ElasticSearch6.x/7.x新特性及性能优化

3306Π-上海站

上海-云砺信息

杨显奎 20200425

目录

1.1ElasticSearch介绍

ElasticSearch是一个实时的分布式搜索分析引擎，内部使用Lucene做索引与搜索。其中必须说明下，Lucene 是一个JAVA实现的jar包，用来管理搜索引擎索引的库，但是必须要懂一点搜索引擎原理的人才能用的好，所以后来又有人基于 Lucene 进行封装，写出了 Elasticsearch。

ES将对搜索引擎的操作都封装成了restful的api，通过http请求进行操作，同时，考虑到海量数据，顺便实现了分布式，是一个可以存储海量数据的分布式搜索引擎。

基于ES，可以很容易的搭建自己的搜索引擎系统，用于分析日志，或者配合已有系统搭建数据分析及查询的准实时平台系统；

全文检索、日志分析、指标分析、APM等众多场景；

1.2发展由来

ES是如何产生的？传统数据库(比如Oracle/MySQL等)，在数据量比较大的时候比如单表几个亿甚至更多的时候，由于存储数据的格式及索引检索限制，很快就出现性能瓶颈，

导致后续一系列的架构改动及代码级联变动，前中后端同学苦不堪言，

问题来了，那么大规模的数据需求改如何检索？

如，当系统数据量单表上了亿，十亿，甚至百亿条的时候，我们在没法使用硬件堆，或者是硬件代价太大的时候，往往会重新从架构的角度去考虑问题：

1）用什么数据库好(MySQL，Oracle，Mongodb，hbase …)；

2）如何解决单点故障(lvs，F5，A10，Zookeep，MQ等)；

3）怎么来保证数据的安全性(热备，冷备，异地多活等)；

4）最麻烦的问题，怎么解决检索问题(数据库代理中间件有mysql-proxy、Cobar、MaxScale等)

5）解决统计分析问题(离线、近实时)；

针对上面的问题，传统数据是如何解决的？

在关系型数据库使用中，我们通常采用如下或者类似架构去解决查询和写入瓶颈

1）通过主从备份的方式解决数据安全性问题

2）数据库代理中间件心跳检测，解决单点故障问题；

3）代理中间件将查询语句分发到各个slave节点进行查询、汇总结果集；

如上，数据量巨大的时候弊端明显，分不分库表，分了库表聚合麻烦，不分，查询检索简直是噩梦，怎么办？能不能找到更合理的处理巨量数据的方式？

思考思考思考 。。。

根据上面的综合思考，把数据完全放入内存或者不放入内存，都不能完全解决问题，不是成本问题，就是效率问题，折中思考，从源头上寻找解决办法：

1）存储数据的时候按照有序存储

2）将数据和索引分离

3）压缩数据

综合上面几条的意思，引申出了当下最火的大数据检索聚合产品 ElasticSearch。

1.3 未来发展

1.多个一线公司的使用实践；

百度（在云分析、网盟、预测、文库、钱包、风控等业务上都应用了ES，单集群每天导入数据量达到惊人的30TB+数据，总共每天60TB+）；

新浪ES 如何分析处理32亿条实时日志；

阿里ES 构建挖财自己的日志采集和分析体系；

有赞ES 业务日志处理；

其他如金融公司对ES的使用也是水涨船高，越来越受欢迎，我们公司目前主业务在ES集群的数据量是60T左右；

2.ElasticSearch本身产品生态及社区活跃度的日渐成长：

产品从单一的ES，到现在工具套件 Elastic(ELK) Stack

社区国内最大的平台 http://elasticsearch.cn/

国外有 https://discuss.elastic.co/

7.x新特性

其他的具体新特性列表请移步官网文档，今天我们只是选取几个经常用到或者比较重要的特性来一起了解一下下；

https://www.elastic.co/guide/en/elasticsearch/reference/7.0/breaking-changes-7.0.html

2.1索引的默认分片数

集群分片数number_of_shards在6.x默认为5，7.x开始，默认为1，使得在数据量不是太大的时候日增索引分片量大减少，使得master在管理分片的资源降低

```
373
374  ## index
375  GET /twitter/_settings
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
```


```
{
  "twitter" : {
    "settings" : {
      "index" : {
        "search" : {⟺},
        "indexing" : {⟺},
        "number_of_shards" : "1",
        "provided_name" : "twitter",
        "creation_date" : "1586246995379",
        "number_of_replicas" : "1",
        "uuid" : "6zSWzZb7TtWv2ZYHL69p1g",
        "version" : {
          "created" : "7040099"
        }
      }
    }
  }
}
```

2.2文档存储结构变化:去除type

在6.x，官方提到7.x时候会删除type，并且6.8版本开始规定每一个index只能有一个type，7版本使用默认的_doc作为type，官方文档将在8.x会彻底移除type;

级联的变化，API请求对索引文档进行操作的时候，默认使用的type就是_doc;

如获取某个索引的ID文档为 GET /index_name/_doc/id

2.3索引mapping关系不再有type属性

如图，在6.x版本创建索引的映射关系语法，在7.x报错如下

```
## mapping type
PUT es_test_index
{
  "settings": {
    "index": {
      "analysis.analyzer.default.type": "ik_max_word"
    }
  },
  "mappings": {
    "item": {
      "properties": {
        "site_id": {
          "type": "long",
          "index": true
        },
        "content": {
          "type": "text"
        }
      }
    }
  }
}
```

```
{
  "error": {
    "root_cause": [
      {
        "type": "mapper_parsing_exception",
        "reason": "Root mapping definition has unsupported
          parameters:  [item : {properties={site_id={index=true,
          type=long}, content={type=text}}}]"
      }
    ],
    "type": "mapper_parsing_exception",
    "reason": "Failed to parse mapping [_doc]: Root mapping
      definition has unsupported parameters:  [item : {properties
      ={site_id={index=true, type=long}, content={type=text}}}]",
    "caused_by": {
      "type": "mapper_parsing_exception",
      "reason": "Root mapping definition has unsupported parameters:
        [item : {properties={site_id={index=true, type=long},
        content={type=text}}}]"
    }
  },
  "status": 400
}
```

从官方文档说明知道，7.x版本去除了mapping type，正确示例如下

```
PUT es_test_index
{
  "settings": {
    "index": {
    "analysis.analyzer.default.type": "ik_max_word",
    "number_of_shards":3,
      "number_of_replicas":1
    }
  },
  "mappings": {
    "properties": {
      "site_id": {
        "type": "long",
        "index": true
      },
      "content": {
        "type": "text"
      }
    }
  }
}
```

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "es_test_index"
}
```

2.4分片搜索空闲时跳过refresh

7.x以前版本的数据插入，每1秒都会有refresh动作，这使得es能成为一个近实时的搜索引擎。但是当没有查询需求的时候，该动作会使得es的资源得到较大的浪费;

在es7.x中，如果一个分片处于搜索空闲状态(30秒内都没有查询)，那么插入数据不会被refresh，直到有一个查询过来触发refresh，或者显式的触发refresh。该改动可以显著的插入索引性能;

2.5 Lucene版本的配套升级到最新的9版本

根据之前的介绍，Lucene的性能直接决定ES的性能，Lucene9在top K及其他查询方面有了很大的性能提升；

10亿+查询不在话下；

2.6跨集群备份

在6.5版本开始引入的功能，到7.0版本多次改进之后，可以在生产环境使用，基于跨集群我们可以做集群双活，这个是ES历史性改进的新特性，非常实用的功能之一；

2.7优化器算法改进 Weak-AND算法

属于查询相关性计算的算法， Weak-AND算法在Term Query查询场景号称有3700%的性能提升，我们环境的实际压测效果是提升10倍左右；

如下所示，除了Term检索，Fuzzy，Phrase, Bool And .Bool OR都有大幅的性能提升!

如图

query	before qps	after qps	效率
Fuzzy	46	59	28%
Phase	4	7	87%
Bool And	9.3	23.5	247%
Bool Or	3.3	9.8	292%
Term	33	1160	3700%

2.8熔断机制-线上最爱的需求

如果自带熔断机制，是不是更完美，7.x实现了该功能， Circuit Breaker 在JVM堆栈层面监测内存使用，使得ElasticSearch 比之前更加健壮，设置indices.breaker.fielddata.limit的默认值已从JVM堆大小的60%降低到40%，线上不会再有OOM(内存溢出)的情况；

相关参数参考如下

indices.breaker.total.limit: 30%

indices.breaker.request.limit: 6%

indices.breaker.fielddata.limit: 3%

针对这个特性，我们在慢查询第一阶段就监控中加入被系统拒绝掉的大查询，进行前期优化；

2.9集群调度系统重构

以前版本的ES统一用一套方便的可插拔的集群调度ZenDiscovery,该调度系统非常方便，但是minimum_master_nodes这个配置项如果被遗忘，可能会导致脑裂,而且在大集群中维护该配置项也是个成本很大的工作;

现在7.x中，该部分功能已被重构，升级之后只需要亚秒级就可以完成选举,而老的版本Zen模式可能需要几秒钟才能完成，也不再需要配置minimum_master_nodes，号称更不用担心脑裂问题。

2.10时间戳纳秒级支持，提升数据精度

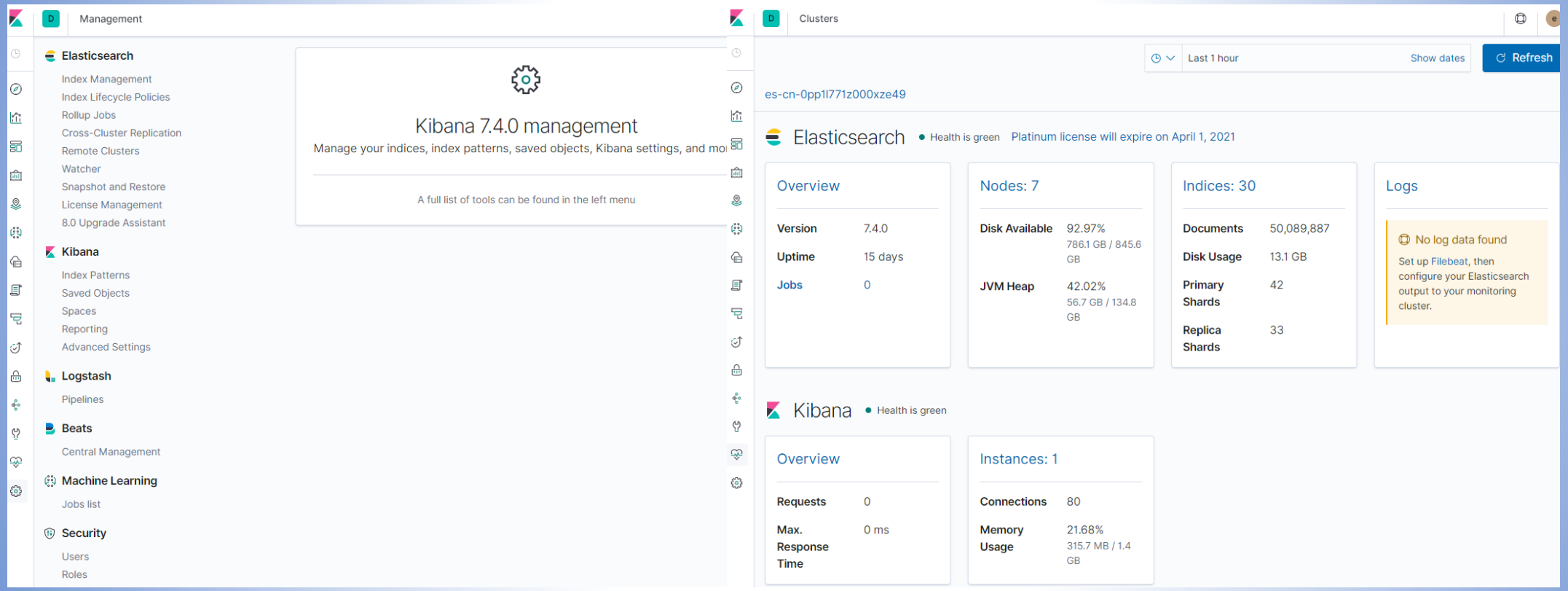
利用纳秒精度支持加强时间序列用例；

到目前为止，Elasticsearch仅以毫秒精度存储时间戳，7.0增加了几个零并带来了纳秒级精度，这提高了高频数据采集用户存储和排序所需数据的精度，很实用；

2.11可视化管理工具kibana功能丰富化

使用的经验表明，ES管理工具里面还数kibana最实用和高效，之前老版本的可视化这块做的不是太友好，7x之后，kibana的可视化已经功能相当丰富，满足ES运维的绝大部分需求；

如下图，kibana的轮廓概貌：


Management
Elasticsearch
Index Management
Index Lifecycle Policies
Rollup Jobs
Cross-Cluster Replication
Remote Clusters
Watcher
Snapshot and Restore
License Management
8.0 Upgrade Assistant
Kibana
Index Patterns
Saved Objects
Spaces
Reporting
Advanced Settings
Logstash
Pipelines
Beats
Central Management
Machine Learning
Jobs list
Security
Users
Roles
Kibana 7.4.0 management
Manage your indices, index patterns, saved objects, Kibana settings, and mo
A full list of tools can be found in the left menu
Clusters
Last 1 hour
Show dates
Refresh
es-cn-0pp1l771z000xze49
Elasticsearch
Health is green
Platinum license will expire on April 1, 2021
Overview
Version 7.4.0
Uptime 15 days
Jobs 0
Nodes: 7
Disk Available 92.97%
786.1 GB / 845.6 GB
JVM Heap 42.02%
56.7 GB / 134.8 GB
Indices: 30
Documents 50,089,887
Disk Usage 13.1 GB
Primary Shards 42
Replica Shards 33
Logs
No log data found
Set up Filebeat, then configure your Elasticsearch output to your monitoring cluster.
Kibana
Health is green
Overview
Requests 0
Max. Response Time 0 ms
Instances: 1
Connections 80
Memory Usage 21.68%
315.7 MB / 1.4 GB

3.性能优化思路

提起性能优化的问题，无论是传统关系型数据库，还是NoSQL、NewSQL，反馈归纳都与这些监控指标中的一个或多个密切相关：

1）CPU负载

2）索引吞吐量

3）搜索吞吐量

4）垃圾收集（GC）活动，ES集群有这个指标

5）搜索线程池队列大小

6）慢查询列表

下面是运维工作中常见的几种问题类型:

3.1 ES非活动（检索/写入）状态资源利用率也非常高

此种问题一般线上表现为，即使没有索引/搜索请求，集群的资源利用率也超过预期，该现象是由于集群中的分片数太多，导致平时没有负载的时候监控上看到的集群资源利用率超过预期；

解决办法，在部署一套集群之前， 最重要的准则是，设计合理的的索引分片，分片数太多(一般是根据集群资源负载情况，超过5000个分片，各个node之间的通讯时间明显延时)；

正在运行的集群，遇到这样的问题，想办法减少活跃分片数，即从考虑冷热数据的原则出发，将可以归档的数据进行分离，使用rollover及shrink功能，以降低活跃分片，降低集群node的通讯延时；

不过在7.x，官方是又根据这个问题重构了通讯延迟的问题，我们在搭建中小型的集群的时候，使用7.x版本，一般这个问题很难遇到，而且7.x默认分片数为1，强烈建议ES集群升级到7.x版本，一般升级到最新稳定版。

3.2线程池存在大量rejected

搜索线程池显示“拒绝”计数的持续增加 GET /_cat/thread_pool

该现象由两种原因导致：

1、查询的目标索引太多分片，超过集群中的逻辑CPU核数，这会在搜索线程池中创建排队任务，从而导致搜索任务被拒绝；

2、磁盘I/O速度慢或在某些情况下完全饱和的CPU导致搜索排队。

这种问题，通常解决方法为；

1、创建索引时采用1主分片&1副本模型，比如使用索引模板在创建索引阶段做好设置（7.0及更高版本默认1主1副）；

2、 Elasticsearch 5.1或更高版本支持搜索任务取消，这对于取消显示在任务管理API中慢查询任务非常有用；

相关API命令

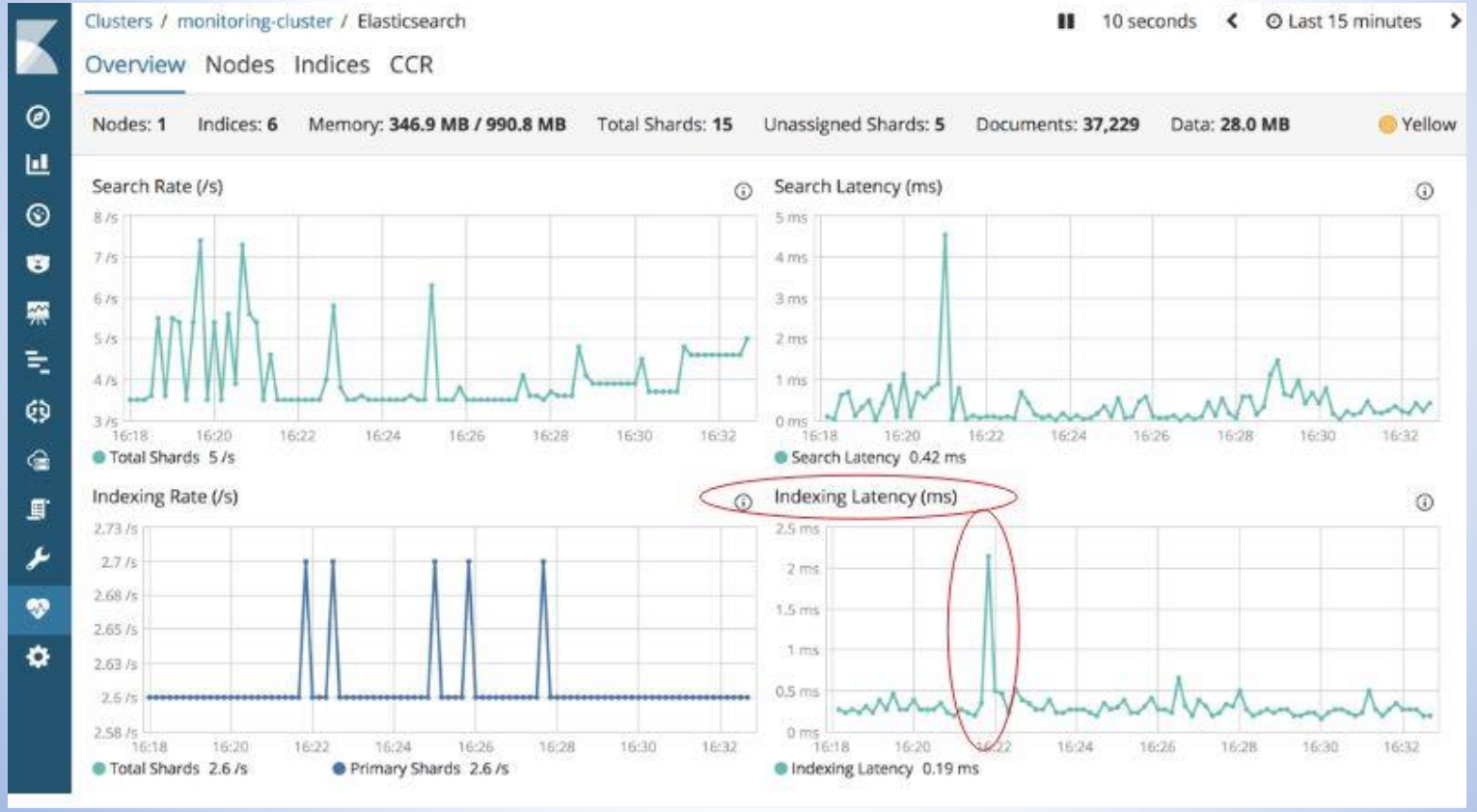
GET _tasks?nodes=nodeId1,nodeId2,....

POST _tasks/oTUltX4IQMOUUVeiohTt8A:27343/_cancel

3、改进磁盘I/O，并确保压测以使用的硬件能够获得最佳性能。

3.3高CPU和索引化延迟

当集群不堪重负时，度量标准关联显示CPU利用率高和索引化延迟大（如下图）。写入数据量大（索引化）会影响搜索性能。


Clusters / monitoring-cluster / Elasticsearch
10 seconds
Last 15 minutes
Overview
Nodes
Indices
CCR
Nodes: 1
Indices: 6
Memory: 346.9 MB / 990.8 MB
Total Shards: 15
Unassigned Shards: 5
Documents: 37,229
Data: 28.0 MB
Yellow
Search Rate (/s)
8 /s
7 /s
6 /s
5 /s
4 /s
3 /s
16:18
16:20
16:22
16:24
16:26
16:28
16:30
16:32
Total Shards 5 /s
Search Latency (ms)
5 ms
4 ms
3 ms
2 ms
1 ms
0 ms
Search Latency 0.42 ms
Indexing Rate (/s)
2.73 /s
2.7 /s
2.68 /s
2.65 /s
2.63 /s
2.6 /s
2.58 /s
Total Shards 2.6 /s
Primary Shards 2.6 /s
Indexing Latency (ms)
2.5 ms
2 ms
1.5 ms
1 ms
0.5 ms
0 ms
Indexing Latency 0.19 ms

解决方法如下

1、调大索引刷新频率

将index.refresh_ interval（文档被索引到数据搜索可见时间间隔）增加到 30 s，通常有助于提高索引性能。项目中要结合具体业务场景，可能会有所不同，因此测试是关键。这样避免了缺省情况下1秒生成一个分段的麻烦。

2、对于负载重的索引，请检查我们的索引调整建议，以优化索引和搜索性能。

包含但不限于：

1）数据初始化阶段refresh设置 -1、副本设置为 0，以提升写入速度；写入完毕后复原

2）关闭swapping

3）使用自动生成ID

3.4副本增加后延时增大

在增加副本分片计数（例如，从1到3）之后可以观察到查询等待时间增加。

如果存在更多数据，则缓存的数据将很快被逐出，导致操作系统层面页面错误增加。文件系统缓存没有足够的内存来缓存经常查询的索引部分。ES查询缓存实现了LRU置换算法：当缓存变满时，最近最少使用的数据被置换以便为新数据腾出空间。

这种问题，一般解决方案如下

1、调整物理内存，一般给文件系统缓存留出至少50%RAM，内存越多，可以缓存的越多，尤其是在集群遇到I / O问题时，堆大小已正确配置，剩下的任何可用于文件系统缓存的剩余物理RAM都可以大大加快搜索性能。

注意，堆内存大小配置建议：Min(31GB，物理机器内存/2)

2、使用query缓存和request缓存加快检索速度

节点级别的query缓存默认是开启的，对应配置：index.queries.cache.enabled

请求缓存默认是开启的，可以动态设置开启
index.requests.cache.enable : true

3、使用preference优化高速缓存

可以使用搜索请求首选项preference来优化所有这些高速缓存。以便每次将某些搜索请求路由到同一组分片，而不是在可用的不同副本之间交替，这将更好地利用请求缓存、节点查询缓存和文件系统缓存。

3.5聚合N多唯一值引起的高内存使用率

查询包含唯一值（例如客户公司ID，用户名，电子邮件地址等）的聚合字段时性能不佳。

在分析堆内存时：Java对象使用"search", "buckets", "aggregation"等术语，消耗大量的堆内存。

聚合在高基数(high-cardinality）字段上运行，需大量资源来获取许多存储桶。

还可以存在涉及nested字段和/或join字段的嵌套聚合。

解决方法如下有效方案

1、提高高基数term聚合的性能，核心：使用eager_global_ordinals: true 提升性能；

2、其他进一步调整，看官网nested字段类型和join字段类型的使用建议，以更好地提高聚合性能；

3.6拆解DSL排查慢查询根源

对于DSL语句慢查询的场景，我们可以尝试逐个删除查询中的功能，并检查查询是否仍然很慢。

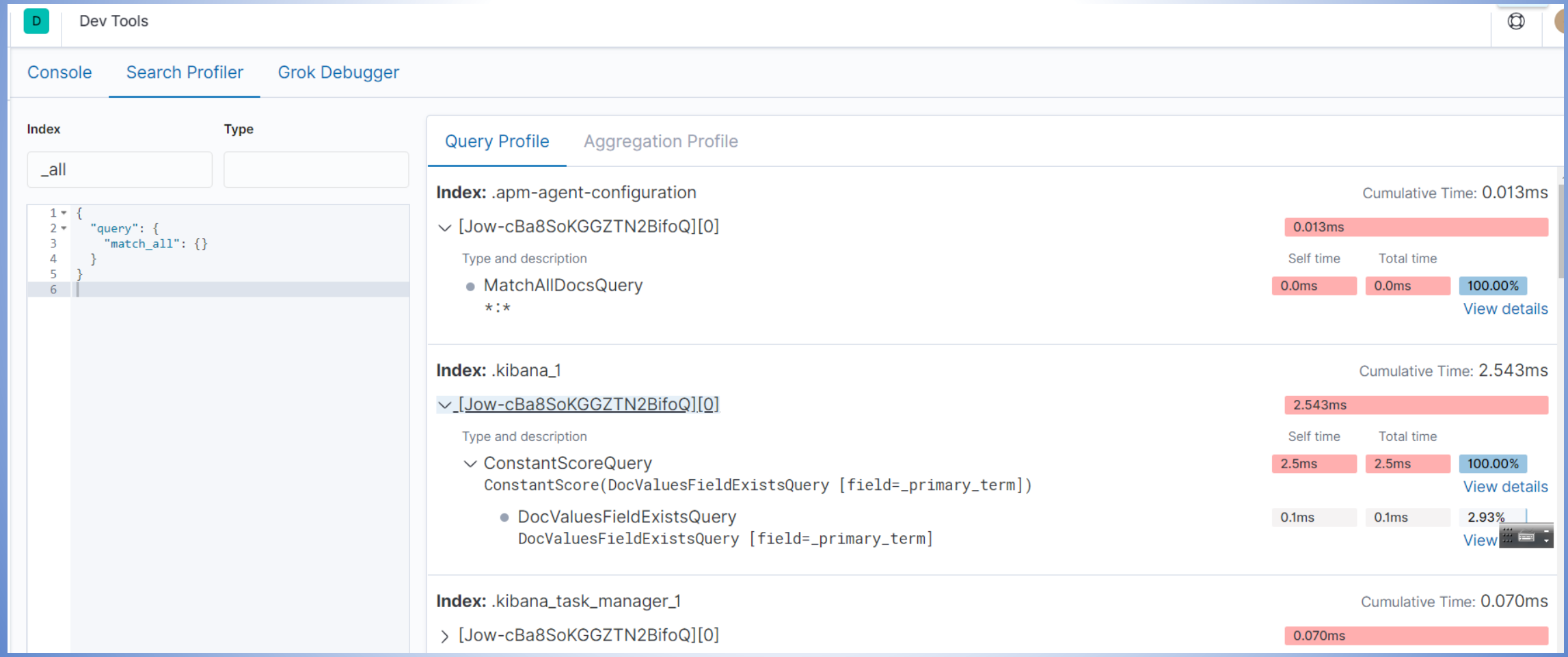
查找最简单查询以重现性能问题有助于隔离分析

1）查询条件较少时候，它仍然很慢吗？

2）没有聚合，它仍然很慢吗？

3）如果size设置为0，它仍然很慢吗？

工具有kibana里面的Search Profile分析执行过程最为方便，如下图，推荐；


D
Dev Tools
Console
Search Profiler
Grok Debugger
Index
Type
_all
1 {
2 "query": {
3 "match_all": {}
4 }
5 }
6
Query Profile
Aggregation Profile
Index: .apm-agent-configuration
Cumulative Time: 0.013ms
[Jow-cBa8SoKGGZTN2BifoQ][0]
0.013ms
Type and description
Self time
Total time
MatchAllDocsQuery
:
0.0ms
0.0ms
100.00%
View details
Index: .kibana_1
Cumulative Time: 2.543ms
[Jow-cBa8SoKGGZTN2BifoQ][0]
2.543ms
Type and description
Self time
Total time
ConstantScoreQuery
ConstantScore(DocValuesFieldExistsQuery [field=_primary_term])
2.5ms
2.5ms
100.00%
View details
DocValuesFieldExistsQuery
DocValuesFieldExistsQuery [field=_primary_term]
0.1ms
0.1ms
2.93%
View
Index: .kibana_task_manager_1
Cumulative Time: 0.070ms
[Jow-cBa8SoKGGZTN2BifoQ][0]
0.070ms

3.7慢日志分析

可以通过配置启用Elasticsearch中的慢日志来获取运行缓慢的查询，

Slowlogs专门用于分片级别，即只应用数据节点

仅协调 Coordinating-only/客户client节点不具备慢日志分析功能，因为它们不保存数据（索引/分片）

Slowlogs会告诉我们：

1）该查询需要多长时间？

2）该查询请求正文的内容是什么？

基于慢日志，进行查询语句的瓶颈分析就方便多了；

Q&A

1. ElasticSearch有哪些天然缺陷？
2. 如果你觉得ES比较好，怎么来给领导或者团队推荐使用基于ES来搭建日志或者搜索平台？
3. 大家怎么看待7X24的运维工作？