**杭州恩牛网络**

**郑珠斌**

**2017年8月12日**



**[本文档是本人在ElasticSearch -v2.3.3 使用过程中一些经验&体会]**

ElasticSearch 白皮书

[ ElasticSearch 简介 3](#_Toc15315)

[ 概述 3](#_Toc7337)

[ ES框架 3](#_Toc24659)

[ ElasticSearch 使用场景 5](#_Toc28225)

[ 检索场景 5](#_Toc22373)

[ 统计分析场景 5](#_Toc18977)

[ 不适用场景 5](#_Toc29347)

[ ElasticSearch集群能力 7](#_Toc5486)

[ 最佳实践 7](#_Toc15561)

[ 业务保障 8](#_Toc10613)

[ 集群扩容 9](#_Toc17771)

[ 集群安全 9](#_Toc27914)

[ 性能优化 10](#_Toc24404)

[ 测试数据 12](#_Toc31218)

[ 测试环境 12](#_Toc3795)

[ 数据索引 12](#_Toc13937)

[ 数据检索 15](#_Toc3972)

[ 数据统计&分析 22](#_Toc19573)

[ 总结 23](#_Toc21849)

[ 使用范例 24](#_Toc29431)

[ 故障&运维 28](#_Toc19182)

[ 故障汇总 28](#_Toc25480)

[ 运维 29](#_Toc24425)

# ElasticSearch 简介

## 概述

ElasticSearch是Elastic基于lucense索引引擎创建的一个全文搜索服务项目 ，该项目有如下特点：

1. 开源项目
2. 近实时数据处理能力
3. 支持分布式水平扩展
4. 分布式集群高可用
5. 海量数据快速检索
6. 服务稳定、可靠

## ES框架

架构图



ES 架构图

1. 节点通过广播的方式在当前网段查找相同clusterId的节点（也可以主动指定目标集群ip信息）
2. 如果没有找到集群 ，从backup master中选举出1个master成立集群
3. master节点通过心跳方式，判断集群健康情况并进行保护
4. master 统一负责索引分片的分配，并向节点进行信息同步
5. master失联后，从backup master重新选取1个master节点
6. client node负责接收客户端应用的读/写请求交互
7. master / data node同时也是client node
8. 索引（index）以分片（shard）离散分布在各个data node
9. data node负责数据存储&计算，data node间根据集群信息进行数据同步

任务执行过程图



任务过程图

1. client app随机发送任务给随机的client node
2. client node根据任务信息拆解成多个子任务发送给响应data node
3. data 节点负责任务处理，并将结果返回给client node
4. client node 对data node 返回的结果进行合并，并返回给client app
5. client node 是client app 跟ES集群交互的1座桥梁
6. 从上图中可以看到整个任务的执行不经过master ，master 不会成为集群水平扩展的瓶颈

# ElasticSearch 使用场景

## 检索场景

ES作为一个索引搜索引擎，顾名思义主要优势还是提供基于索引的海量数据检索能力

1. 文档全文检索（如维基百科）

通过分词算法提取文档关键字，并对关键字创建索引

通过关键字查找文档

支持关键字“精确”、“模糊”及“多关键字”组合检索

1. 结构化数据检索（如账单明细）

随机组合查询维度

各查询维度支持“精确”、“模糊”匹配

## 统计分析场景

ES除了提供海量数据的检索能力外，也具备部分基于索引的海量数据统计分析能力

1. 数据统计

基于埋点日志明细，统计app页面的点击情况

基于账单明细，统计各个城市的消费能力

1. 推荐系统（如电商购物推荐）

通过商品分类&浏览记录统计分析出用户可能感兴趣的商品进行推荐

## 不适用场景

1. KV查询

ES从功能角度上来说可以当成KV数据库使用

相比Cassandra/HBase等明确KV定位的存储库来说没有优势

如果想要兼容“复杂查询”、“聚合统计”及“KV查询”，ES的使用效果并不好，请求容易受到“复杂查询”及“聚合统计”请求的影响，导致请求发生堵塞&延时，而“KV查询”往往需要”高并发低延时”

1. 复杂聚合统计

ES聚合统计完全依赖于内存，统计中间结果不落地

统计深度太深或者维度值基数过于庞大（如UserId）会导致请求失败或服务发生OOM异常导致集群瘫痪

1. 精确UV统计

ES 支持count（distinct $attribute） 语法

由于中间计算结果不落地 ，当计算资源（内存）超出限制时，计算结果是一个近似值（可能存在很大误差）

1. 海量数据捞取

查询结果在千万级别以上，并希望将结果集全部爬取出来做进一步分析，ES的性能并不理想。

具体性能可以参考关系型数据库（oracle/mysql）的游标性能

# ElasticSearch集群能力

## 最佳实践

1. master node、data node及client node 分开部署

ES集群的每个节点（实例）默认同时拥有master node、data node及client node角色，分开部署这三个角色可以降低集群故障概率。

master node单独部署可以防止集群发生脑裂

data node 负责数据的存储及计算，在集群中承受的压力最大，跟master node 部署在一起会影响到master node 稳定性（包括脑裂）

client node 可以认为是为data node 减轻压力，如承担了data node结果集的合并（可以大大降低data node 发生OOM概率）、接收数据写入请求并将数据整理好发送到对应的data node 上（如批写入）

1. 索引分片（shard）分布

分片均匀分片在各个data node

基于集群并发性能考虑，建议每个索引每个节点不要分配超过3个分片

大表根据“时间/其它维度”切成多个索引，并对检索条件进行优化

1. 索引分片大小

经测试&借鉴，每个分片的大小在20GB左右集群性价比比较高

创建索引的时候，可以默认数据&索引比为1：1，即是每个分片存储10GB数据+10GB索引

1. 单节点多实例

ES运行于jdk上，分配给每个实例的对内存不要超过32GB

为使得内存价值最大化 ，可以在同一个节点上部署多个实例

所有实例的堆内存合计值不要超过节点总内存的一半，另一半内存预留给操作系统及ES堆外内存使用，提高聚合统计性能

建议每个实例分配10~32个Vcore

1. 内存&磁盘空间比

经借鉴（百度测试结论），内存&磁盘空间比在1:32集群有比较高的性价比

具体可根据业务的实际需求做适当调整

1. 负载均衡

合理分配索引分片，尽可能使其离散到集群的各个节点上

避免分片堆积在个别节点上，发挥不出集群分布式处理能力

分片堆积个别节点上，影响的不仅仅是当前索引还会影响到其它索引性能（如造成了服务器负荷过大 ）

1. GC 回收策略

经过实践，选择G1 GC策略，在集群稳定性上要优于CMS GC策略

1. 慎用分词

目前ES支持“ik”、“ansj”、“standard”等分词算法

可以对词典进行自定义配置

在ES是一个大池子的时候，词典的定义必定无法一一满足

ES数据类型支持List列表，建议对文本切好以后以列表形式写入ES

1. “\_all” 关键字

“\_all” 使用起来看似便利，实际上极度消耗资源，禁止使用

1. 结合HBase等KV存储使用

在架构上可以让ES只存储索引并将文档内容存储在KV存储，让ES专注于做自己擅长的事

## 业务保障

结合“ES 最佳实践”，聚焦于“ES业务保障”，可以从如下几点入手：

1. 数据安全保障

结合数据属性&业务属性（离线&线上），考虑是否对索引创建副本及副本数

0副本情况下，节点故障索引不可读不可写，如果磁盘故障还可能造成数据丢失不可恢复

1副本情况下，节点故障索引可读不可写，数据不会丢失

2+副本情况下，节点故障索引可读可写，数据不会丢失

1. 性能保障

聚焦业务从读&写两方面评估性能，可以从如下方面衡量：

数据量级，峰值tps & 磁盘使用（包括是否需要使用SSD等）

数据延时要求，ES提供近实时数据服务，适当的延时可以提高tps性能

服务范围，峰值qps & 聚合请求会比检索请求更消耗资源

1. 稳定性保障

资源评估，结合tps、qps及服务内容合理评估所需的资源

业务属性，线上业务还是离线业务，是否需要重点保障

聚焦服务内容，业务本身是否是比较高的负荷，是否会对其它业务造成影响

为了避免/减低业务之间相互影响，目前有一下两种策略：

1. 分集群部署，彻底杜绝干扰

负荷较高（容易干扰到其它业务）单独部署

需要重点保障业务单独部署

B、 同一集群，资源隔离

原则上同分集群部署无异

利用ES的“机架/资源区”概念对集群资源进行物理隔离

两种策略，各有利弊：

分集群部署更清晰更彻底，更容易理解

资源隔离，管理成本更低，更符合分布式水平扩容概念

## 集群扩容

1. 瓶颈评估

集群性能遇到瓶颈，如内存&磁盘空间比超过了理论值

更高的性能要求（tps、qps及响应速度等因素）

新的资源池（更高的业务保障，不跟其它业务相互影响等）

1. 扩容方案

3台服务器做为1个资源区，作为1个扩容单元

目前技术可以做到：

1个业务可以严格指定1个及多个资源区

1个资源区可以为多个业务使用

因此，我们在做集群资源隔离实际就在于如何管理好这些资源区

## 集群安全

ES 本身不支持集群权限安全认证

目前比较普遍在使用的权限安全认证方案是使用Elastic公司提供的shield权限安全认证插件，该插件需要商业付费

1. shield 权限认证插件
2. 支持功能齐全

有角色（role）&用户（user 概念）

role 可以对多音指定读、写及管理等权限

user 通过role 创建，1个用户可以指定多个角色

1. 认证方式多样化

支持权限库认证，将创建的用户直接保留在ES索引里

支持ldap库认证

支持文件密码认证

## 性能优化

1. 写性能优化

批量提交，建议4MB左右

更少的副本，初始导入可不设置副本待导入结束后重置副本

适当加大延时，默认1s刷新1次（ES近实时由来）

禁用”\_all”

不要索引的列不要创建索引，index:no

适当加大并发

延长日志回滚时间间隔

分片均匀分布在各节点上

增加分片，不要轻易增加分片

加大索引缓冲区

增强硬件，如SSD盘

1. 读性能优化

分片均匀分布

禁用“\_all”

定期合并索引文件

数据冷热分离

定期将冷数据从热数据资源区迁移到冷数据资源区

为热数据提供更优的硬件设备

自定义路由，结合数据属性对数据分类存储

如地图检索，将同一个区域的数据放在相同分片列表

请求时指定路由，避免扫描不必要的分片

数据类型严格约束

时间类型不要定义为字符串

能用int类型就不要用long

只存储索引不保留文档内容

将文档内容放KV存储

数据预热

据具体情况可以定时将常用的数据预热到内存中

# 测试数据

## 测试环境

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| nodeIp | es.nodeId | 内存 | CPU | 硬盘 | 是否master | 是否datanode | 是否clientnode |
| 172.16.152.105 | node\_105 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.105 | node\_105\_2 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.106 | node\_106 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.106 | node\_106\_2 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.109 | node\_109 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.109 | node\_109\_2 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.110 | node\_110 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.110 | node\_110\_2 | 128GB | 40Vcores | 1.5T\*5(SSD) | Y | Y | Y |
| 172.16.152.111 | client\_111 | 128GB | 40Vcores | 1.5T\*5(SSD) | N | N | Y |
| 172.16.152.111 | client\_111\_2 | 128GB | 40Vcores | 1.5T\*5(SSD) | N | N | Y |

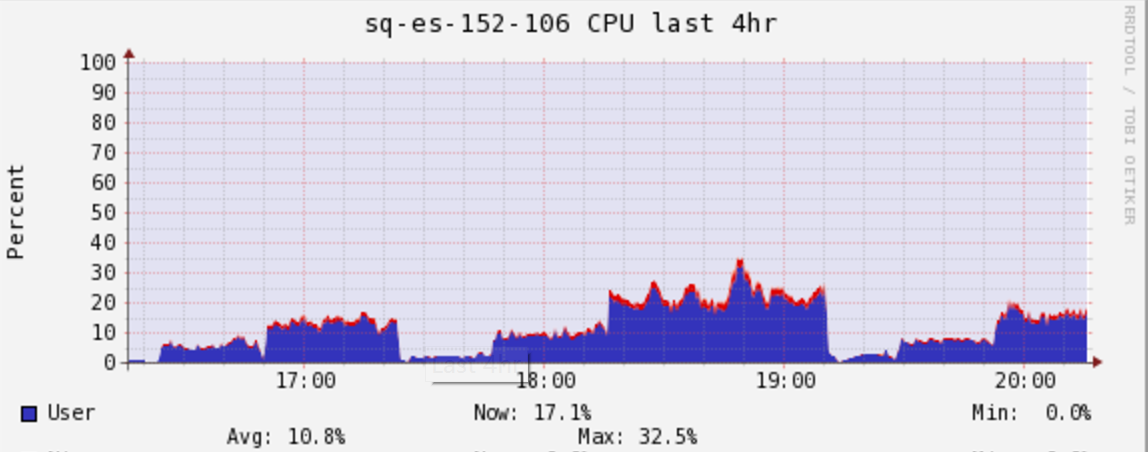
## 数据索引

1. 单表索引

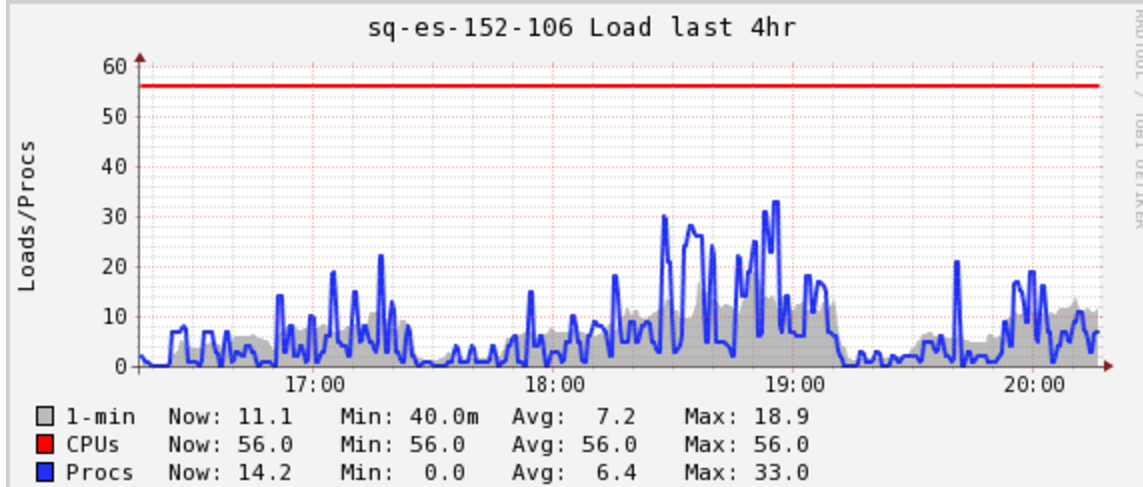
|  |  |
| --- | --- |
| 数据大小 | 列情况 |
| 500B | Int：10  Long：10  Double：10  Timestamp：10  String：10，每列20个字节 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 并发数 | 分片数 | 副本数 | 写入性能/s | 磁盘空间使用/亿条 |
| 1 | 8 | 0 | 5850 |  |
| 5 | 8 | 0 | 19500 |  |
| 10 | 8 | 0 | 33000 | 159GB |
| 并发数 | 分片数 | 副本数 | 写入性能/s | 磁盘空间使用/亿条 |
| 1 | 8 | 1 | 4950 |  |
| 5 | 8 | 1 | 14800 |  |
| 10 | 8 | 1 | 24700 | 169GB/327GB |

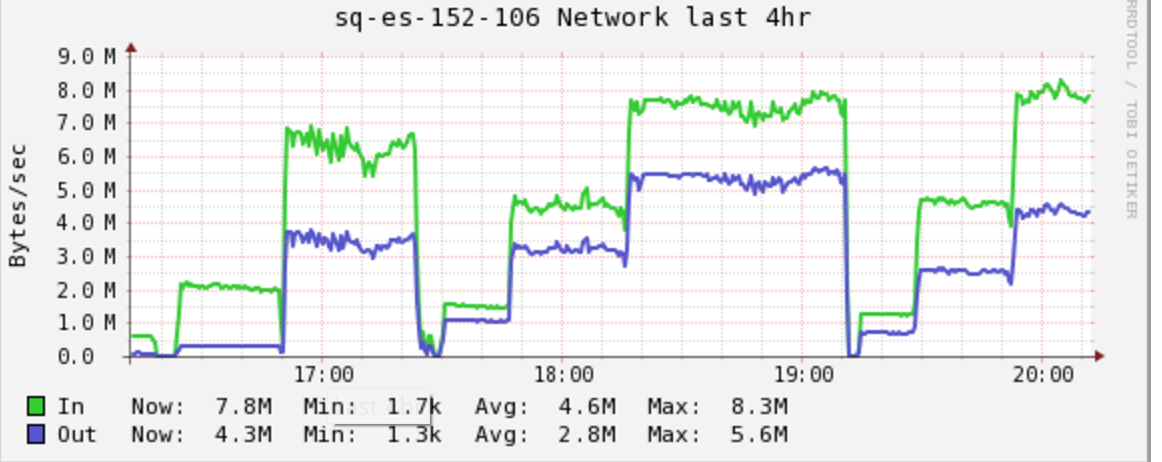
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 并发数 | 分片数 | 副本数 | 写入性能/s | 磁盘空间使用/亿条 |
| 1 | 16 | 0 | 6500 |  |
| 5 | 16 | 0 | 24800 |  |
| 10 | 16 | 0 | 39600 | 171GB |



CPU 使用率



CPU Load



网络I/O情况



cpu 等待磁盘I/O

1. 父子表索引

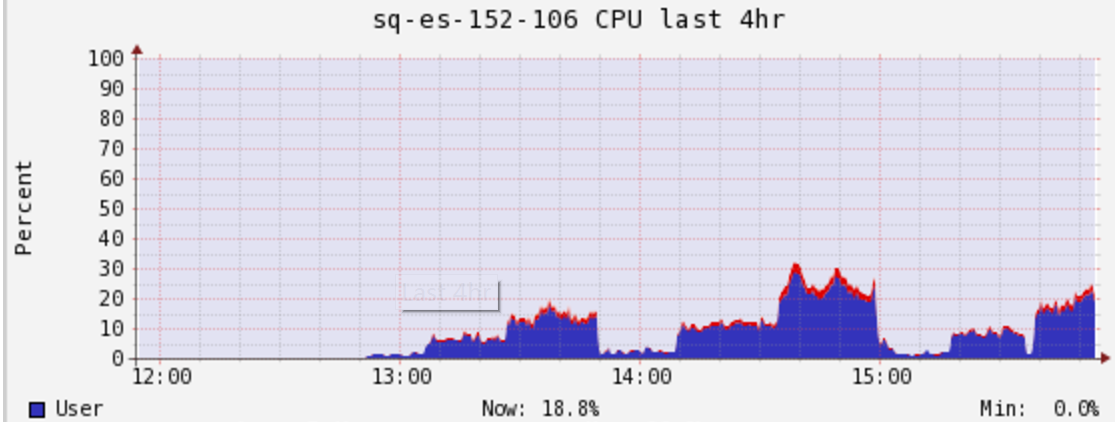
|  |  |  |  |
| --- | --- | --- | --- |
| 父表数据大小 | 列情况 | 子表 | 子表列情况 |
| 200B | Int：4  Long：4  Double：4  Timestamp：4  String：4，每列20个字节 | 300B | Int：6  Long：6  Double：6  Timestamp：6  String：6，每列20个字节 |

其中主从表记录数比1：5（0~10条随机）

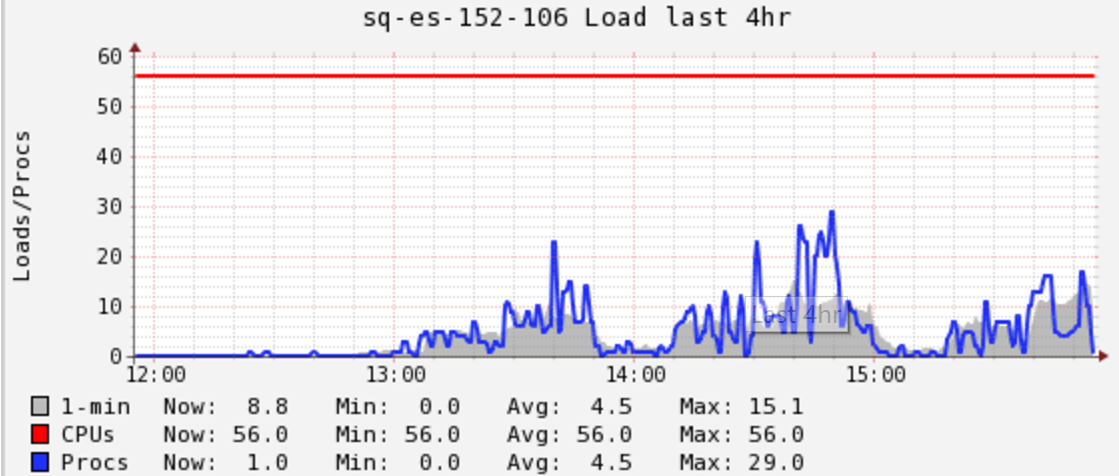
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 并发数 | 分片数 | 副本数 | 主表性能/s | 子表性能 | 磁盘使用(2KW/10Kw) |
| 1 | 8 | 0 | 1610 | 8050 |  |
| 5 | 8 | 0 | 5750 | 28750 |  |
| 10 | 8 | 0 | 9150 | 45750 | 120GB |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 并发数 | 分片数 | 副本数 | 主表性能/s | 子表性能 | 磁盘使用(2KW/10Kw) |
| 1 | 8 | 1 | 1250 | 6250 |  |
| 5 | 8 | 1 | 4300 | 21500 |  |
| 10 | 8 | 1 | 6950 | 34750 | 135GB/255GB |

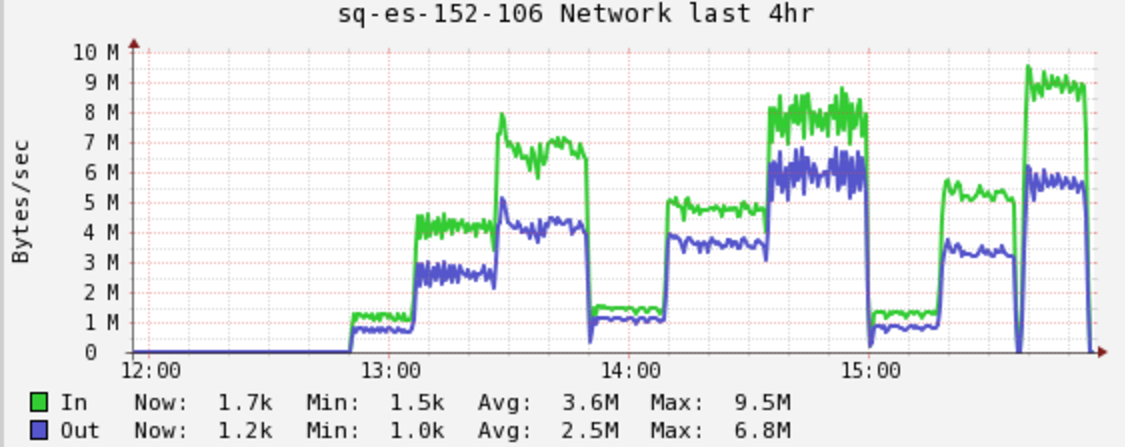
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 并发数 | 分片数 | 副本数 | 主表性能/s | 子表性能 | 磁盘使用(2KW/10Kw) |
| 1 | 16 | 0 | 1800 | 9000 |  |
| 5 | 16 | 0 | 7150 | 35750 |  |
| 10 | 16 | 0 | 12100 | 60500 | 130.5GB |



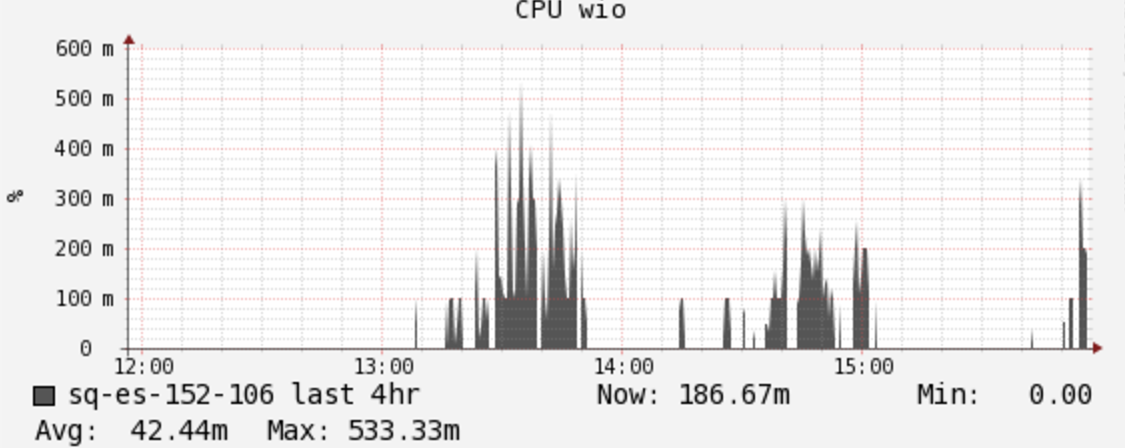
cpu 使用率



Cpu load



网络I/O

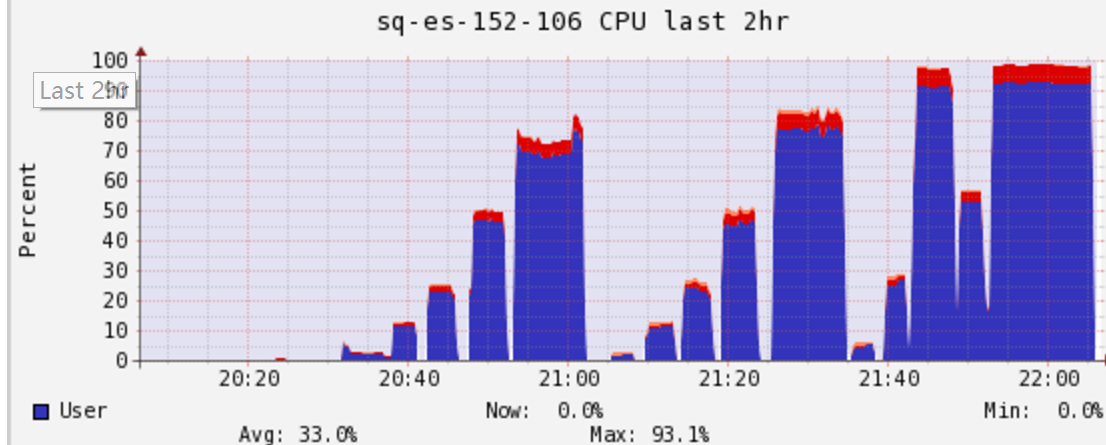


cpu 等待磁盘I/O

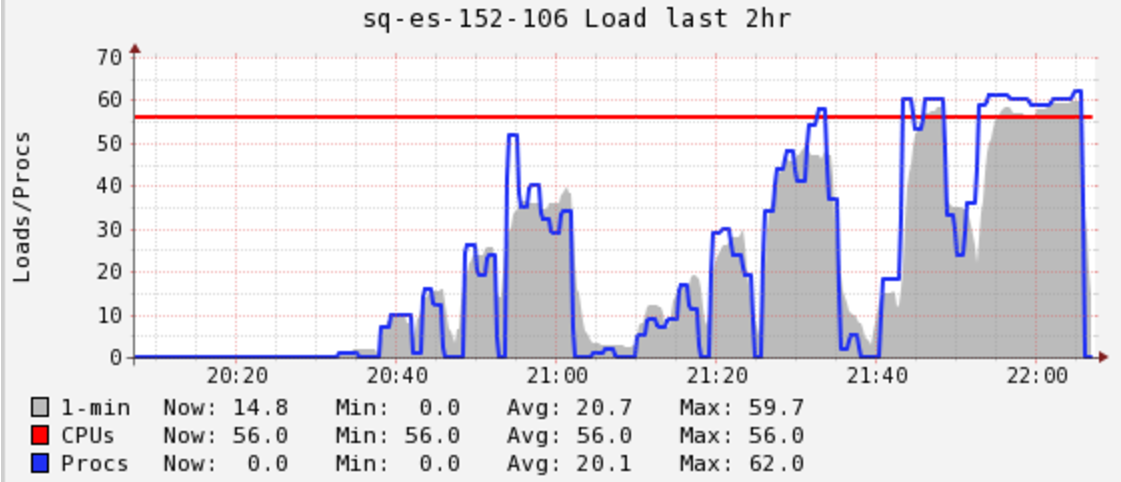
## 数据检索

1. 单表检索（1次测试1个表）

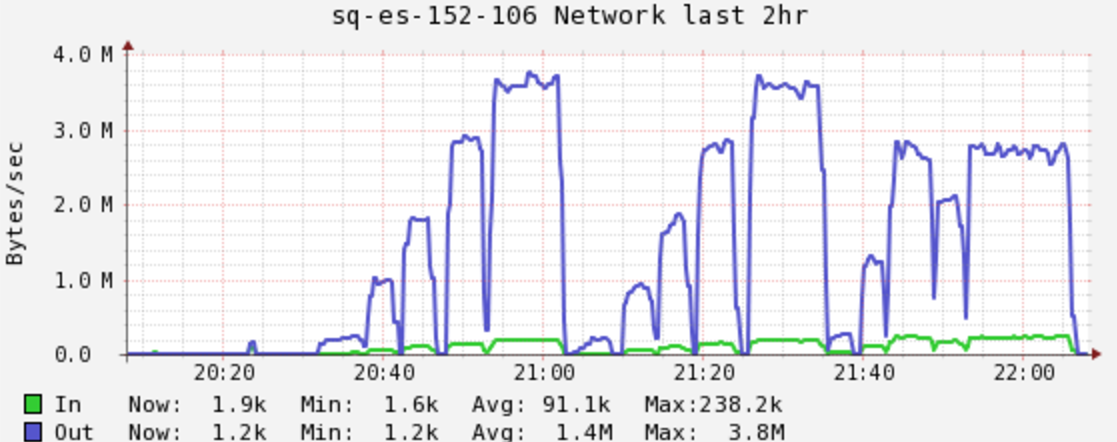
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据量级 | 磁盘空间 | 分片数 | 副本数 | 并发数 | 平均(ms) | 最大(ms) | 最小(ms) | >1000ms |
| 1亿 | 159GB | 8 | 0 | 1 | 192 | 1054 | 4 | 0.1% |
| 1亿 | 159GB | 8 | 0 | 5 | 201 | 883 | 17 | 0% |
| 1亿 | 159GB | 8 | 0 | 10 | 212 | 964 | 19 | 0% |
| 1亿 | 159GB | 8 | 0 | 20 | 274 | 1087 | 27 | 0.5% |
| 1亿 | 159GB | 8 | 0 | 50 | 639 | 2584 | 95 | 14% |
| 1亿 | 169GB/327GB | 8 | 1 | 1 | 242 | 1694 | 27 | 1.5% |
| 1亿 | 169GB/327GB | 8 | 1 | 5 | 227 | 976 | 21 | 0% |
| 1亿 | 169GB/327GB | 8 | 1 | 10 | 235 | 1114 | 30 | 0.1% |
| 1亿 | 169GB/327GB | 8 | 1 | 20 | 284 | 1093 | 18 | 1% |
| 1亿 | 169GB/327GB | 8 | 1 | 50 | 563 | 1632 | 50 | 14% |
| 1亿 | 171GB | 16 | 0 | 1 | 171 | 1046 | 16 | 0.1% |
| 1亿 | 171GB | 16 | 0 | 5 | 165 | 819 | 15 | 0% |
| 1亿 | 171GB | 16 | 0 | 10 | 211 | 868 | 21 | 0% |
| 1亿 | 171GB | 16 | 0 | 20 | 300 | 1281 | 43 | 3% |
| 1亿 | 171GB | 16 | 0 | 50 | 770 | 1828 | 191 | 15% |



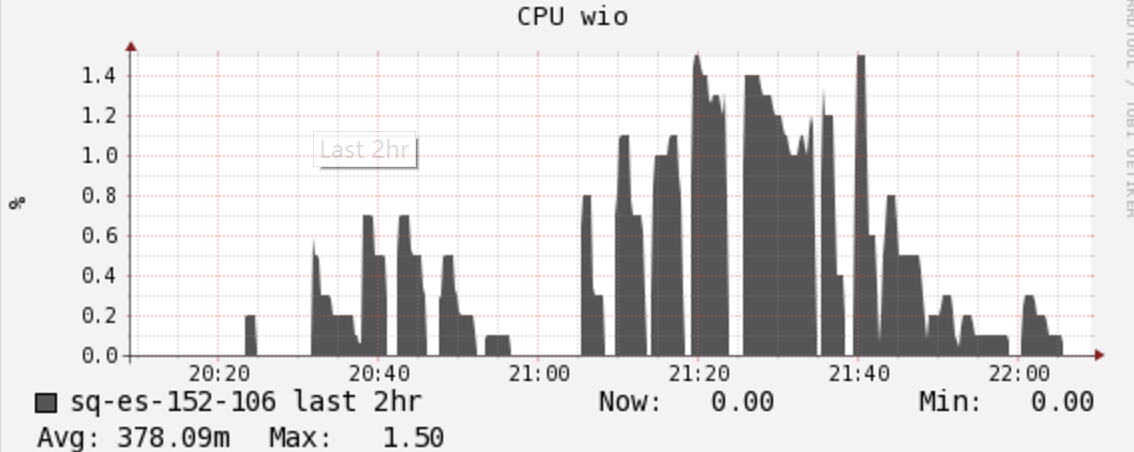
cpu 使用率（凹下去那段是先测试了20并发）



cpu load

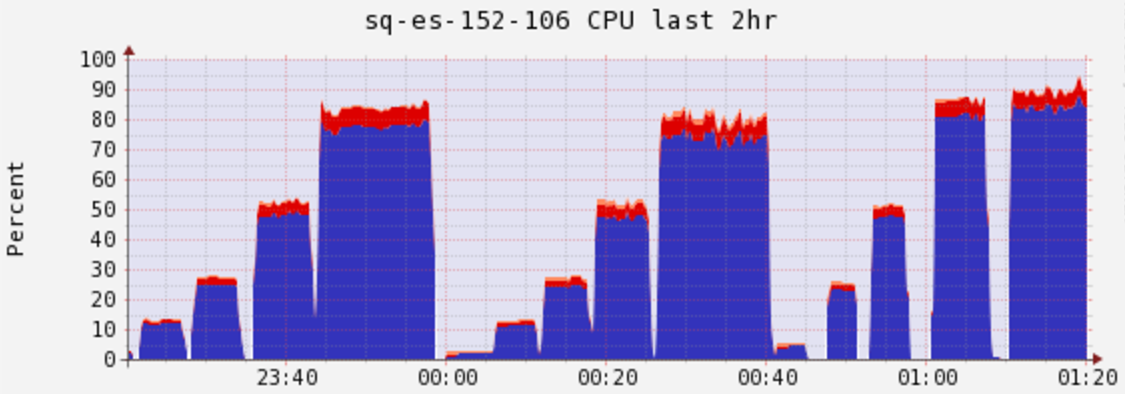


网络I/O

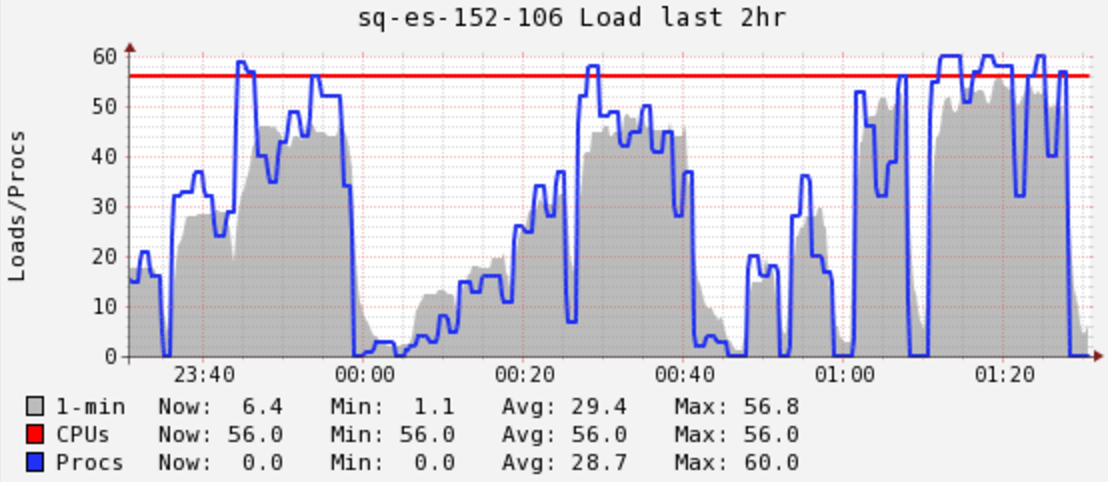


cpu 等待磁盘I/O

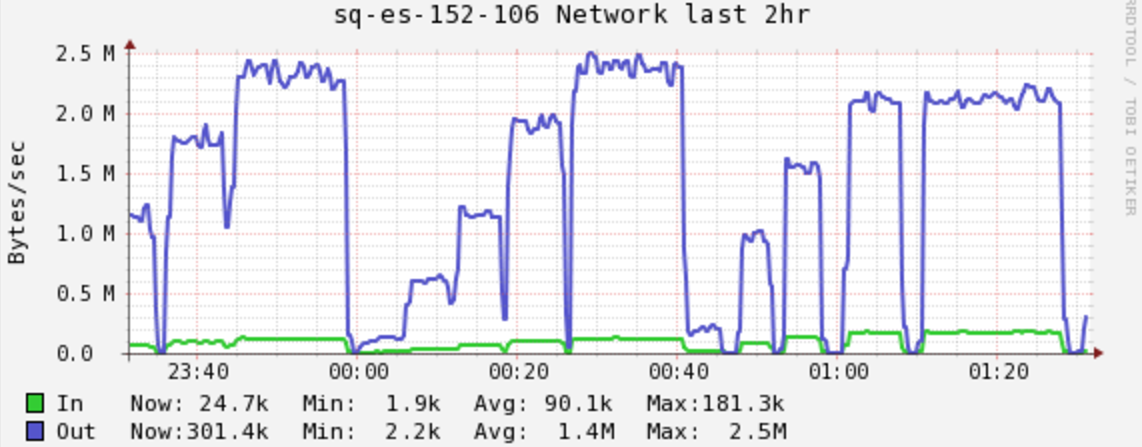
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据量级 | 磁盘空间 | 分片数 | 副本数 | 并发数 | 平均(ms) | 最大(ms) | 最小(ms) | >1000ms |
| 2亿 | 325GB | 8 | 0 | 1 | 332 | 2432 | 38 | 2.5% |
| 2亿 | 325GB | 8 | 0 | 5 | 340 | 1328 | 34 | 3.5% |
| 2亿 | 325GB | 8 | 0 | 10 | 366 | 1436 | 32 | 6% |
| 2亿 | 325GB | 8 | 0 | 20 | 445 | 1728 | 37 | 11% |
| 2亿 | 325GB | 8 | 0 | 50 | 874 | 2634 | 128 | 21% |
| 2亿 | 315GB/629GB | 8 | 1 | 1 | 363 | 1749 | 47 | 5% |
| 2亿 | 315GB/629GB | 8 | 1 | 5 | 333 | 1361 | 34 | 3% |
| 2亿 | 315GB/629GB | 8 | 1 | 10 | 352 | 1355 | 30 | 5% |
| 2亿 | 315GB/629GB | 8 | 1 | 20 | 427 | 1747 | 43 | 13% |
| 2亿 | 315GB/629GB | 8 | 1 | 50 | 849 | 2629 | 69 | 24% |
| 2亿 | 317GB | 16 | 0 | 1 | 235 | 1613 | 19 | 1% |
| 2亿 | 317GB | 16 | 0 | 5 | 255 | 1351 | 29 | 3% |
| 2亿 | 317GB | 16 | 0 | 10 | 282 | 1110 | 28 | 1% |
| 2亿 | 317GB | 16 | 0 | 20 | 396 | 1552 | 40 | 10% |
| 2亿 | 317GB |  |  | 50 | 1071 | 2376 | 373 | 53% |



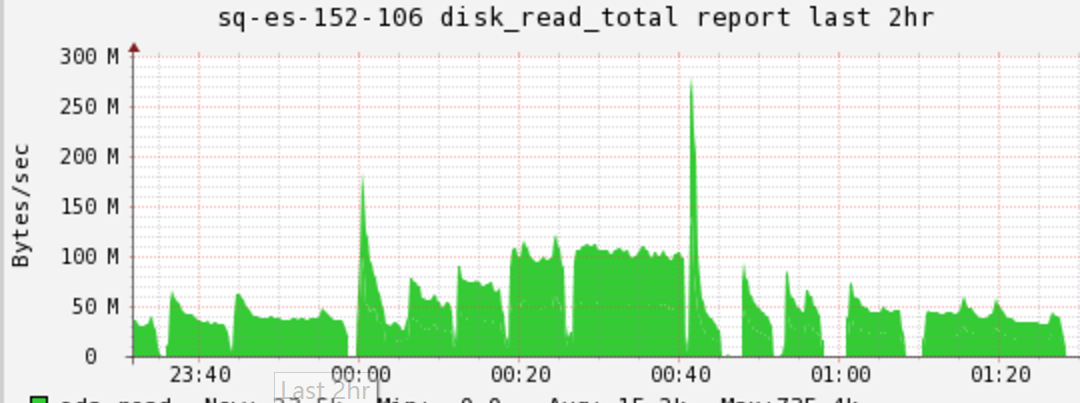
cpu 使用率



cpu load



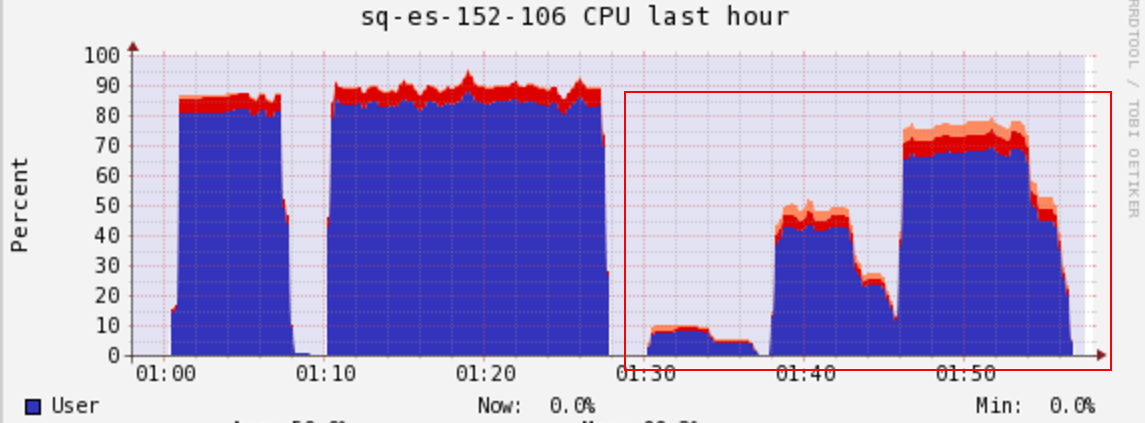
网络I/O



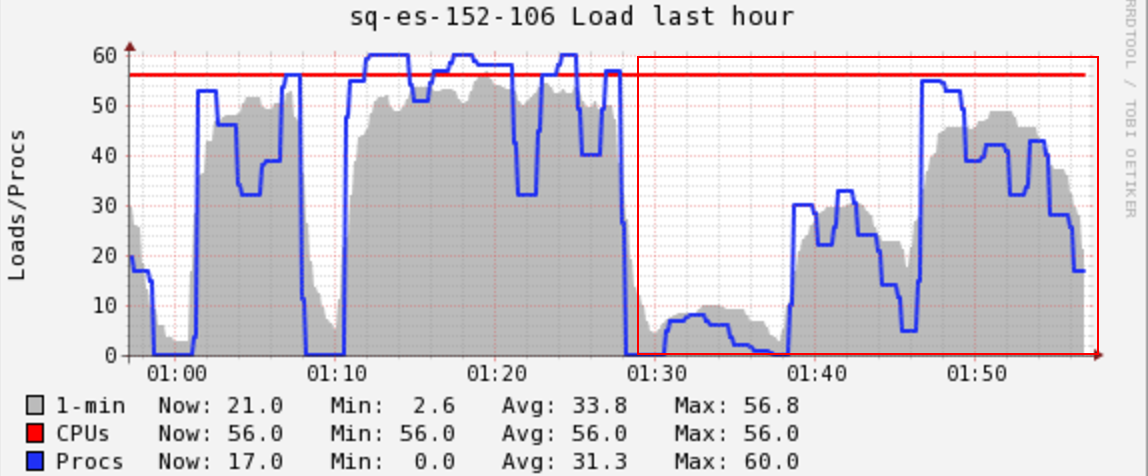
磁盘I/O

2、 单表混合检索（同时测试3个表）

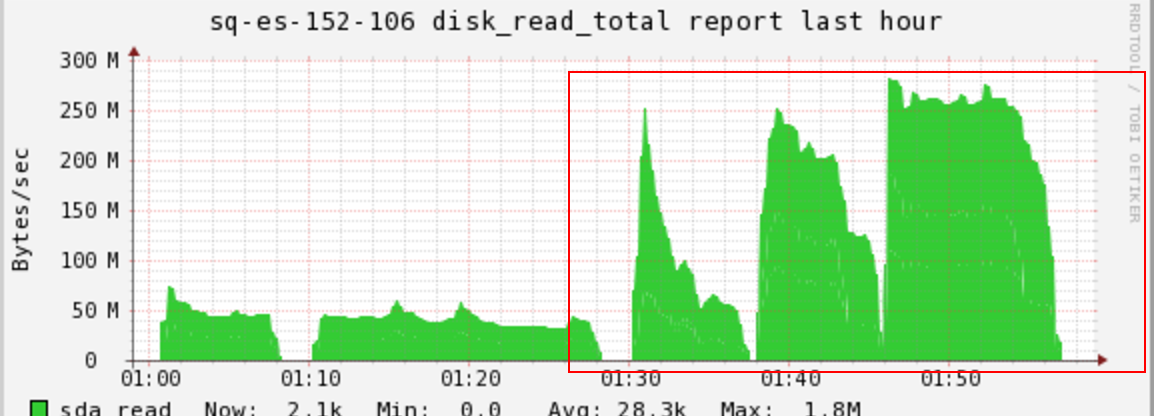
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据量级 | 磁盘空间 | 分片数 | 副本数 | 并发数 | 平均(ms) | 最大(ms) | 最小(ms) | >1000ms |
| 2亿 | 325GB | 8 | 0 | 1 | 393 | 2787 | 27 | 6% |
| 2亿 | 325GB | 8 | 0 | 5 | 459 | 1597 | 52 | 13% |
| 2亿 | 325GB | 8 | 0 | 10 | 644 | 2056 | 98 | 13% |
| 2亿 | 315GB/629GB | 8 | 1 | 1 |  | 2289 | 47 | 7% |
| 2亿 | 315GB/629GB | 8 | 1 | 5 | 496 | 1892 | 51 | 12% |
| 2亿 | 315GB/629GB | 8 | 1 | 10 | 753 | 3090 | 157 | 17% |
| 2亿 | 317GB | 16 | 0 | 1 | 226 | 951 | 22 | 0% |
| 2亿 | 317GB | 16 | 0 | 5 | 298 | 1192 | 35 | 1% |
| 2亿 | 317GB | 16 | 0 | 10 | 498 | 2224 | 49 | 12% |



cpu 使用率



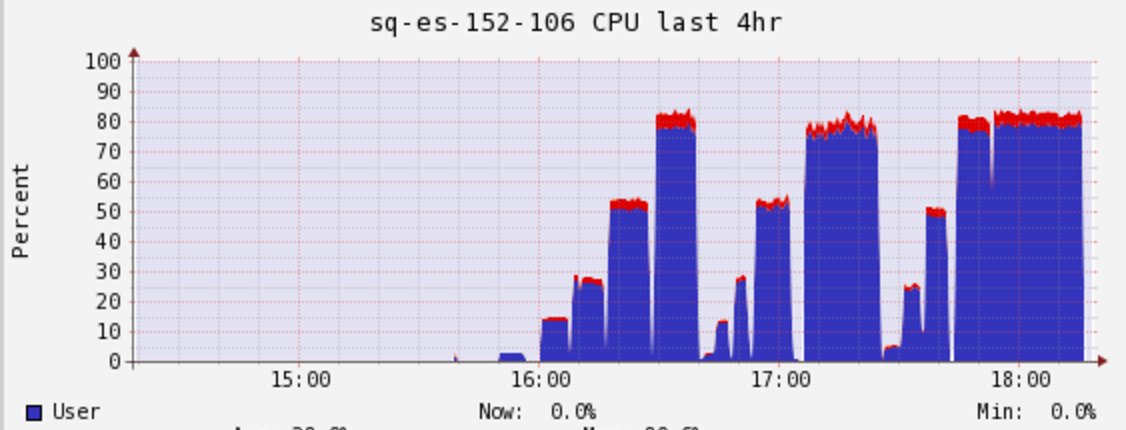
cpu load



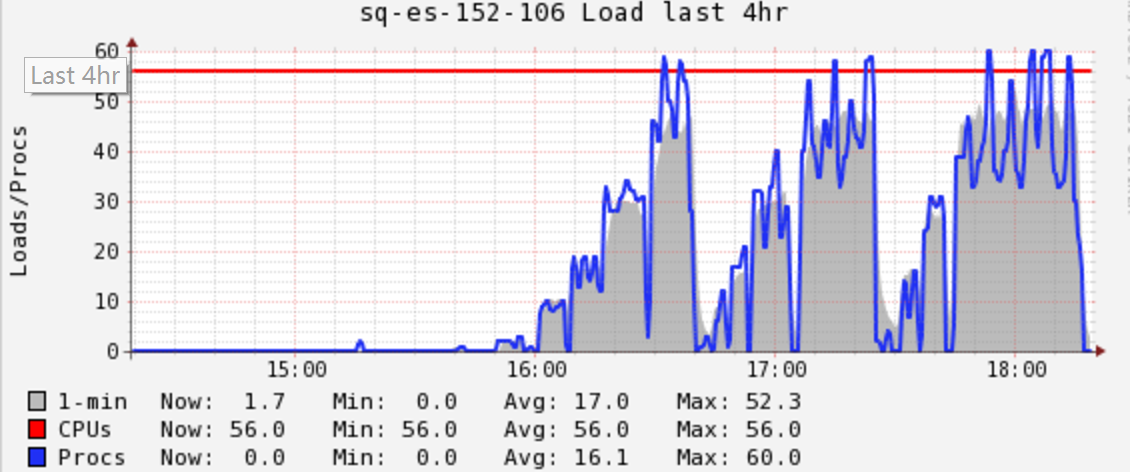
磁盘I/O

3、 主从表检索

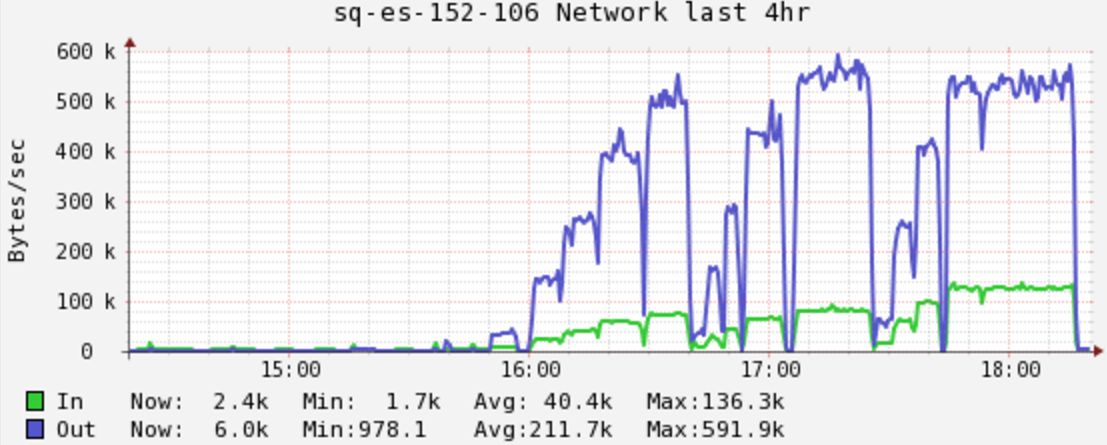
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 主表数据量 | 子表数据量 | 磁盘空间 | 分片数 | 副本数 | 并发数 | 平均(ms) | 最大(ms) | 最小(ms) | >1000ms |
| 2000W | 1亿 | 120GB | 8 | 0 | 1 | 390 | 1488 | 37 | 2.50% |
| 2000W | 1亿 | 120GB | 8 | 0 | 5 | 445 | 1914 | 35 | 6 |
| 2000W | 1亿 | 120GB | 8 | 0 | 10 | 496 | 1873 | 45 | 12 |
| 2000W | 1亿 | 120GB | 8 | 0 | 20 | 651 | 2351 | 51 | 21 |
| 2000W | 1亿 | 120GB | 8 | 0 | 50 | 1292 | 4110 | 332 | 56 |
| 2000W | 1亿 | 135GB/255GB | 8 | 1 | 1 | 447 | 1894 | 52 | 5 |
| 2000W | 1亿 | 135GB/255GB | 8 | 1 | 5 | 405 | 1463 | 63 | 3 |
| 2000W | 1亿 | 135GB/255GB | 8 | 1 | 10 | 439 | 1546 | 85 | 6 |
| 2000W | 1亿 | 135GB/255GB | 8 | 1 | 20 | 584 | 2210 | 55 | 19 |
| 2000W | 1亿 | 135GB/255GB | 8 | 1 | 50 | 1129 | 3938 | 93 | 48 |
| 2000W | 1亿 | 130GB | 16 | 0 | 1 | 264 | 1184 | 26 | 0.3 |
| 2000W | 1亿 | 130GB | 16 | 0 | 5 | 286 | 1144 | 20 | 0.6 |
| 2000W | 1亿 | 130GB | 16 | 0 | 10 | 370 | 1366 | 38 | 0.7 |
| 2000W | 1亿 | 130GB | 16 | 0 | 20 | 545 | 1837 | 78 | 12 |
| 2000W | 1亿 | 130GB | 16 | 0 | 50 | 1369 | 3117 | 477 | 69 |



cpu 使用率



cpu load



网络I/O

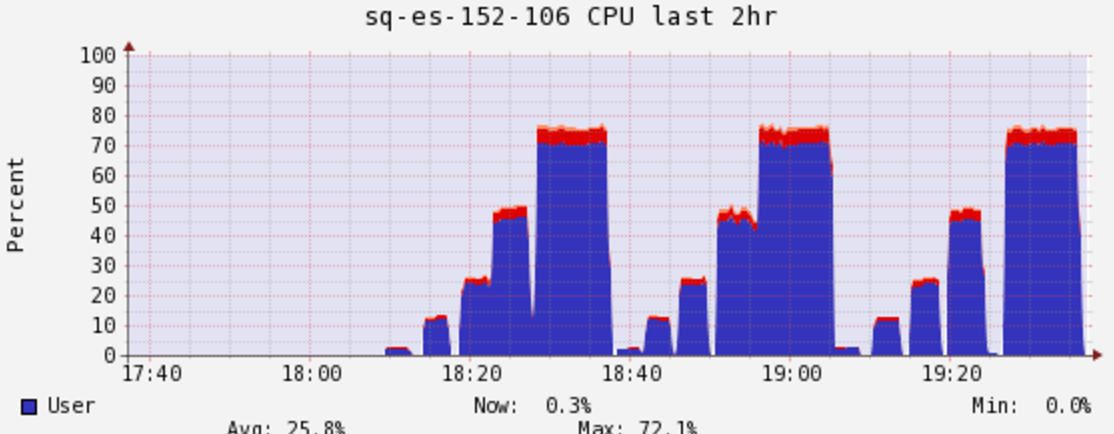


cpu 等待磁盘I/O

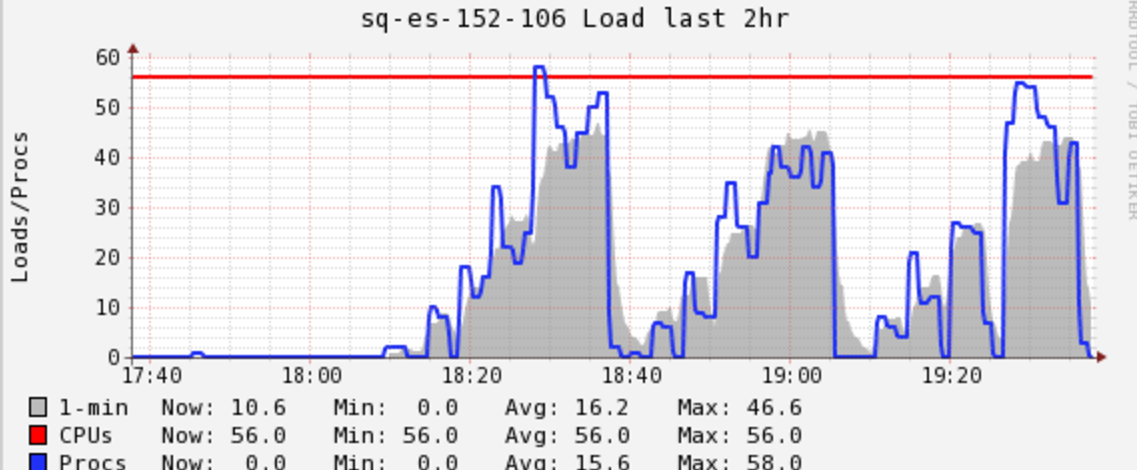
## 数据统计&分析

1. 单表聚合

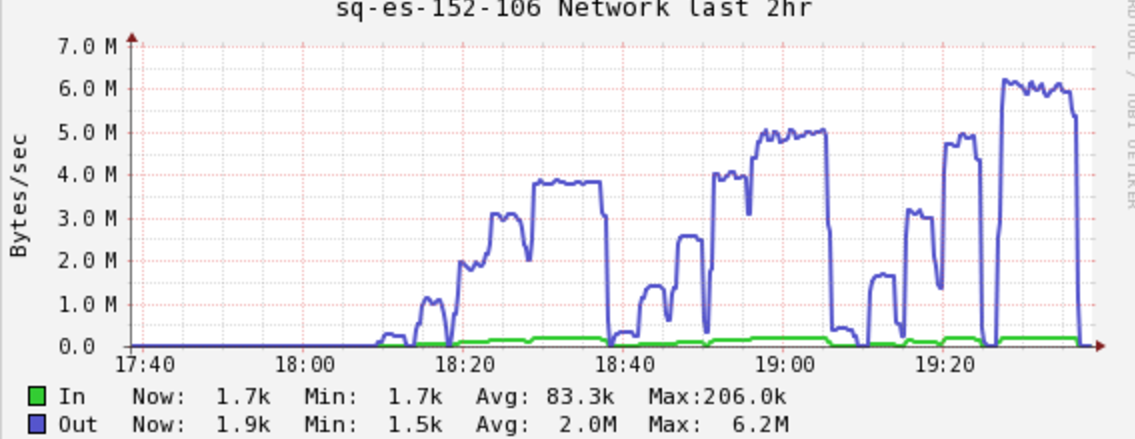
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据量级 | 磁盘空间 | 深度 | 分片数 | 副本数 | 并发数 | 平均(ms) | 最大(ms) | 最小(ms) | >1000ms |
| 1亿 | 159GB | 1 | 8 | 0 | 1 | 199 | 909 | 25 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 5 | 214 | 937 | 20 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 10 | 226 | 984 | 22 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 20 | 289 | 1081 | 25 | 0.3% |
| 1亿 | 159GB | 1 | 8 | 0 | 50 | 570 | 1681 | 77 | 16% |
| 1亿 | 159GB | 1 | 8 | 0 | 1 | 189 | 777 | 15 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 5 | 207 | 788 | 17 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 10 | 219 | 965 | 19 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 20 | 278 | 1119 | 28 | 1.2% |
| 1亿 | 159GB | 1 | 8 | 0 | 50 | 564 | 1746 | 53 | 15% |
| 1亿 | 159GB | 1 | 8 | 0 | 1 | 181 | 907 | 7 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 5 | 200 | 870 | 17 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 10 | 230 | 926 | 20 | 0% |
| 1亿 | 159GB | 1 | 8 | 0 | 20 | 274 | 1062 | 25 | 1% |
| 1亿 | 159GB | 1 | 8 | 0 | 50 | 578 | 1707 | 67 | 14% |



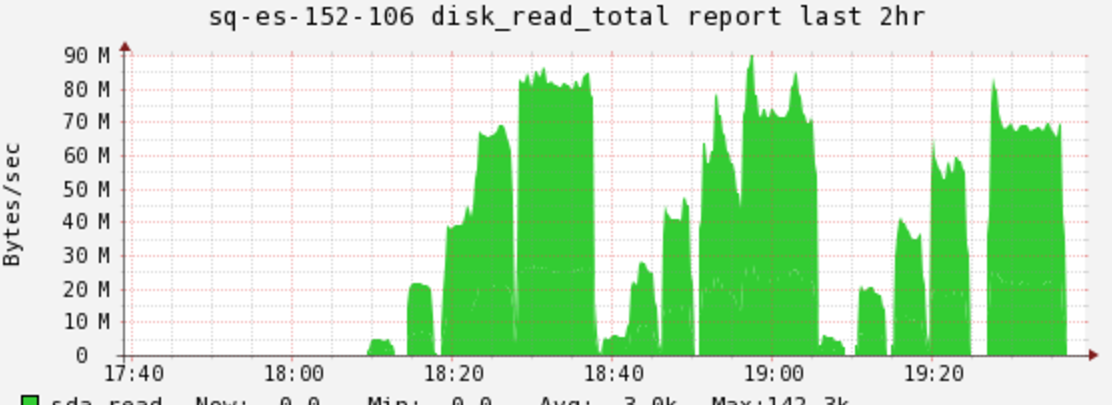
cpu 使用率



cpu load



网络I/O



磁盘I/O

## 总结

1. 索引性能
2. 并发写入可以提高索引写入性能，并随着并发数增长效果也在下 降
3. 更多的分片数有更好的索引性能，一定量下优势有限

C、 更多的副本会降低索引性能及增大集群负荷

D、 同1个分片数据在同一个数据目录（同一块磁盘）

1. 检索性能
2. ES 不适合高并发检索
3. 副本对检索性能几无影响
4. 更多的分片在并发低的时候可以提高检索性能，并发上去后性能 反而有所下降

D、 相比单标检索，主从表关联检索性能较差

E、 可以根据性能&qps适当调整分片大小，建议20GB

F、 检索对网络I/O无感

G、 耗磁盘I/O，建议使用SSD盘

H、 高qps请求，耗cpu

I、 ES检索有缓存，实际性能相较测试结果偏差（参考混合检索测试 结果）

1. 聚合统计
2. 聚合统计相比较检索性能差异不大
3. 聚合统计更耗内存资源

## 使用范例

1. 禁用“\_all”，在创建索引时指定



1. 资源隔离，指定索引资源区，创建时指定（可调整）



1. 索引分片均匀分布到各个data node

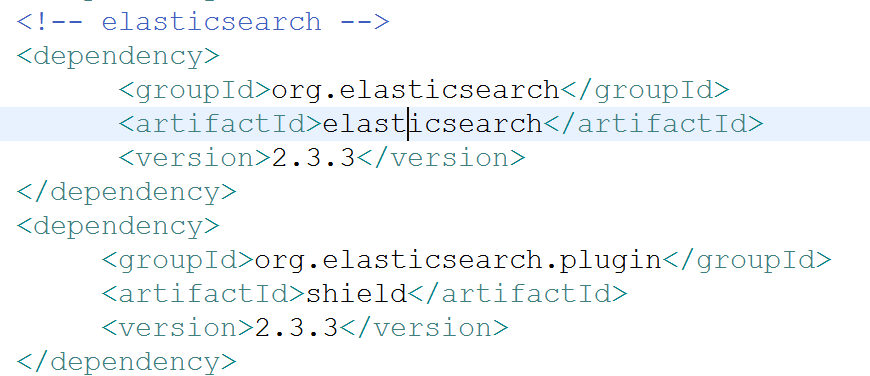


结合分片数、副本数、指定资源区data node 数，计算出每个data node 最多分配多少跟shard

1. ES Java api

可参考：https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/index.html

A、 maven 依赖



B、 tcp 连接（使用单例连接方式）



*client*.addTransportAddress

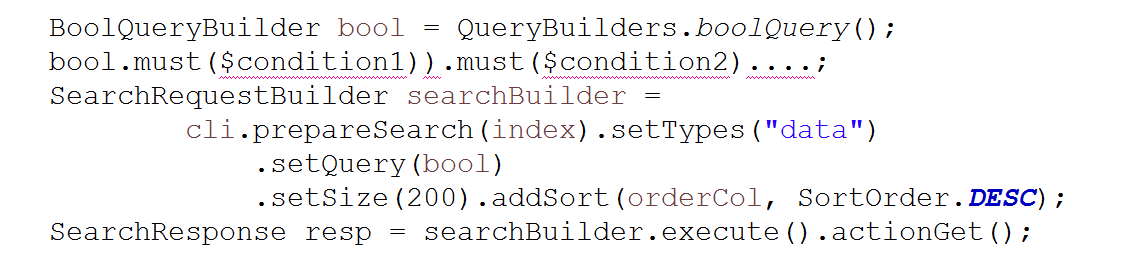
指定连接ip列表

client.transport.siniff

默认（true）自动扫描到所有集群节点

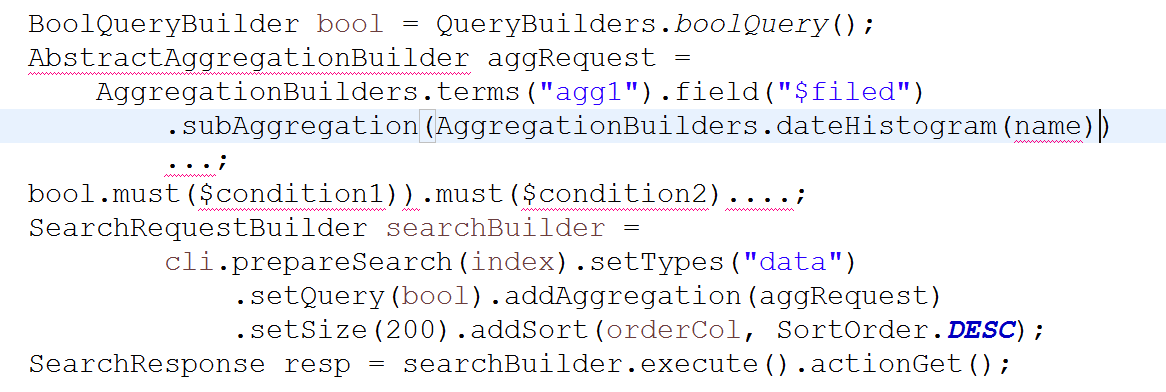
由于master / data node同时也是client node，为防止业务app 将请求发送到master/data node 可以设置为false（否则分节点 单独部署也就失去意义）

C、 检索



检索条件可以通过boll条件树自由组合

D、 聚合



聚合请求通过subAggregation做多层嵌套

# 故障&运维

## 故障汇总

1. 微服务日志消费延时

故障排查：

有1个索引（日增量300GB）的分片分布在同1个节点上，造成负载不均，节点负荷过大并引起了连锁反应（其它索引也延时了）

Logstash 消费kafka写入ES默认配置，为进行优化

解决方案：

调整索引配置 ， 使索引均匀分布

调整logstash 写ES并发数（从1个调整为3个）

调整logstash写ES的批次大小（从500调整到5000）

1. data node 出现OOM退出集群

故障排查：

有grafana有1个聚合任务配错了 ，查询时间跨度大且颗粒度小（1s）

解决方案：

关闭该任务

调整ES 自我保护机制阈值

1. data node 出现OOM退出集群

故障排查：

埋点日志聚合请求做结果集合并时发生OOM

解决方案：

调整请求发送策略，将请求发送到ES client node

1. 节点同master失联，退出集群

故障排查：

该节点load太高导致

load太高是由于grafana有1个链接地址同时发送了80多个的买点日志聚合请求

解决方案：

将1个月前的买点日志暂时关闭

暂时关闭该grafana链接，找到业务方进行请求优化（对聚合请求进行合并）

1. 监控墙数据延时

故障排查：

盘古资源紧张，标签数据全量同步没有在预定时间及时结束，导致在集群高峰期负荷过大

解决方案：

优化标签数据同步（全量转增量），正在进行中

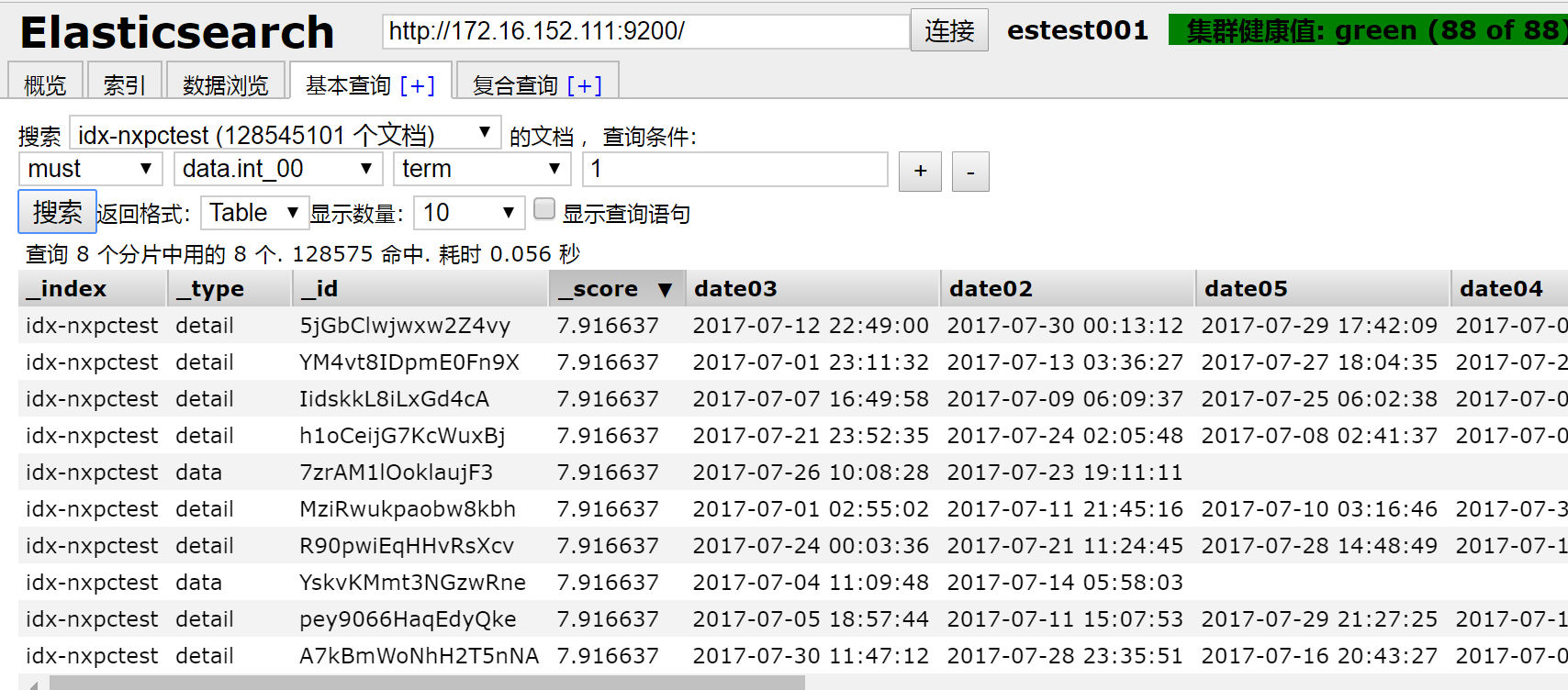
对标签数据进行资源区隔离，服务器在采购中

## 运维

1. head web插件

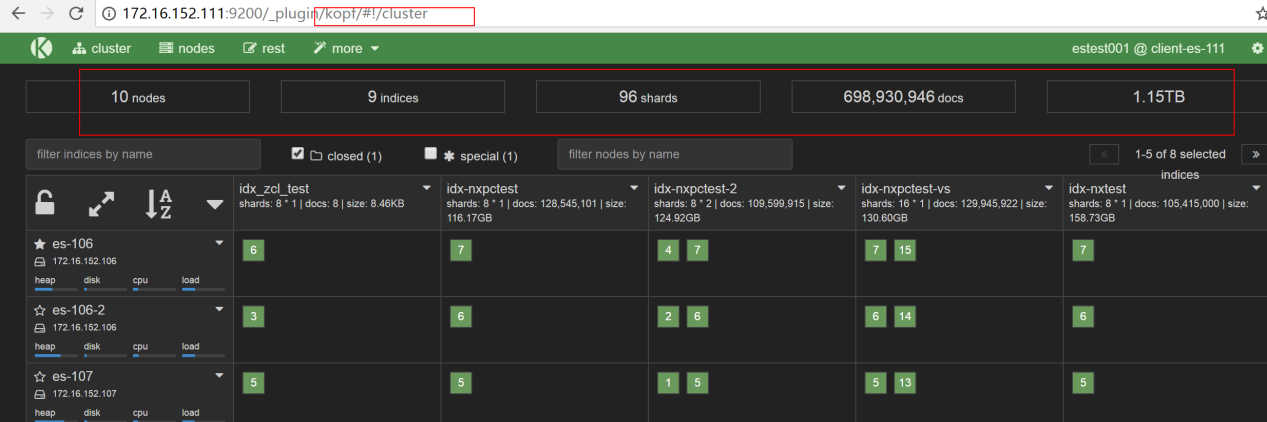
可以在head插件上比较便捷创建、检索索引





1. kopf 创建

kopf插件可以清晰看到集群负荷情况



1. 集群告警

根据ES提供的API获取集群信息，具体参考：

<https://www.elastic.co/guide/en/elasticsearch/reference/2.3/cat.html>

<https://www.elastic.co/guide/en/elasticsearch/reference/2.3/cluster-stats.html>

<https://www.elastic.co/guide/en/elasticsearch/reference/2.3/cluster-nodes-stats.html>

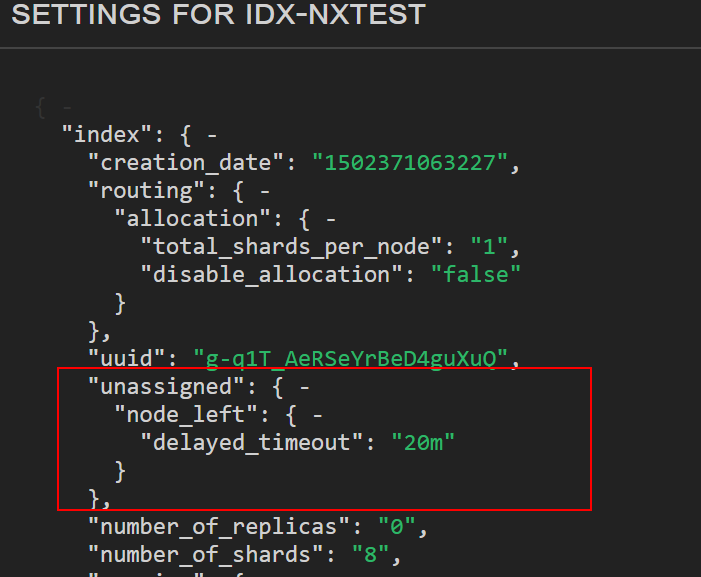
定时从这些接口爬取集群信息，判断集群的健康情况并进行相应告警

1. 运维小窍门

A、 延迟故障分片自动分配，避免重新创建分片（很消耗资源）

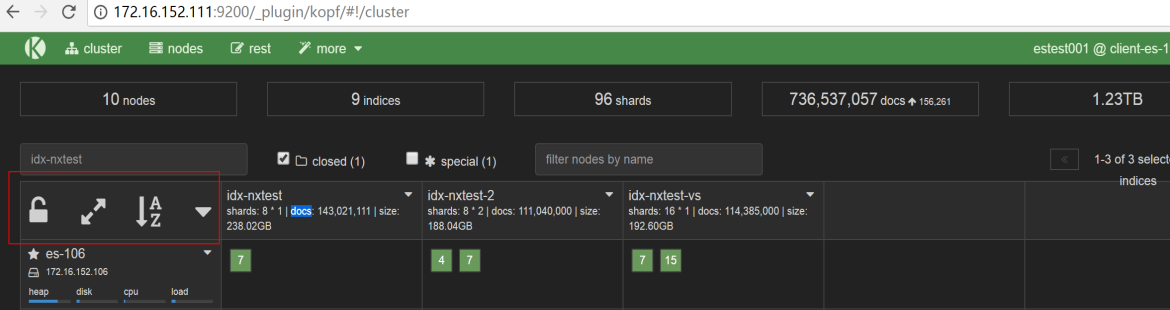
如果节点假死 ， 则故障分片可恢复

故障节点快速恢复/重启，故障分片可快速恢复



B、 重启节点前，先关闭集群分片分配

节点退出集群后，集群会默认为集群故障，并对故障分片重新分配



C、 慢查询日志review

在集群出现负荷（cpu load）告警时，查看集群慢查询日志看是 否有不合理请求频繁在刷并定位到具体索引，在必要情况下暂时 关闭该索引以保障其它业务

慢查询日志：

${ES\_Log\_Path}/${clusterId}\_index\_search\_slowlog.log