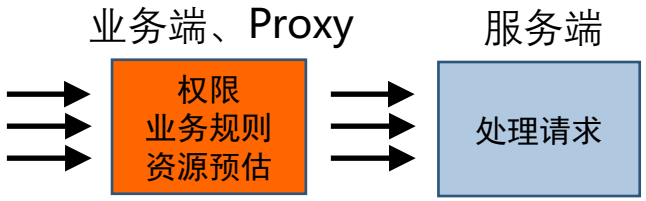


根因：接入层内存被打爆

二、 高可用及性能方向的优化—高并发请求

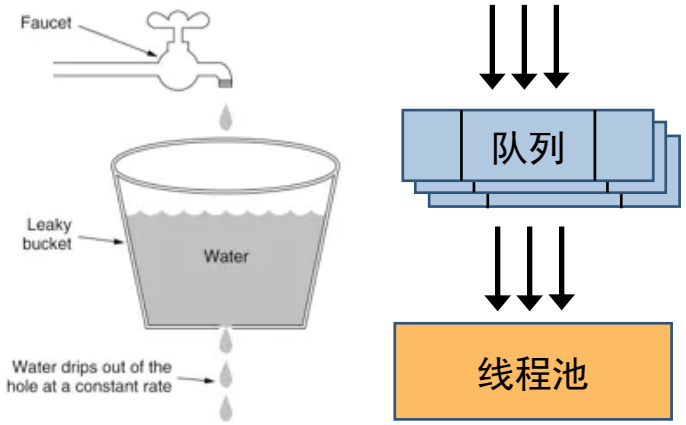
优化方案 - 服务限流

- 除了并发请求数量，还要能精准控制内存资源
- 通用性要强，能作用于各个层级实现全链限流



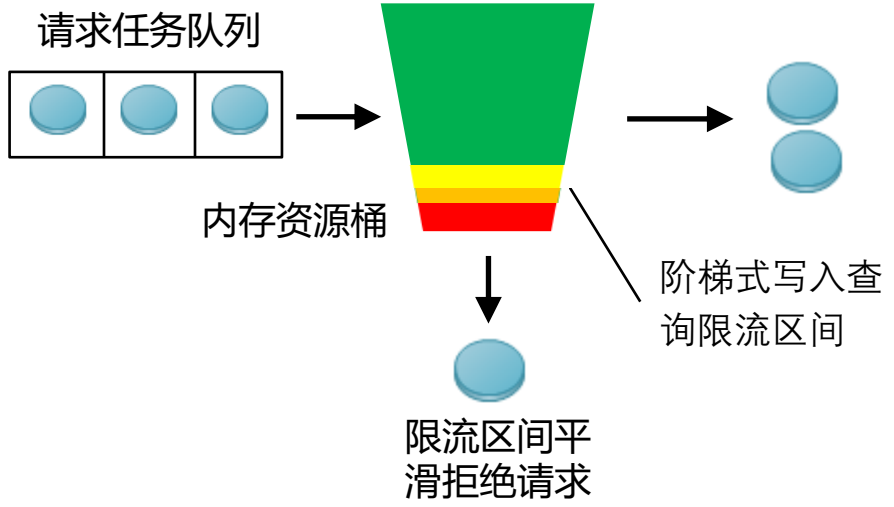
业务端 / Proxy限流

- 难以准确预估资源使用
- 场景适应能力弱，运维成本高



基于请求数的漏桶（原生方案）

- 基于请求数量控制，不能控制资源
- 只作用于传输层，限流链路不完整



基于内存资源的漏桶（优化方案）

- 全链限流，接入层 + 传输层
- 服务降级，阶梯式限制写入、查询

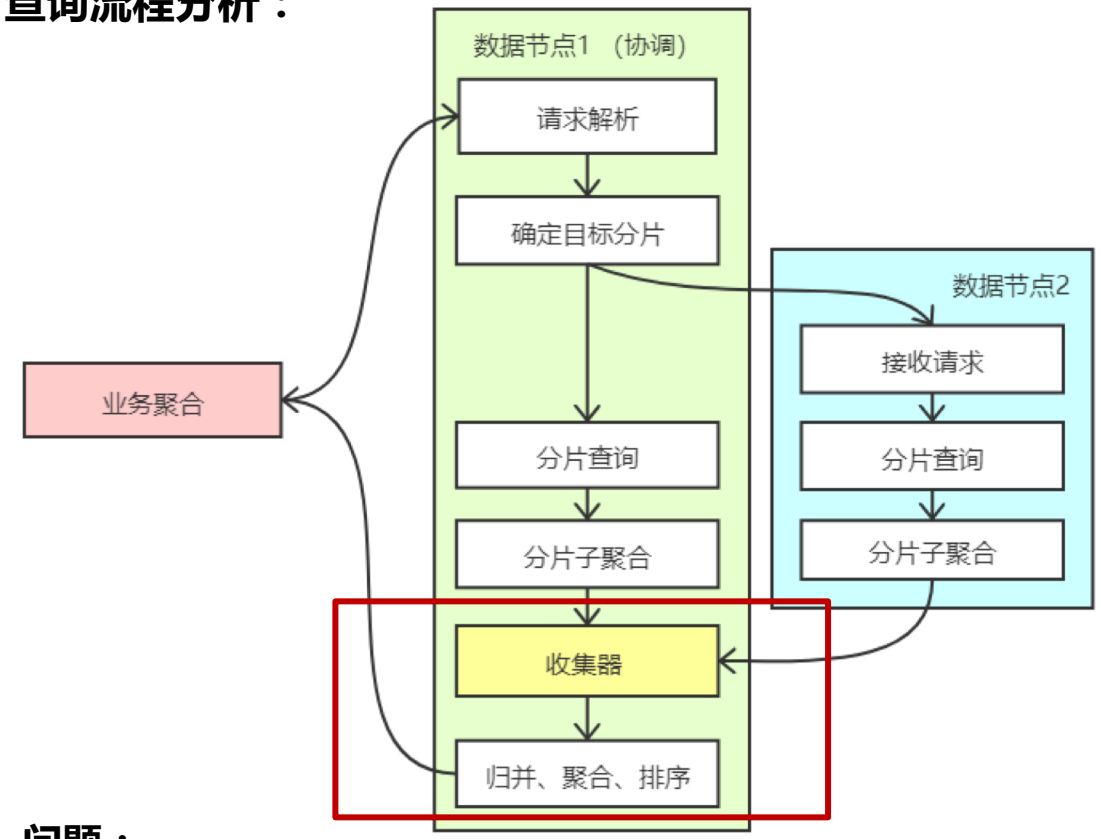
二、 高可用及性能方向的优化—大聚合查询

场景分析

两层嵌套聚合，结果百万级

```
{
  "aggs": {
    "dimension1": {
      "terms": {
        "field": "user",
        "size": 1000000
      },
      "aggs": {
        "dimension2": {
          "terms": {
            "field": "domain",
            "size": 1000000
          },
          "aggs": {
            "top1": {
              "top_hits": {
                "size": 1
              }
            }
          }
        }
      }
    }
  }
}
```

查询流程分析：



问题：

- 大量汇聚结果反序列化后内存膨胀
- 二次汇聚产生新的结果集打爆内存

二、 高可用及性能方向的优化—大聚合查询

优化方案 - 内存膨胀预估 + 流式检查

原生方案：限制最大返回结果桶数

```
ElasticsearchStatusException[Elasticsearch exception
[type=search_phase_execution_exception, reason=]]; nested:
ElasticsearchException[Elasticsearch exception
[type=too_many_buckets_exception, reason=Trying to create too many buckets. Must be less than or equal to: [10000] but was [10001]. This limit can be set by changing the [search.max_buckets] cluster level setting.]];
```

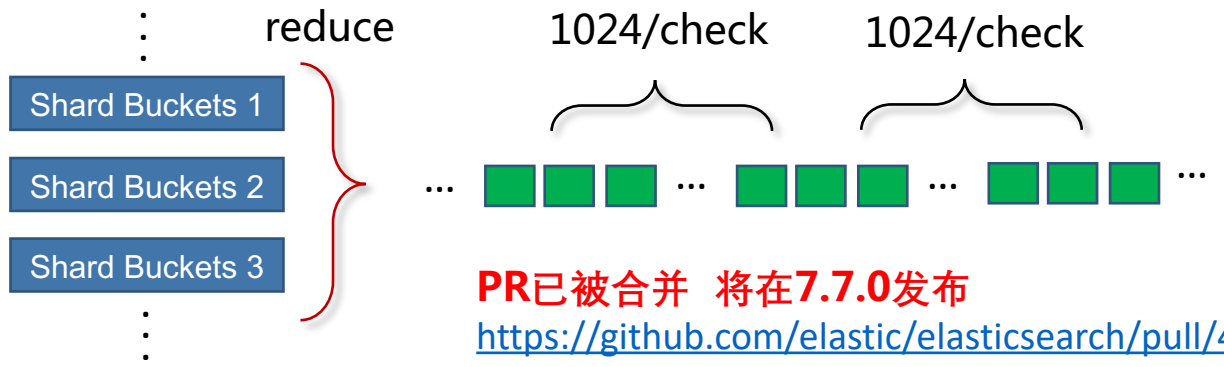
挑战：

- 分析场景结果桶数十万、百万是常态，默认1万不够
- 调整不灵活，大了内存会崩掉，小了满足不了业务需求

第一阶段：根据经验值计算**内存膨胀系数** $\lambda = \frac{\text{真实内存消耗}}{\textit{Byte}\text{ 流大小}}$

如果当前 JVM使用内存 + 预估所耗内存 已超过阈值则直接熔断

第二阶段： reduce 过程中流式检查桶数，每增加固定数量的桶检查一次内存，如果已经超限则直接熔断。



PR已被合并 将在7.7.0发布
https://github.com/elastic/elasticsearch/pull/46751

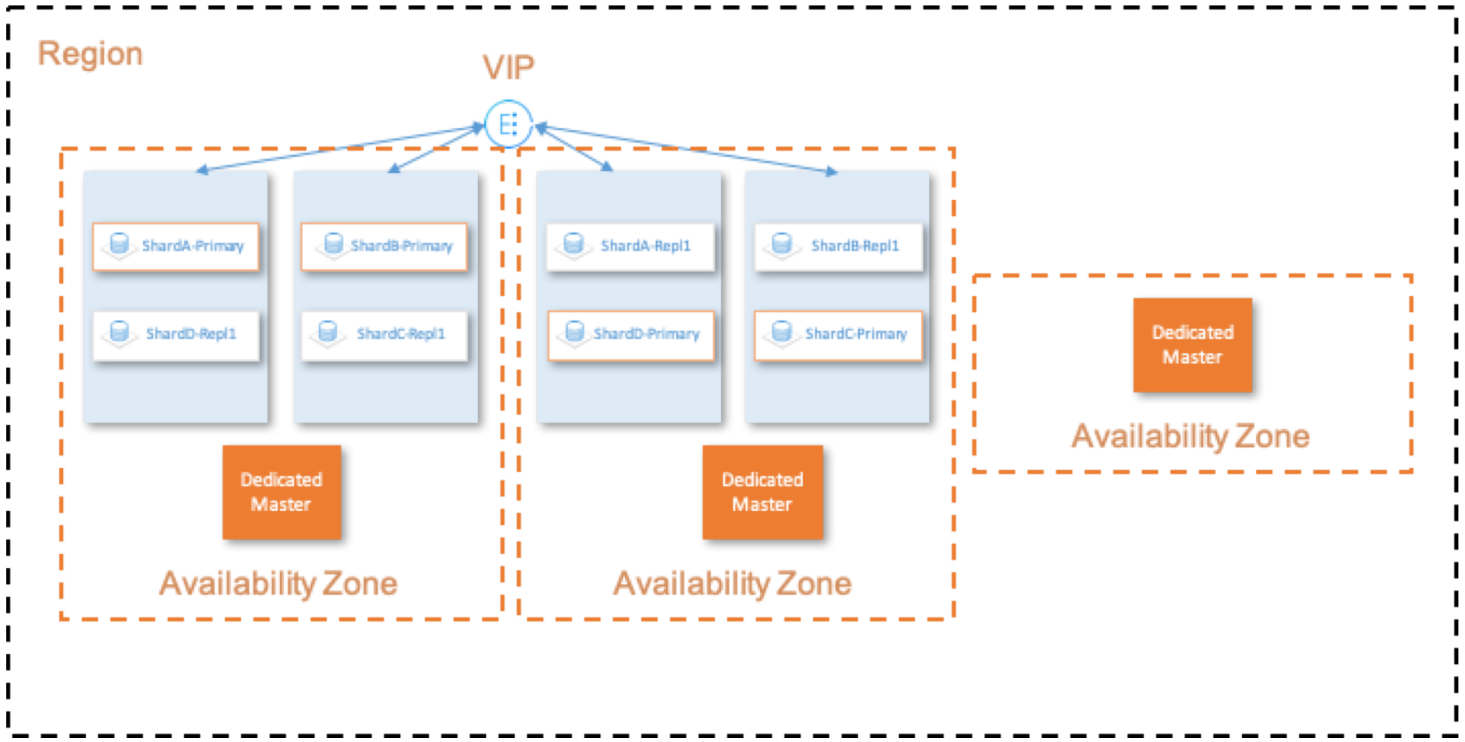
优点：用户无需关心最大结果桶数，只要内存足够就可以最大化满足业务需求

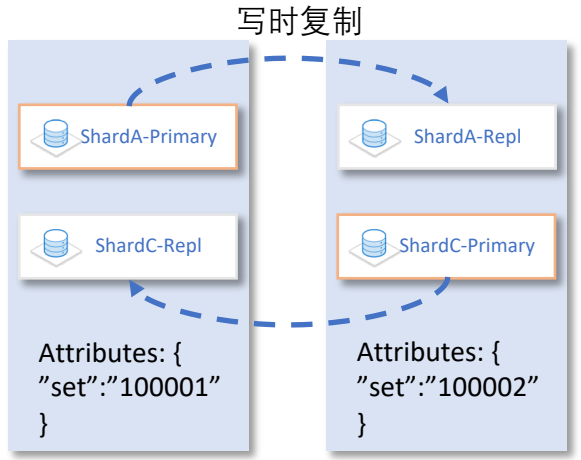
不足：大请求被拒绝，牺牲了用户的查询体验（batch reduce 缓解）

二、 高可用及性能方向的优化—多可用区部署

场景分析

某电商商品搜索业务，ES服务作为召回系统，是业务链路上的关键路径
需要保证可用性达到**99.99%**





开启分片分配可用区感知

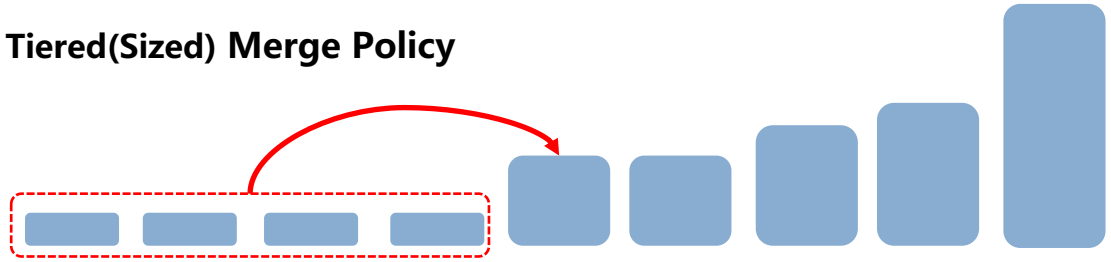
cluster.routing.allocation.awareness.attributes: set

多个副本平均分配到各可用区

二、 高可用及性能方向的优化—合并策略

ES底层Lucene基于LSM Tree的数据文件

Tiered(Sized) Merge Policy



原生合并策略：

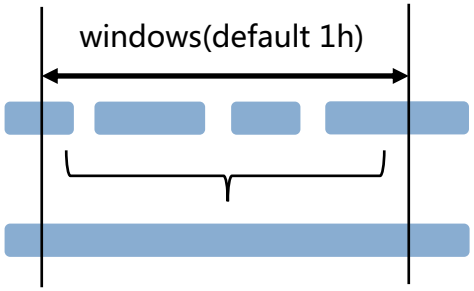
- 按文件大小相似性合并
- 默认一次固定合并10个文件

优点：合并高效，快速降低文件数

不足：数据不连续，文件裁剪能力弱

业内典型的合并策略：

Time-Window Merge Policy

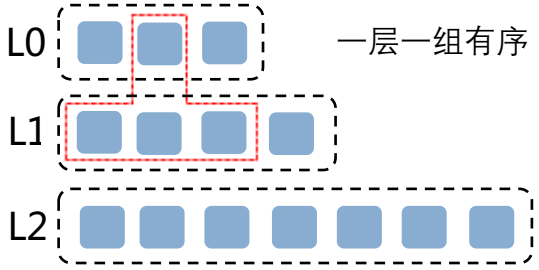


典型产品： Cassandra、HBase

优点：数据按时间序合并，支持表内TTL

不足：只适合时序场景；文件大小可能不一致，影响合并效率

Leveled Merge Policy

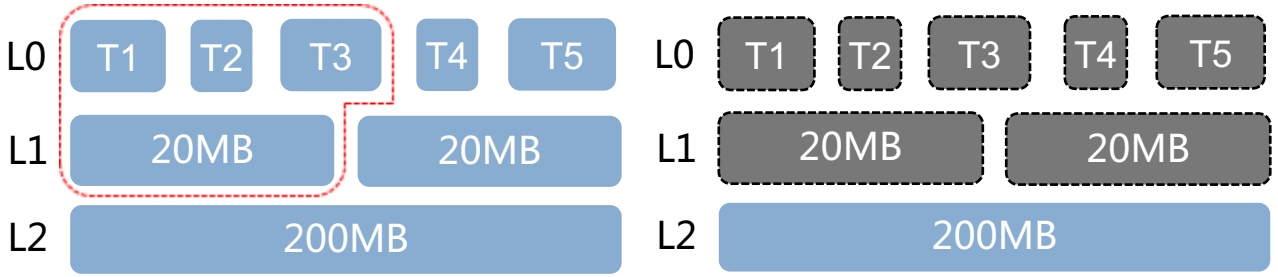


典型产品：LevelDB，RocksDB

优点：查询高效

不足：写放大较严重，影响TPS

优化方案：提升数据连续性、收敛文件数量，提升文件裁剪能力



数据连续性：按时间序分层合并

- **文件之间按创建时间排序**
- 除L0层外**每层按目标文件大小合并**，不固定单次合并文件数量
- **不足：**因大小不足有少量未合并文件

文件收敛：冷分片持续合并

- **持续合并**超过5min不写入的分片
- 控制合并并发
- 控制合并范围，过冷数据不再合并

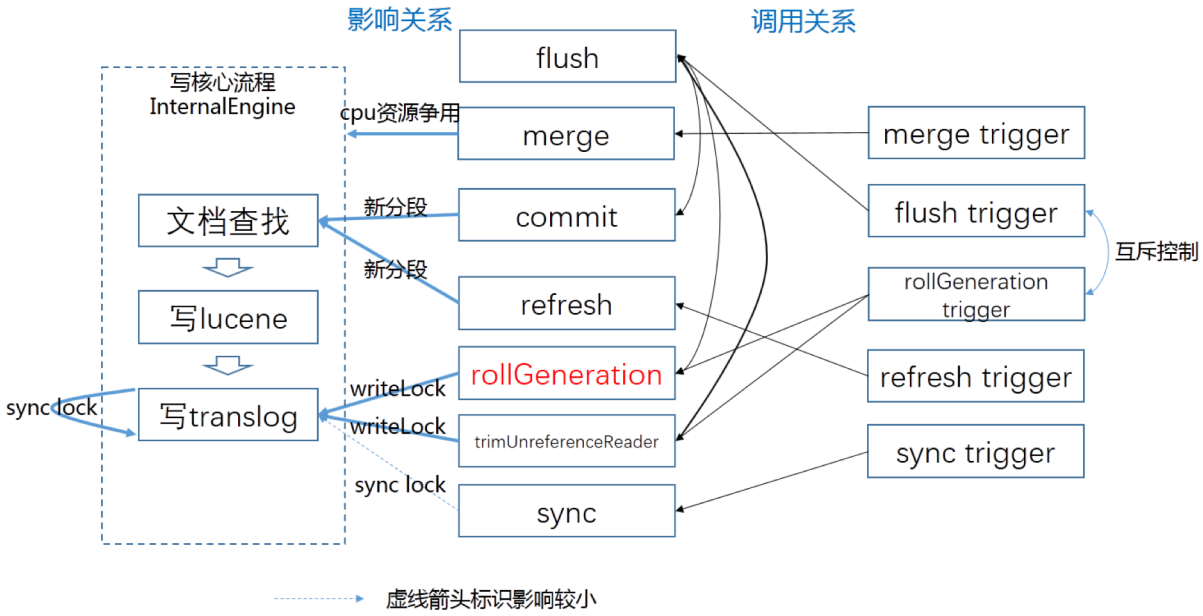
优化效果：搜索场景查询性能提升 40%

二、 高可用及性能方向的优化—大吞吐写入

场景分析

腾讯某金融业务，画像分析系统基于ES构建，14亿+数据需要快速的导入，导入任务每天夜间执行一次
写入性能不高，CPU使用率不高。通过lucene基准压测发现ES写入性能相比lucene直接写入相差50%左右

ES内部的各种操作对写入的影响

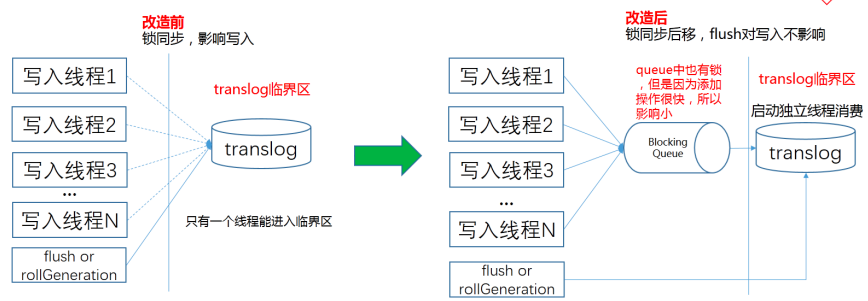


源码改造方案（使用BlockingQueue异步写）

已废弃

思路
使用内存队列，解耦写入操作（原来是同步写synchronized）和其他translog的磁盘刷新操作（包括文件滚动）

方案1



实现思路
通过包装器，除add方法外其他方法代理给原translog，add方法通过queue进行写入

方案2

```
public void rollGeneration() throws IOException {
    // make sure we move most of the data to disk outside of the writeLock
    // in order to reduce the time the lock is held since it's blocking all threads
    sync();
    try (Releasable ignored = writeLock.acquire()) {
        try {
```

二、 高可用及性能方向的优化—性能优化总结

腾讯云 | 云社区 intel

系统性性能优化

缓存策略	开销/ 命中率	GC 策略
优化器	CBO	RBO
执行引擎	查询剪枝	IO 重排
存储引擎	合并策略	文件裁剪

优化效果

- 查询毛刺 **750ms ->50ms**
- FST OffHeap, Tencent Kona JDK

- 点查询加范围查询性能提升**一倍**
- 内置通用的基于规则优化的模板

- 带排序场景聚合性能提升**3-7倍**
- 顺序抓取查询性能**10%**

- 搜索场景查询性能提升**40%**
- 带主键的写入性能提升**一倍**

性能优化相关的PR：

聚合查询优化（ES 7.6.0）

https://github.com/elastic/elasticsearch/pull/48399

Translog刷新优化（ES 7.4.0）

https://github.com/elastic/elasticsearch/pull/45765

https://github.com/elastic/elasticsearch/pull/47790

文件裁剪优化（Lucene 8.3.0)

https://github.com/apache/lucene-solr/pull/884

缓存策略优化 (Lucene 8.4.0)

https://github.com/apache/lucene-solr/pull/940

目录

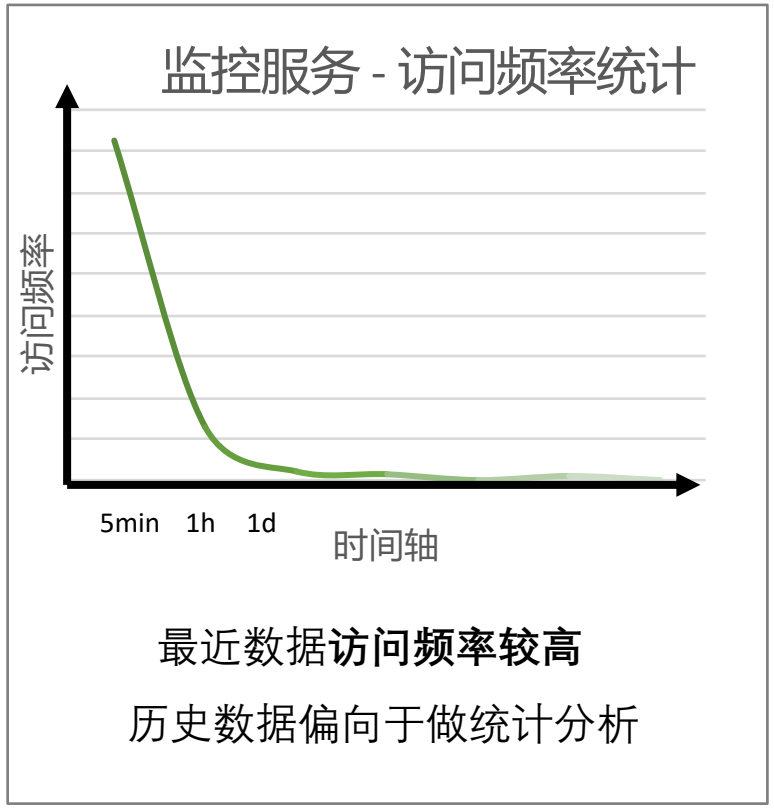
三、 低成本解决方案上的优化—存储成本

场景分析

整个腾讯云监控基于ES，单集群平均写入速度1000w/s，业务至少保留近半年数据可供查询。
1000w(QPS) × 86400(秒) × 180(天) × 50byte(avg doc size) × 2(replicas) ≈ 14 PB ≈ **1500台物理机**
远超出业务成本预算，如何在满足业务需求同时降低成本？

优化方案

调研业务数据访问频率



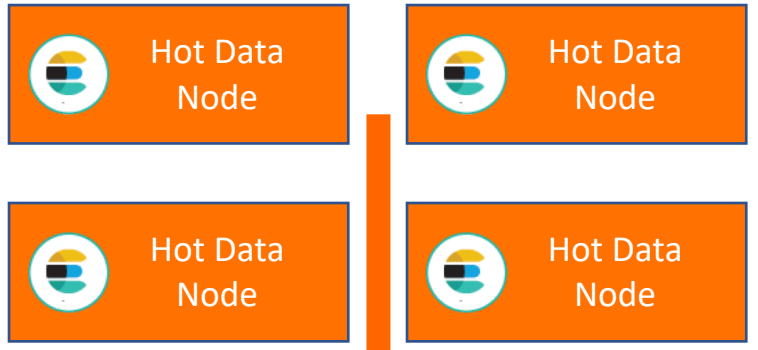
最近数据**访问频率较高**

历史数据偏向于做统计分析

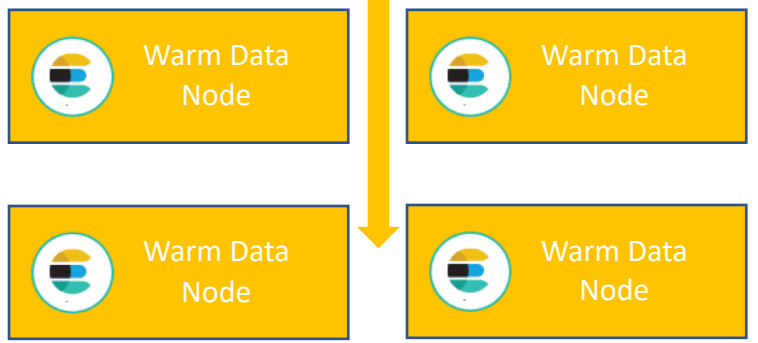
存储成本解决方案

- 冷热分离
- 冷备(COS)
- 生命周期管理
- 多种机型
- 多盘策略

高配置高IO机型



低配置低成本机型



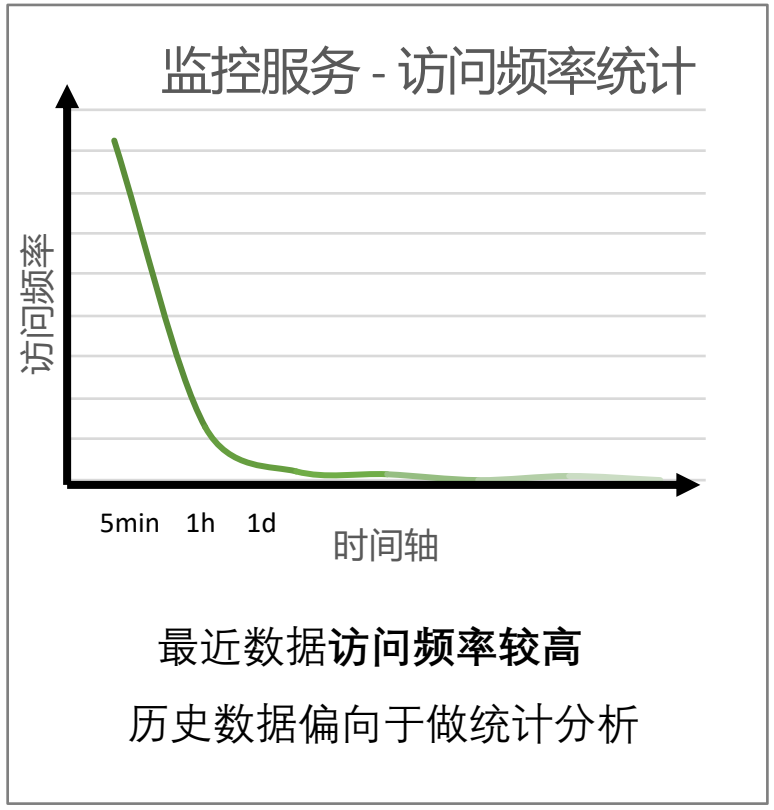
三、 低成本解决方案上的优化—存储成本

场景分析

整个腾讯云监控基于ES，单集群平均写入速度1000w/s，业务至少保留近半年数据可供查询。
1000w(QPS) × 86400(秒) × 180(天) × 50byte(avg doc size) × 2(replicas) ≈ 14 PB ≈ **1500台物理机**
远超出业务成本预算，如何在满足业务需求同时降低成本？

优化方案

调研业务数据访问频率

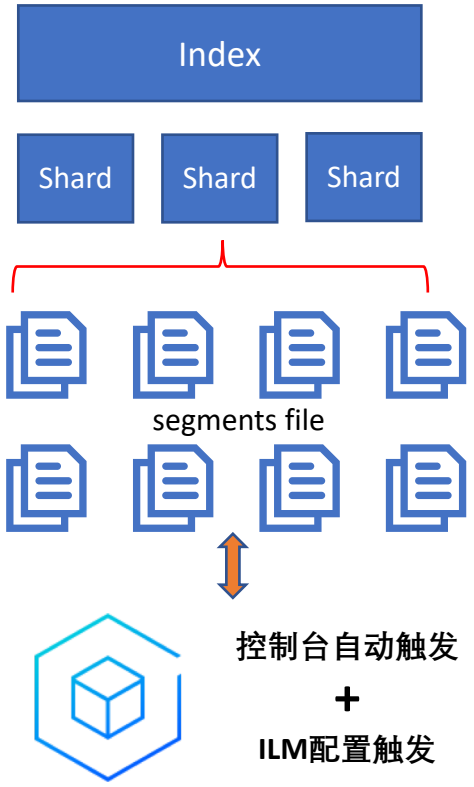


最近数据**访问频率较高**

历史数据偏向于做统计分析

存储成本解决方案

- 冷热分离
- 冷备(COS)
- 生命周期管理
- 多种机型
- 多盘策略



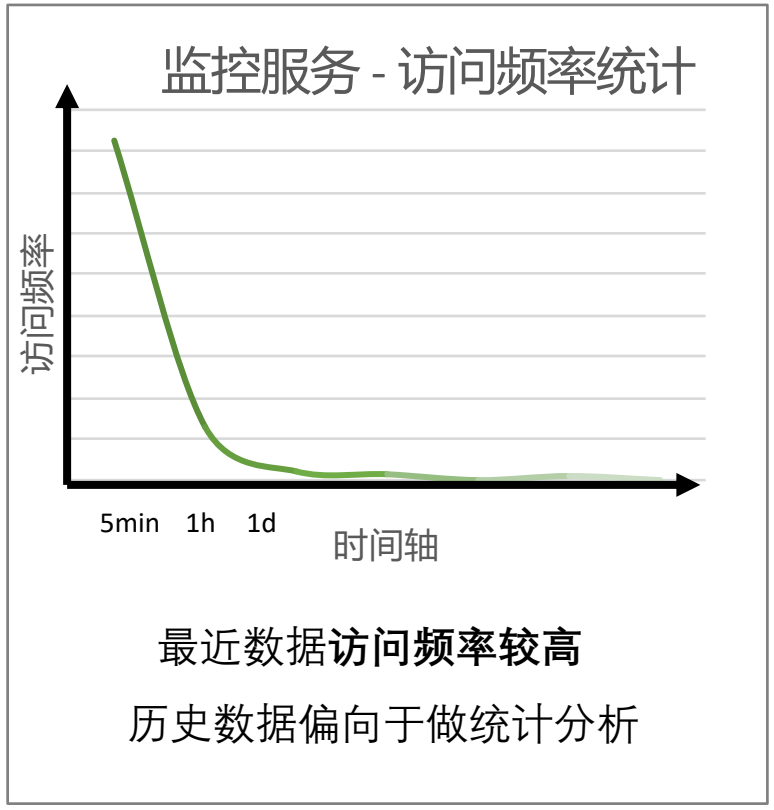
三、 低成本解决方案上的优化—存储成本

场景分析

整个腾讯云监控基于ES，单集群平均写入速度1000w/s，业务至少保留近半年数据可供查询。
1000w(QPS) × 86400(秒) × 180(天) × 50byte(avg doc size) × 2(replicas) ≈ 14 PB ≈ **1500台物理机**
远超出业务成本预算，如何在满足业务需求同时降低成本？

优化方案

调研业务数据访问频率



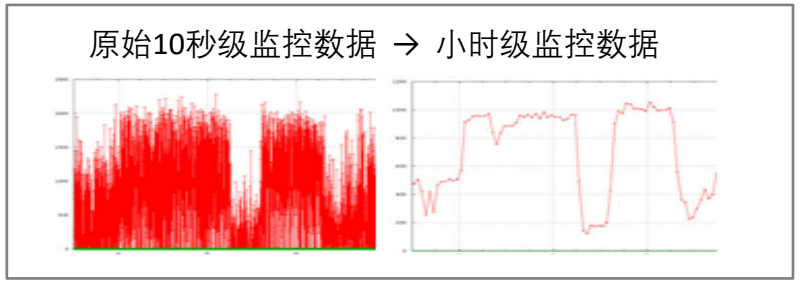
最近数据**访问频率较高**

历史数据偏向于做统计分析

存储成本解决方案

- 冷热分离
- 冷备(COS)
- 生命周期管理
- 多种机型
- 多盘策略

Rollup – 降精度



ILM – 索引生命周期管理

- **Hot Phrase**
  - Rollover
- **Warm Phrase**
  - Shrink
  - Frozen Indice
  - Rollup
- **Cold Phrase**
  - Close Indice
  - SnapShot
  - Delete Indice

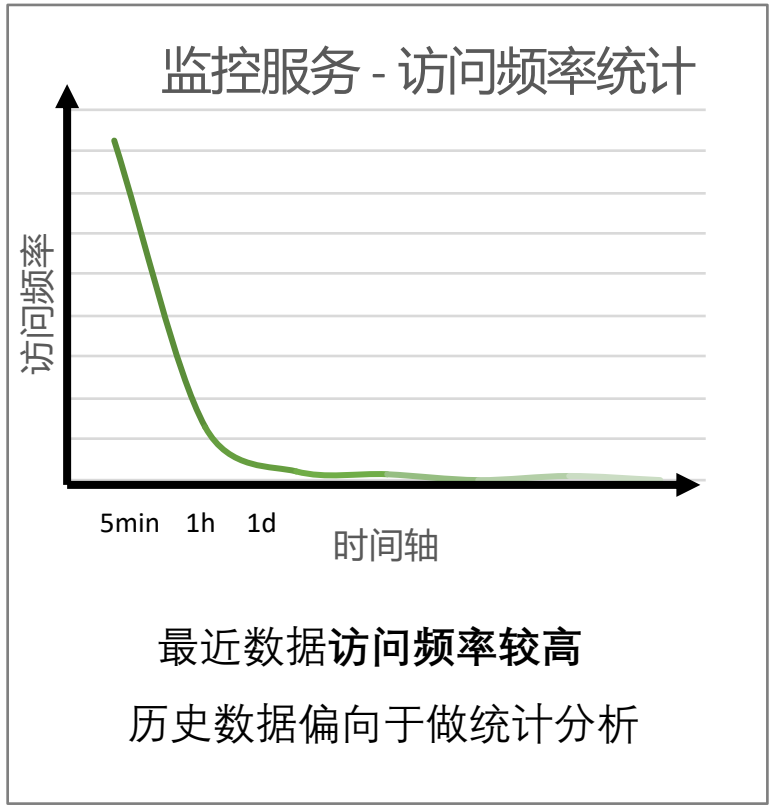
三、 低成本解决方案上的优化—存储成本

场景分析

整个腾讯云监控基于ES，单集群平均写入速度1000w/s，业务至少保留近半年数据可供查询。
1000w(QPS) × 86400(秒) × 180(天) × 50byte(avg doc size) × 2(replicas) ≈ 14 PB ≈ **1500台物理机**
远超出业务成本预算，如何在满足业务需求同时降低成本？

优化方案

调研业务数据访问频率



最近数据**访问频率较高**

历史数据偏向于做统计分析

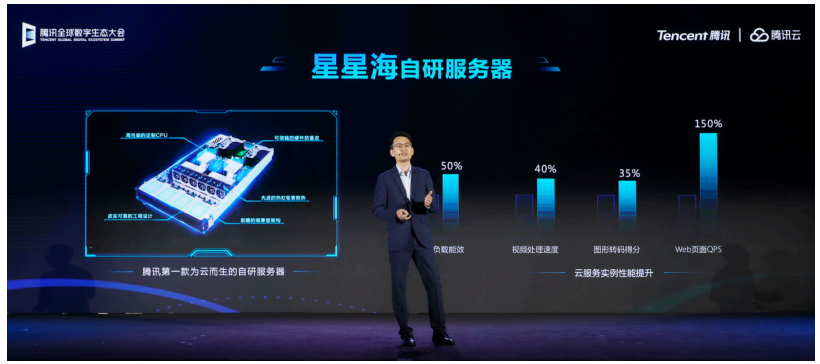
存储成本解决方案

- 冷热分离
- 冷备(COS)
- 生命周期管理
- **多种机型**
- 多盘策略

多种机型满足不同场景

标准型 1:4/1:2	内存型 1:8	本地盘机型
日志、监控等场景	搜索、数据分析等场景	海量数据存储场景

腾讯自研星星海SA2机型
综合性能提升**35%**以上，成本降低**30%**



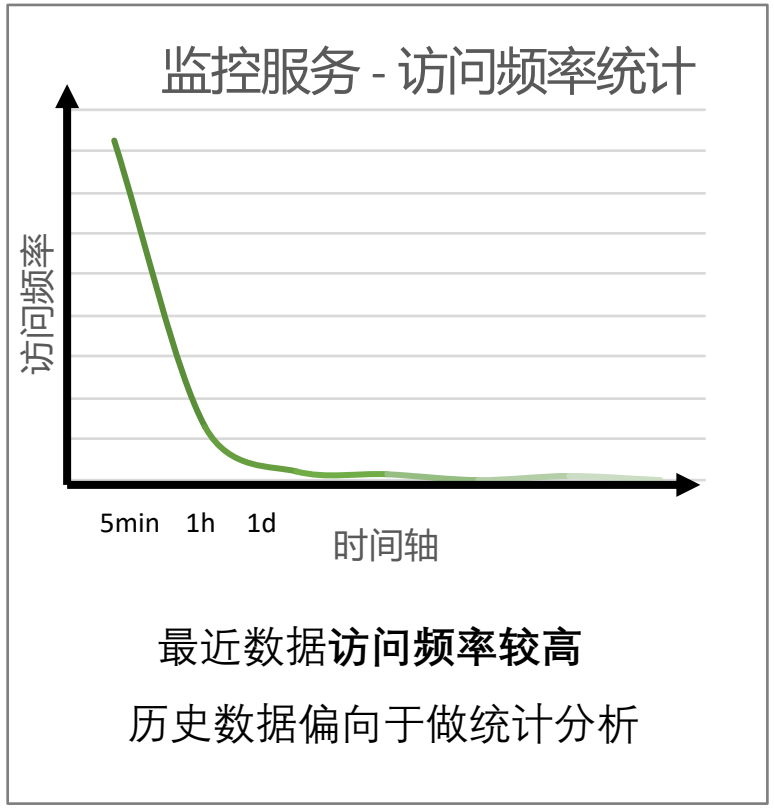
三、 低成本解决方案上的优化—存储成本

场景分析

整个腾讯云监控基于ES，单集群平均写入速度1000w/s，业务至少保留近半年数据可供查询。
1000w(QPS) × 86400(秒) × 180(天) × 50byte(avg doc size) × 2(replicas) ≈ 14 PB ≈ **1500台物理机**
远超出业务成本预算，如何在满足业务需求同时降低成本？

优化方案

调研业务数据访问频率



最近数据**访问频率较高**

历史数据偏向于做统计分析

存储成本解决方案

- 冷热分离
- 冷备(COS)
- 生命周期管理
- 多种机型
- 多盘策略

单盘IO打满

突破CBS单盘IO能力限制
（260MB/S SSD） （150MB/S 高性能）
最大利用本地盘机型能力

提升单节点IO能力
减少节点数降低成本
减少分片数避免写放大

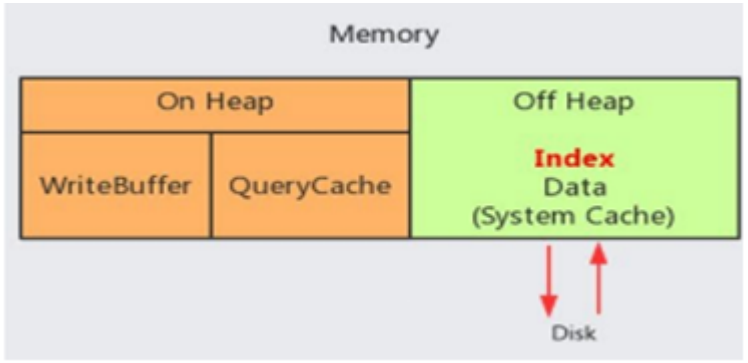
三、 低成本解决方案上的优化—计算成本

场景分析

某风控公司多维分析系统，堆内存使用率长期在80%以上
业务数据量大且Text字段占比较多，通常需要在全量数据中进行多维分析，为了避免节点OOM，降低单节点的数据量，需要提升一倍的节点数量

某日志集群	磁盘总量	磁盘使用率	内存总量	堆内存总量	堆内存使用率	FST内存
name	disk.total	disk.used_percent	ram.max	heap.max	heap.percent	fst
node-1	42.9tb	33.05	62.2gb	31.8gb	72	16.8gb
node-2	42.9tb	32.80	62.5gb	31.8gb	67	16.8gb
node-3	35.8tb	38.64	62.2gb	31.8gb	74	16.6gb

原生方案



原理：依赖系统缓存，**按需加载文件**

优点：实现简单

不足：淘汰策略由系统控制，海量数据场景性能N倍损耗

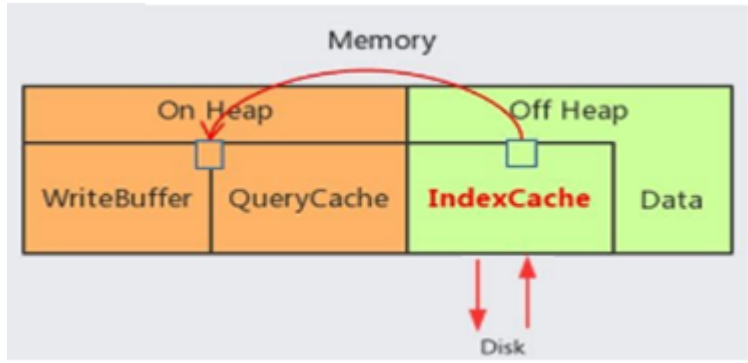
线上集群分析：

- 堆内存使用率很高
- 而磁盘使用率却比较低

堆内存使用率为什么会高？

- FST 占用绝大部分堆内存：50% ~ 70%
- FST 与数据量成正比：10TB 磁盘 ： 10GB ~ 15GB FST

优化方案



原理：借助堆外内存，释放堆内存

优点：Cache保障FST空间，精准淘汰策略提高内存使用率，多级Cache提高性能

不足：堆外内存管理开发成本高

目录

四、 未来展望

可用性：强化智能诊断

强化节点自愈自运维功能

性能：扩展多维分析能力

成本：存储与计算分离(coming soon)

可搜索的备份(searchable snapshot)

功能：完善数据集成通道

强化大数据平台融合

腾讯云 | 云社区 intel

互 动 问 答

简历请发
ethanbzhang@tencent.com

关注［云加社区］
回复［线上沙龙］获取PPT

扫码免费体验Elasticsearch
领取折扣特惠