Elastic 中文社区 Meetup 武汉站

腾讯 Elasticsearch 增强版内核演进之路

黄　华
Elasticsearch Top 100 contributor
腾讯 Elasticsearch 内核负责人

目录

腾讯 ES 发展历程

腾讯 ES 发展历程

内部云建设

- 时序场景应用（腾讯云监控）
- 可用性：区间平滑限流
- 成本：Rollup (成本数量级下降)

公有云CES发布

- 内部云单集群 PB 级 500+节点
- 托管平台建设（公有云）
- 内核 Bug 持续修复
  - 分布式锁
  - 大量重复任务压垮 master
  - 大规模集群长时间无法选主
  - 节点反复加入、离开
  - ……

Elastic商业合作

- X-Pack 商业套件
- 专有云适配
- 可用性：Breaker 增强，全链路限流
- 性能：
  - 普通写入提升 20%+
  - 带主键写入提升 50%+
  - 聚合查询提升 7倍+
  - 查询毛刺降低 10倍+

规模化运营

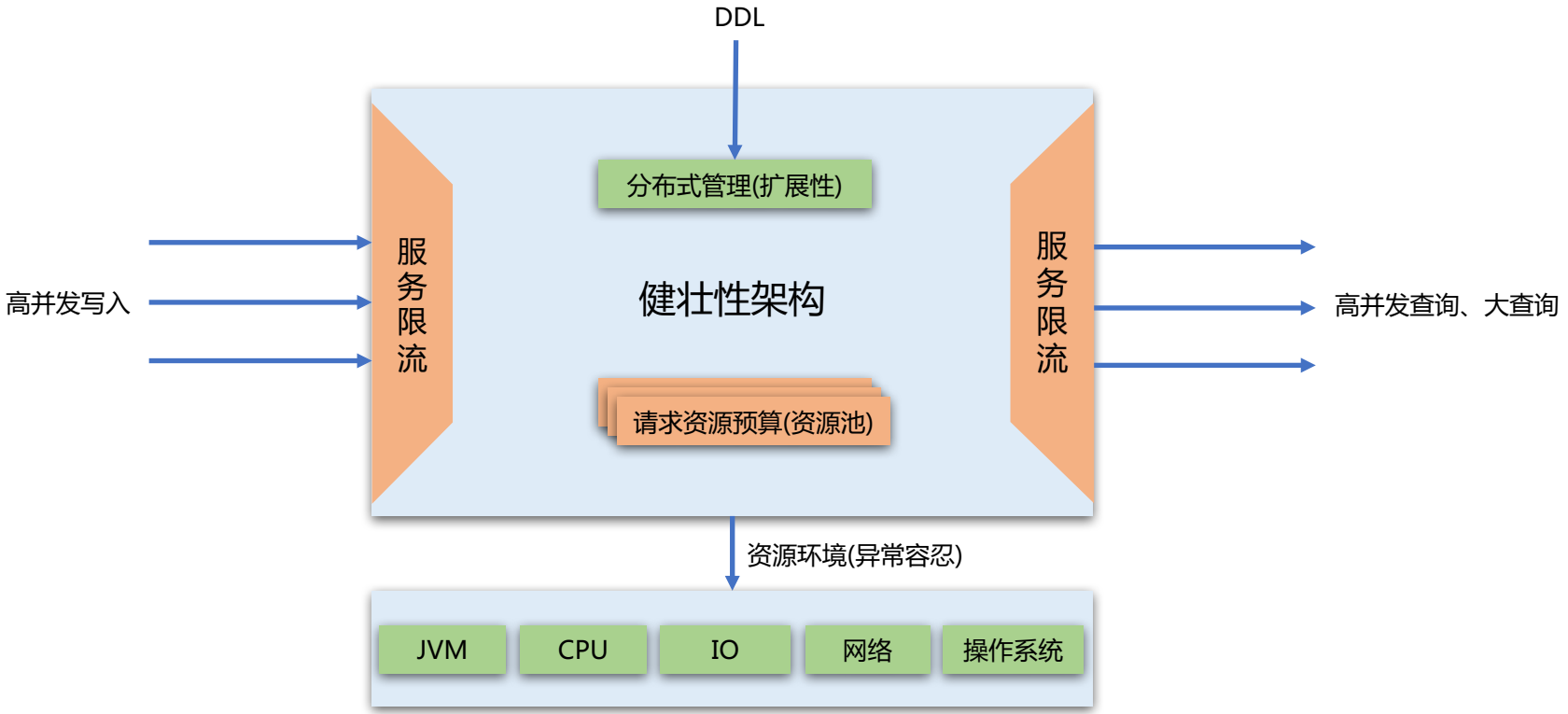
- 百 PB 规模
- 内部增速 300%+
- 外部增速 500%+
- 扩展性：千级节点、百万级分片
- 成本：高效压缩算法、编码优化

扩生态，强内核

- 完善 Elastic Stack 产品生态
- 闭环 PB 级数据处理方案
- 智能诊断
- 可用性：均衡算法、多级分区
- 成本：
  - Searchable Snapshot（7.14）
  - Hybrid Store

腾讯 ES 增强版内核演进

可用性 — 健壮性架构



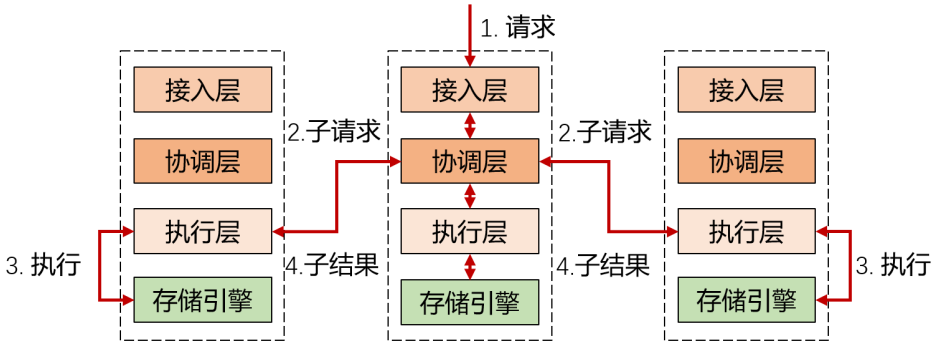
理想的健壮性架构：高压力固若金汤（服务限流）+ 不可靠环境提供可靠服务（异常容忍）+ 分布式线性扩展（扩展性）

ES 可用性问题分析

主要问题

- 大查询：扫描、聚合压垮节点内存
- 高并发点查询：并发查询毛刺触发拒绝
- 高并发批量写：超高写入并发易抖动、雪崩
- 硬件异常：导致集群抖动、雪崩

ES 读写模型



优化思路

- ES 官方：漏斗限流，功能不完善
- 腾讯 ES：全链路限流 + 异常容忍 + 异常恢复

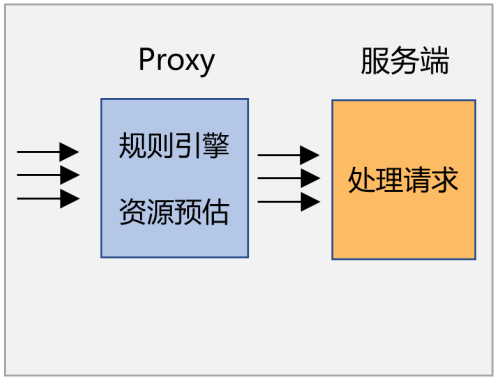
资源分析

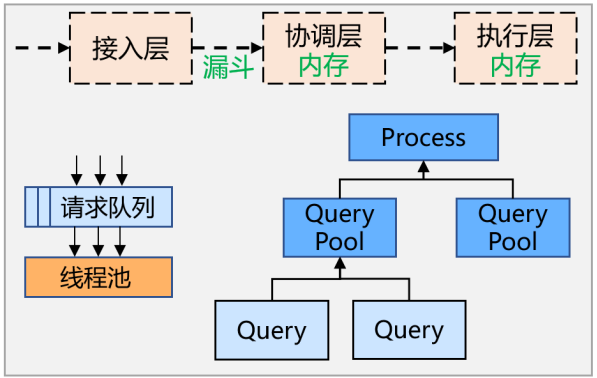
瓶颈：**内存** > **计算** > **硬盘** > 网络

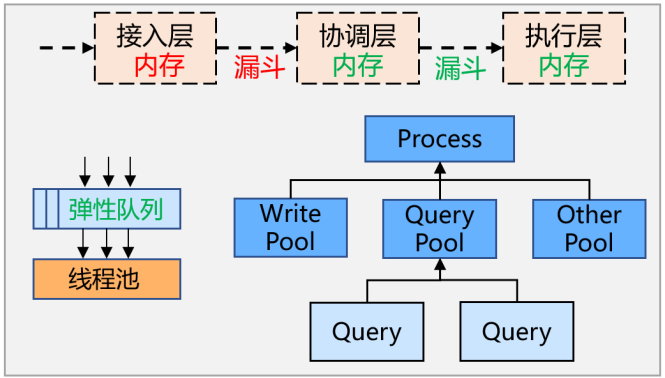
请求类别	计算	内存	硬盘	网络
大查询	***	*****	***	*
高并发点查询	****	***	****	*
高并发批量写	****	****	*	*

可用性优化 – 服务限流方案

方案





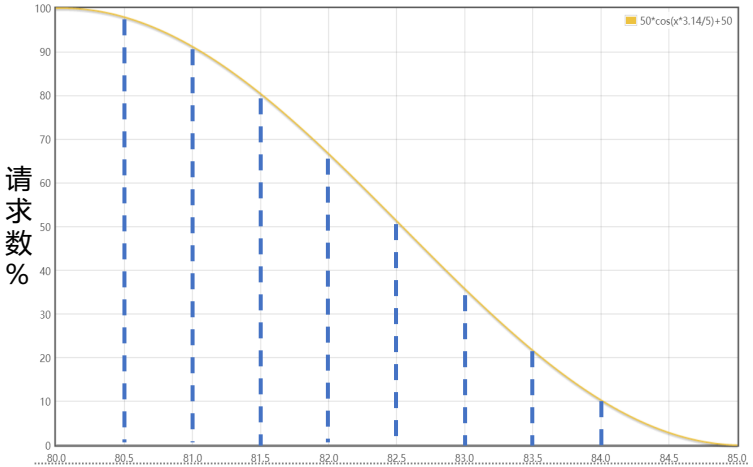


	Proxy 限流	漏斗限流 + 分层内存熔断（Impala）	弹性漏斗 + 全链路内存熔断（腾讯ES）
优点	灵活，独立部署，变更易	综合考虑CPU、IO、内存矛盾	综合考虑CPU、IO、内存矛盾，系统性全链路
适用	微服务：高并发、轻量请求	交互式查询：低频、大查询	混合场景：高并发读写、大查询
缺点	• 无法准确预估资源消耗 • 场景适应能力弱，运维成本高	• 大查询：仅统计关键路径 • 高并发读写：易拒绝、甚至崩溃	• 小规格实例极端情况难覆盖 异常快速恢复（GC负债管理）

可用性优化 – 服务限流方案

主要挑战：

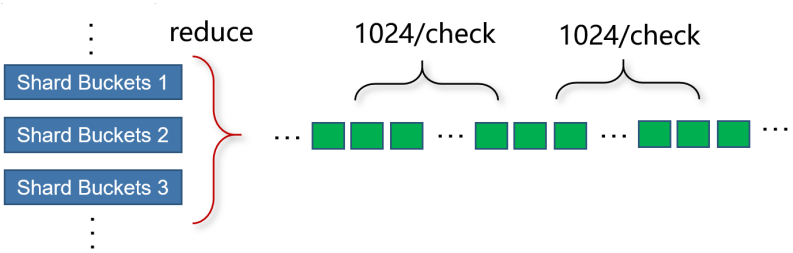
- 服务限流，如何保持高吞吐？
- 大查询，如何精准熔断？



内存限流区间

解决方案：

- 区间平滑限流，防抖动
- 全链路内存统计 + 步长限流 + 细粒度超时检测



PR 已合并

https://github.com/elastic/elasticsearch/pull/46751

官方作为亮点功能

Highlights

Better support for large aggregations

Ever run into an Elasticsearch error about a `too_many_buckets_exception`? Igor made some recent changes which should significantly reduce the odds of users running

Back in the wild west days of Elasticsearch, ~~feeds~~ aggregations could consume as much memory as they wanted and routinely caused nodes to crash by running out of memor leading to crashing. So we implemented `search.max_buckets`, a simple threshold that halts execution if we collect more than 10k buckets (by default).

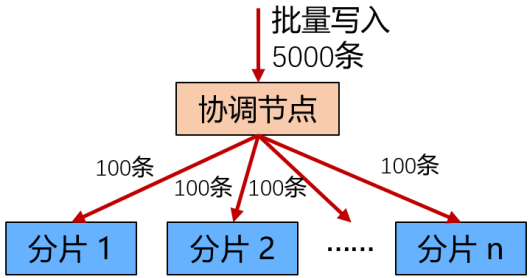
This largely fixed nodes crashing, but also has the side-effect of rejecting many reasonable aggregations. It left Kibana and other "middleware" in a sticky position - neither Kiba the setting counted buckets as they are collected internally, not as they are exposed to the user. This is counterintuitive to a user; a query asking for only 100 results could fail b experience - especially with Kibana needing to deliver the bad news.

In 7.7, we merged a community contribution which leveraged the real-memory circuit breaker. This means that our accounting of memory usage is more accurate and that the m the `search.max_buckets` parameter to 65,535 and also changes the semantics of the setting so that buckets are only counted after the final reduction is done. This makes users writing large aggregations is out of the way.

可用性优化 – 异常容忍

问题

- 百万级 TPS，写入拒绝率高
- 资源利用率低

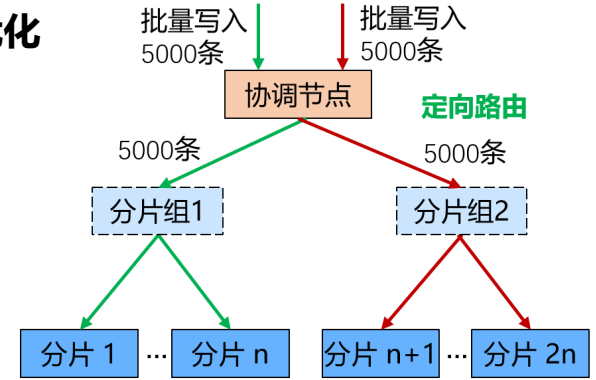


分布式写入，高扇出，长尾放大影响

原因

- 长尾节点：GC、慢盘、网络抖动
- 分布式写入：高扇出放大影响，写入堆积

优化



分组定向路由，低扇出，容忍慢节点

	写入吞吐(TPS)	CPU	拒绝率
开源版本	51w	31%	3‰
定向路由	113w(+121%)	49%(+58%)	0‰

可用性优化 – 集群扩展性

问题：PB 级日志业务按时间分区（多分片）

- Master 堵塞：元数据变更慢（10秒级）
- 写入拒绝：写入触发建表，请求等待堆积
- 资源碎片化：大量中小集群，运维成本高

原因：

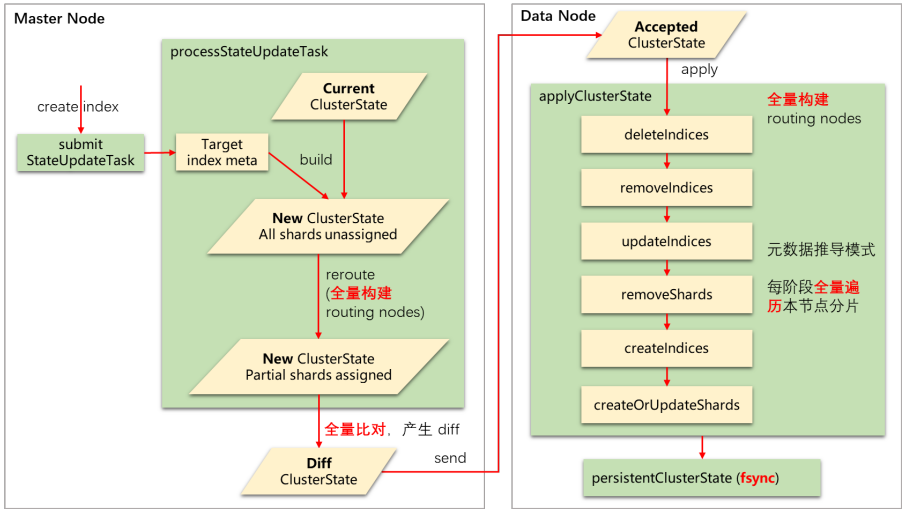
- 元数据变更推导模式管理集群
- 两阶段提交，全量构建、全量遍历、同步落盘

扩展性瓶颈：

- 元数据：分片数达到 3w，变更需15s；
- 节点数不超过 100
- 官方策略：单节点分片数限制（ <1k ）

*提交优化官方回复：*On a more reasonably-sized cluster of 30k shards the savings are tiny. Therefore I am closing this.

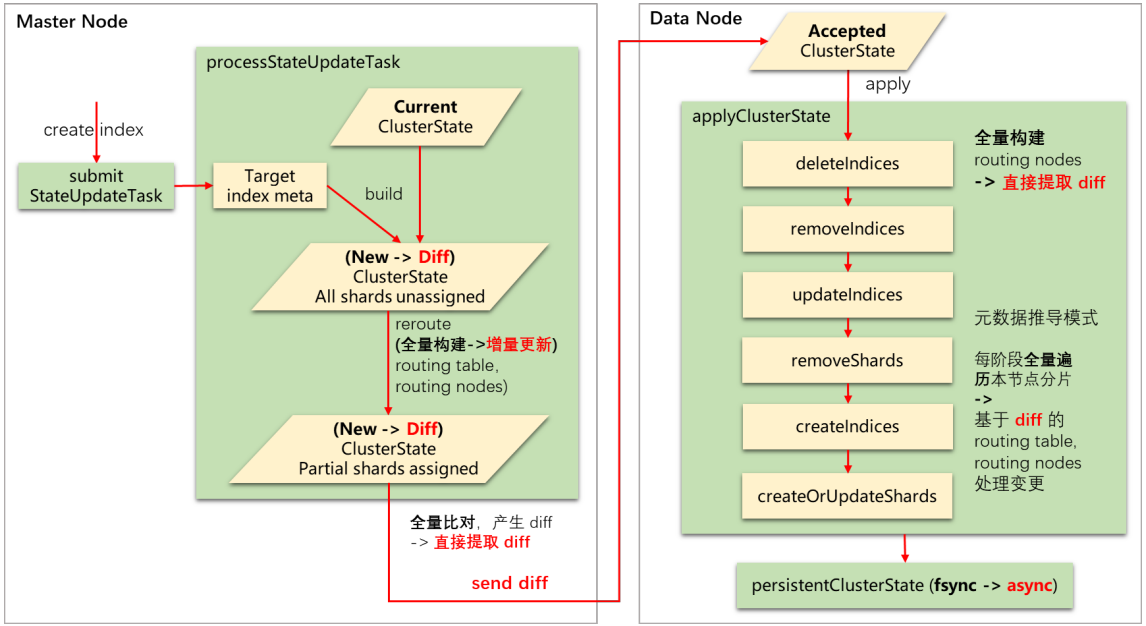
以索引创建为例，元数据变更流程：

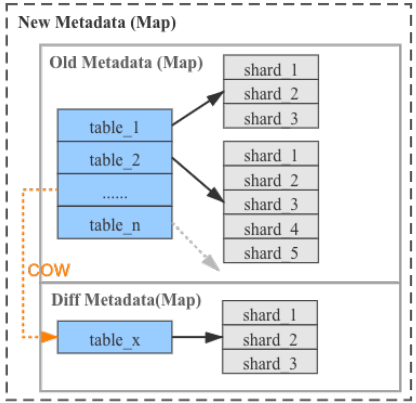


可用性优化 – 集群扩展性优化

方案：全链路增量、Diff 化

- 元数据核心：分片-节点 和 节点–分片 的映射
- 多版本控制：增量 Diff + COW





	开源ES	腾讯ES	优化效果
分片数	3w	100w	X 33
节点数	100	1000	X 10
DDL耗时	15s	500ms	X 30
DQL耗时	5s	200ms	X 25

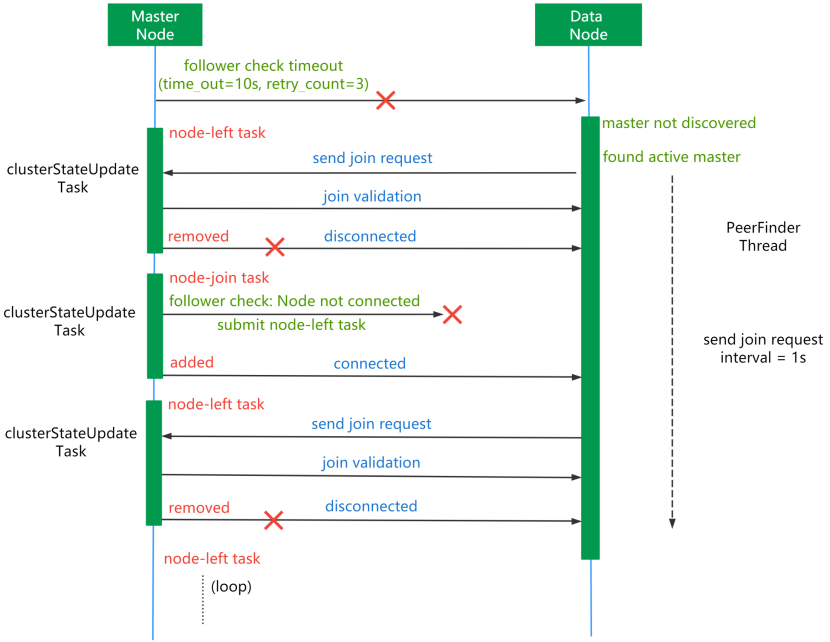
Patch：

ES-64753 ES-46520 ES-65045 ES-56870

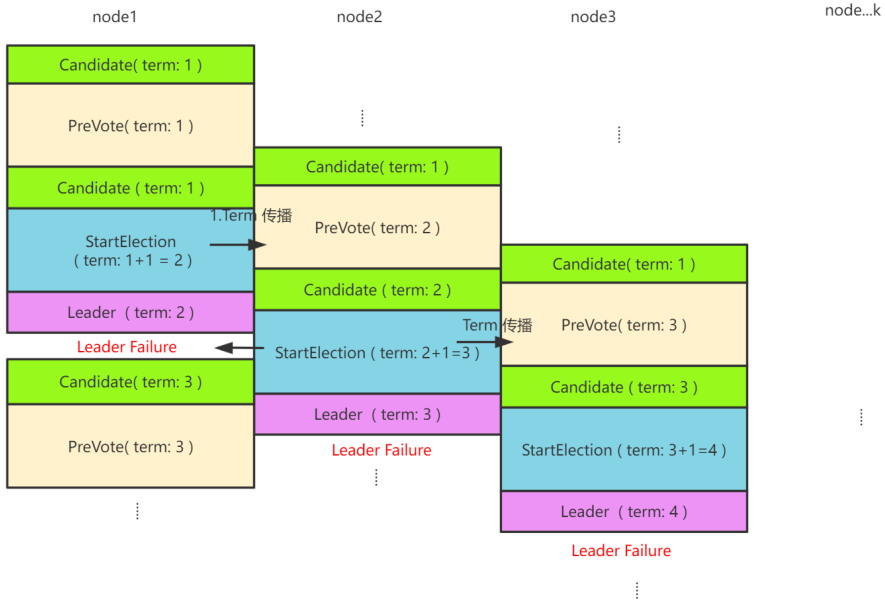
ES-59044 ES-65172 ES-60564 ES-59204

可用性优化 – 内核Bug持续修复

节点脱离后反复 left/join 长时间无法成功加入集群



集群全量重启后长时间无法成功选主



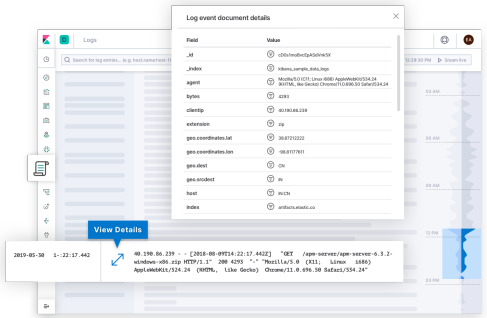
https://github.com/elastic/elasticsearch/issues/67873

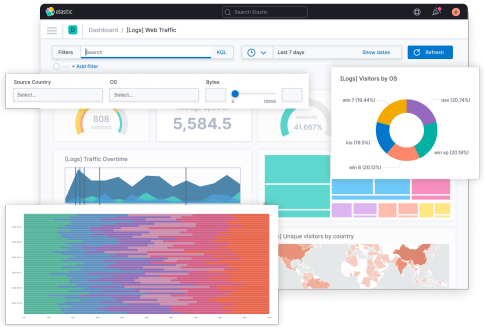
可用性优化总结

PaaS 平台可用性：99.99%

性能优化

场景分析





写入

- 代表场景：日志、监控等海量时序数据
- 高并发：每秒千万级吞吐，
  带id写入性能衰减1倍

查询

- 搜索服务：高并发、低延时，十万级 QPS，
  平响20ms，P95<100ms
- 聚合分析：大查询，内存、CPU开销高

性能优化

接入层

缓存策略 | 序列化 | 鉴权优化 | 字段裁剪

- 查询毛刺 750ms ->50ms Lucene-940
- 序列化裁剪查询性能提升10%
- 鉴权优化读写性能提升30%

执行层

查询剪枝 | IO 重排 | 对冲查询 | CBO/RBO

- 带排序场景聚合性能提升3-7倍 ES-48399 ES-57273
- 顺序抓取查询性能10%
- Translog锁优化写入性能提升20% ES-45765 ES-47790

存储层

文件裁剪 | 合并策略 | 压缩算法 | 编码优化

- 搜索场景查询性能提升40% Lucene-884
- 带主键的写入性能提升一倍 Lucene-9663

性能优化 – 执行引擎

场景分析

Composite Aggregation 6.5 GA

```
GET monitor_index/_search
{
  "size": 0,
  "query": {
    "bool": {
      "filter": [ {
        "range": {
          "timestamp": {
            "gte": "2019-01-01 00:00:00",
            "lte": "2019-01-06 00:00:00",
            "format": "yyyy-MM-dd HH:mm:ss",
            "time_zone": "+08:00"
          }
        }
      }]
    }
  },
  "aggs": {
    "NAME": {
      "composite": {
        "size": 100,
        "sources": [
          {"ip": {"terms": {"field": "ip"  } }},
          {"name": {"terms": {"field": "name"  } }},
          {"device": {"terms": {"field": "device"  } }}
        ]
      }
    }
  }
}
```

原生方案

固定长度的堆，不断迭代构建聚合结果

根据 field1、field2聚合：

流式聚合 Size: 3

doc	field1	field2
(1)	D	23
(2)	E	23
(3)	C	46
(4)	A	46
(5)	C	65
(6)	B	23
(7)	F	75

每次翻页，全量遍历

Stream Response key1

field1	field2
A	46
B	23
C	46

结束标志 AfterKey

Stream Response key2

field1	field2
C	65
D	23
E	23

Stream Response key3

field1	field2
F	75

性能优化 – 执行引擎

优化方案：Index Sorting + AfterKey 跳转 + Early Termination

```
{
  "monitor_index": {
    "settings": {
      "index": {
        "refresh_interval": "30s",
        "translog": {
          "sync_interval": "10s",
          "durability": "async"
        },
        "sort": {
          "field": [
            "ip",
            "name",
            "device"
          ],
          "order": [
            "asc",
            "asc",
            "asc"
          ]
        },
        "number_of_replicas": "1",
        "number_of_shards": "6"
      }
    }
  }
}
```

写入时基于group key进行排序

doc	field1	field2
(1)	A	46
(2)	B	23
(3)	C	46
(4)	C	65
(5)	D	23
(6)	E	23
(7)	F	75

1.提前结束

2.AfterKey跳转

3. 提前结束

Stream Response key1

field1	field2
A	46
B	23
C	46

结束标志 AfterKey

Stream Response key2

field1	field2
C	65
D	23
E	23

Stream Response key3

field1	field2
F	75

优化效果：聚合查询性能提升 3 – 7 倍（PR已合并）

https://github.com/elastic/elasticsearch/pull/48399 7.6.0
https://github.com/elastic/elasticsearch/pull/48130

成本优化

场景分析

搜索场景

计算	600核
内存	250GB
存储	100GB
网络	2Gb/s

少量数据、少量写入、高并发小查询

日志/时序场景

计算	12核
内存	600GB
存储	30TB
网络	2Gb/s

海量数据、高并发写入、少量大查询

优化方案

- 缓存策略
- 鉴权优化
- 序列化裁剪
- 字段裁剪
- 定向路由
- 新硬件
- 过期清理
- 备份归档
- 冷热分层
- 内存优化
- 数据上卷
- 压缩编码
- 高密度机型
- 多租户

腾讯云已支持 Searchable Snapshot

成本优化 — 内存优化

内存开销为什么这么高？

- 索引：常驻内存，与硬盘存储量呈正比
- 日志场景访问频率低：历史数据 75%+
- 案例：64G 内存支持 3T 存储 (ES痛点)

挑战：

- 提升内存利用效率
- 保持查询性能

性能极致优化：

- 零拷贝降低数据读写开销
- 两级 Cache 降低高频 Cache 重复查找开销

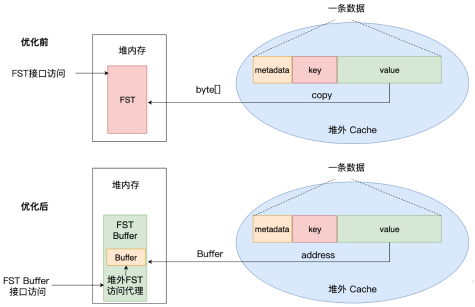
效果：

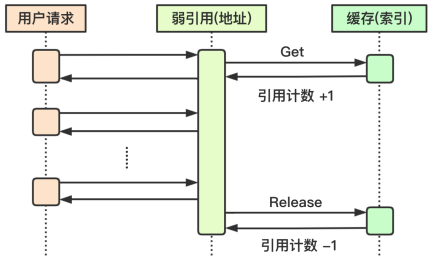
- 内存下降87%，读写性能基本持平
- 单机节点磁盘管理能提升10倍

方案

(Off-Heap)

	ES 官方	腾讯 ES 方案
原理	索引依赖系统缓存，按需加载	基于独立 Cache 进行缓存 精确控制淘汰策略、查找开销
优点	实现简单	独立可控，性能损失少
缺点	• 系统缓存不区分索引、数据 • 点查询性能有5+倍的抖动	





内存使用	Cache命中率	读QPS	写QPS	90%读时延	GC时间	单节点支持
8GB(-87%)	99%+	+0.02%	+1.58 %	-2.00%	-25%	30TB(x10)

成本优化 — 数据上卷（Rollup）

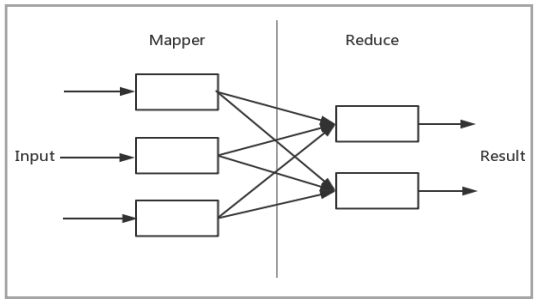
监控等时序场景：

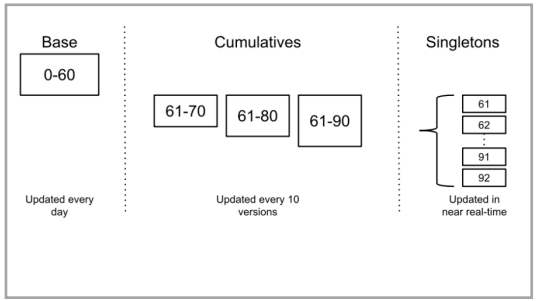
- 超长周期存储：半年+
- 用户诉求：成本低 & 访问快，如何兼顾？

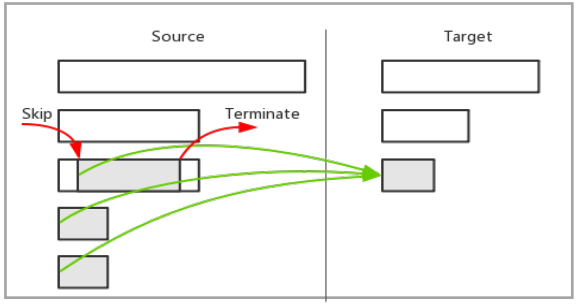
业界思路：

- 数据上卷：计算置换存储（细粒度数据预聚合）
- 分类：降维度、降精度(时间)

方案







	离线计算（Kylin）	Compact（Google Mesa）	异步流式引擎（腾讯 ES）
原理	基于Hadoop/Spark进行预计算	多表写入，Compact 多路归并排序数据	并发异步流式任务，原始数据多路归并
优点	实现简单	原始数据与上卷数据同步写，利于查询	无冗余表，计算高效，内存可控
缺点	计算开销高，依赖复杂	冗余写入，计算成本高	

效果：大幅优化存储成本、查询性能

数据粒度	数据量(天)	时间线性能	计算开销
分钟	61.7G	440ms	10% CPU
小时	1.1G（60倍~）	15ms(29~倍)	1%~ CPU

重点：

- 分片级并发：避免协调节点二次聚合
- 流式聚合：数据有序，高效裁剪

成本优化 — 压缩编码

存储挑战：

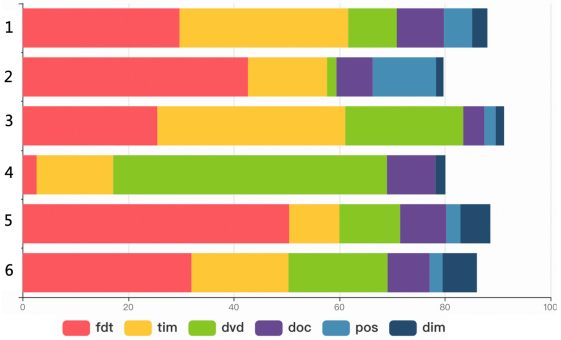
- 数据格式丰富：行存、列存、倒排索引等
- 压缩算法两极分化：LZ4 (快)、Deflate (压缩比高)

优化方案：

- 数据类型裁剪：场景模板调优
- 通用压缩算法：引入 Zstd 兼容性能、压缩比优势

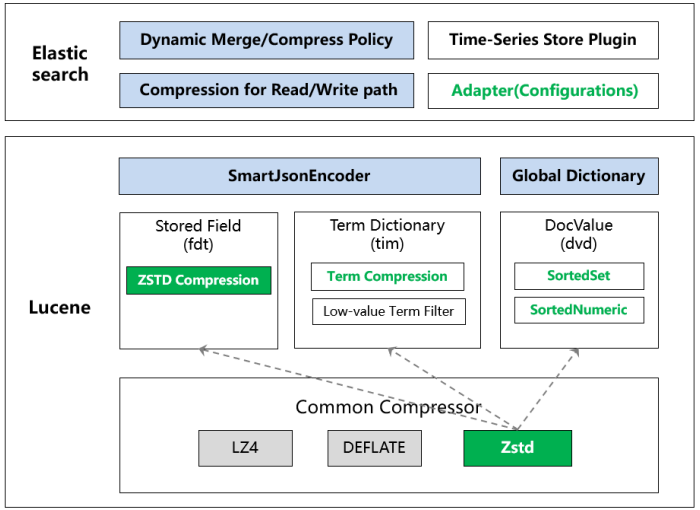
总体效果：行存压缩比提升 ~35%，其它类型 ~20%

算法	压缩比	性能(压缩速度)
LZ4	2.101	740 MB/s
Deflate	2.743	90 MB/s
Zstd	2.884	500 MB/s



文件类型：

- 行存文件(fdt)
- 列存文件(dvd)
- 字典文件(tim)
- ……



持续发展


业务场景
实时日志分析
全文检索
时序处理
生态打通
Elastic 生态
大数据生态
ES 内核
可用性
性能
成本
系统健壮性
数据容灾
缺陷修复
优化器
执行器
存储引擎
软件架构
硬件选型
场景调优
支撑系统
托管平台
监控平台
诊断平台
集群管理
作业管理
监控告警
审计日志
健康巡检
智能调优

开源贡献及未来规划

开源贡献

- 150+ PR
- 8 Contributors（两位全球 Top 100）
- 全球贡献最多第三方公司

未来规划

- 内核持续增强（HybridStore、Vectorization、Serverless）
- 生态打通（大数据）
- 场景拓展（交互式分析，闭环 PB 级数据处理方案）

Thanks

- 大规模火热招聘中 ……
- 腾讯 TEG 云架构平台部
  Elasticsearch、ClickHouse ……
- 专家、高级、应届生、实习生 ……
- **深圳、北京、武汉 ……**