elasticsearch 亿级数据检索案例与原理

Java知音 1周前

> 来源：mikevictor
>
> cnblogs.com/mikevictor07/p/10006553.html

一、前言

数据平台已迭代三个版本，从头开始遇到很多常见的难题，终于有片段时间整理一些已完善的文档，在此分享以供所需朋友的

实现参考，少走些弯路，在此篇幅中偏重于ES的优化，关于HBase，Hadoop的设计优化估计有很多文章可以参考，不再赘述。

二、需求说明

项目背景：

在一业务系统中，部分表每天的数据量过亿，已按天分表，但业务上受限于按天查询，并且DB中只能保留3个月的数据(硬件高配)，分库代价较高。

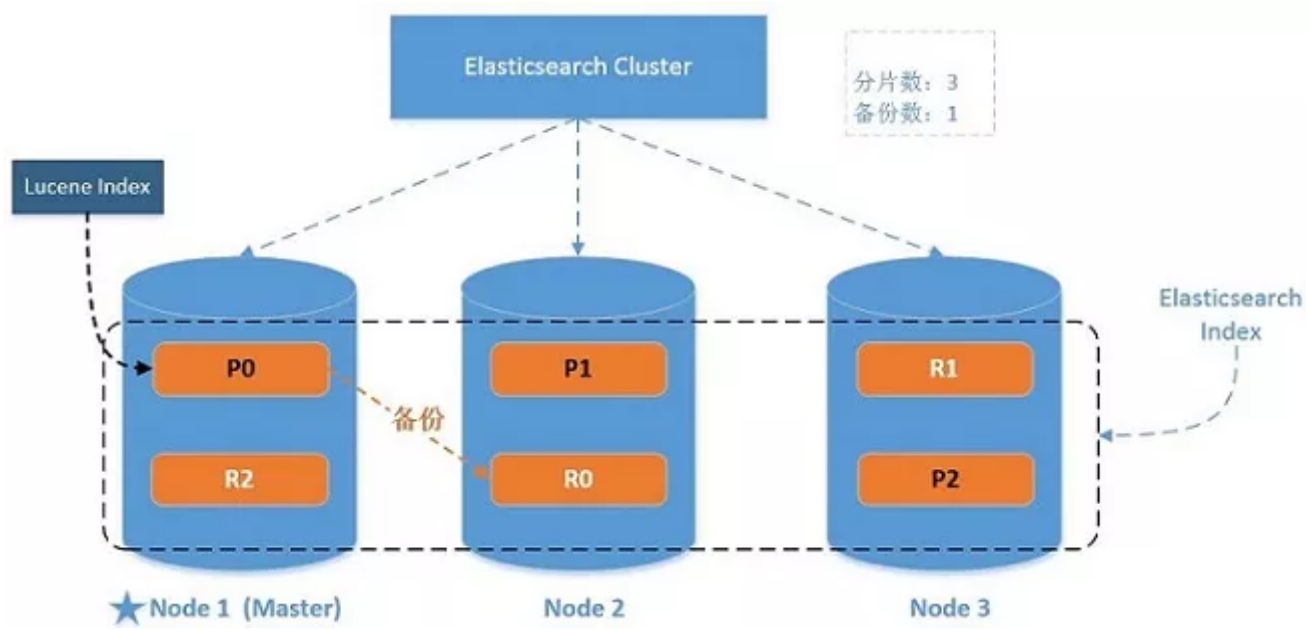
改进版本目标：

1. 数据能跨月查询，并且支持1年以上的历史数据查询与导出。
2. 按条件的数据查询秒级返回。

三、elasticsearch检索原理

3.1 关于ES和Lucene基础结构

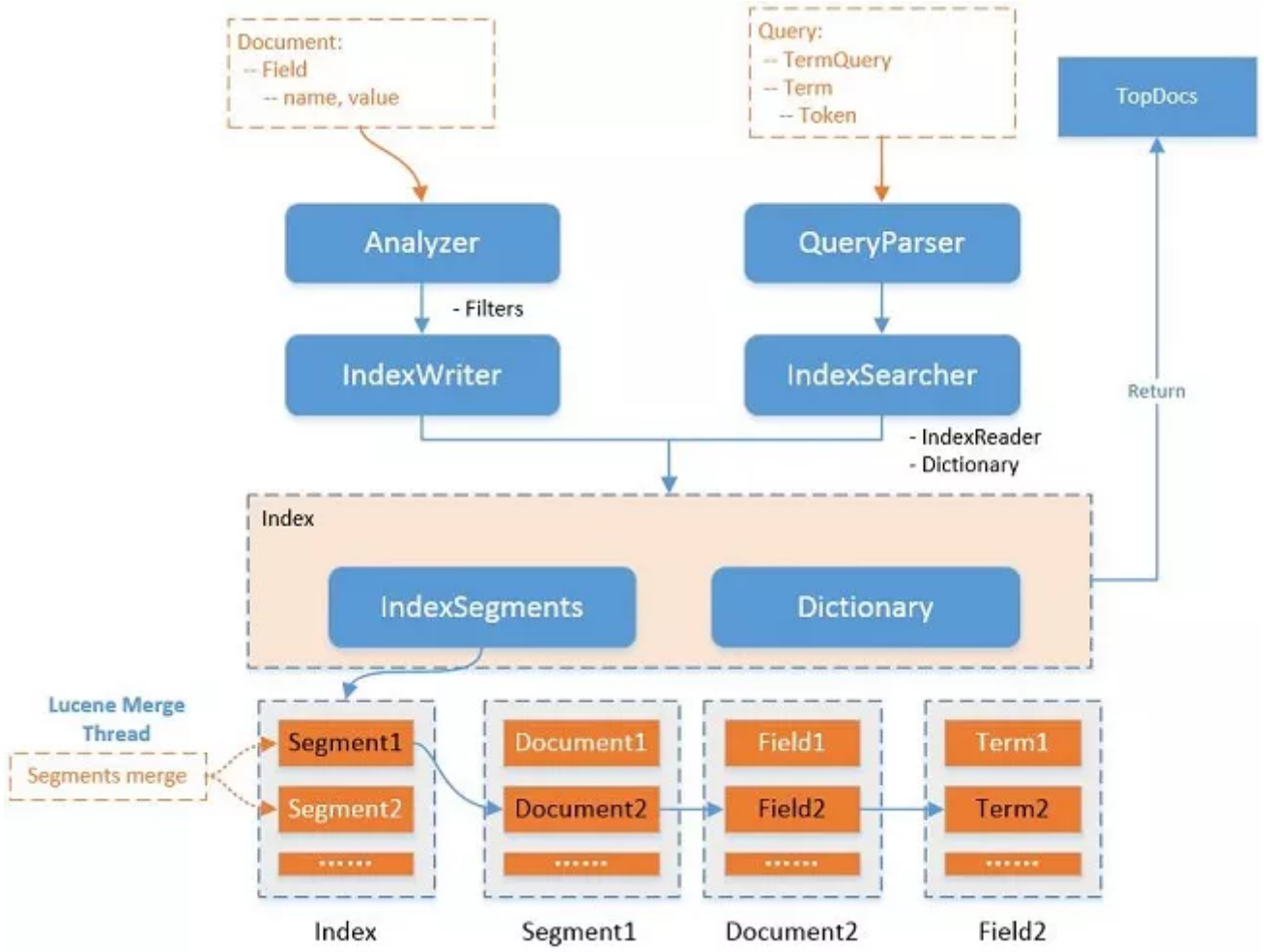
谈到优化必须能了解组件的基本原理，才容易找到瓶颈所在，以免走多种弯路，先从ES的基础结构说起(如下图)：



一些基本概念:

- **Cluster** 包含多个Node的集群
- **Node** 集群服务单元
- **Index** 一个ES索引包含一个或多个物理分片，它只是这些分片的逻辑命名空间
- **Type** 一个index的不同分类，6.x后只能配置一个type，以后将移除
- **Document** 最基础的可被索引的数据单元，如一个JSON串
- **Shards** 一个分片是一个底层的工作单元，它仅保存全部数据中的一部分，它是一个Lucence实例 (一个lucene索引最大包含2,147,483,519 (= Integer.MAX_VALUE - 128)个文档数量)
- **Replicas** 分片备份，用于保障数据安全与分担检索压力

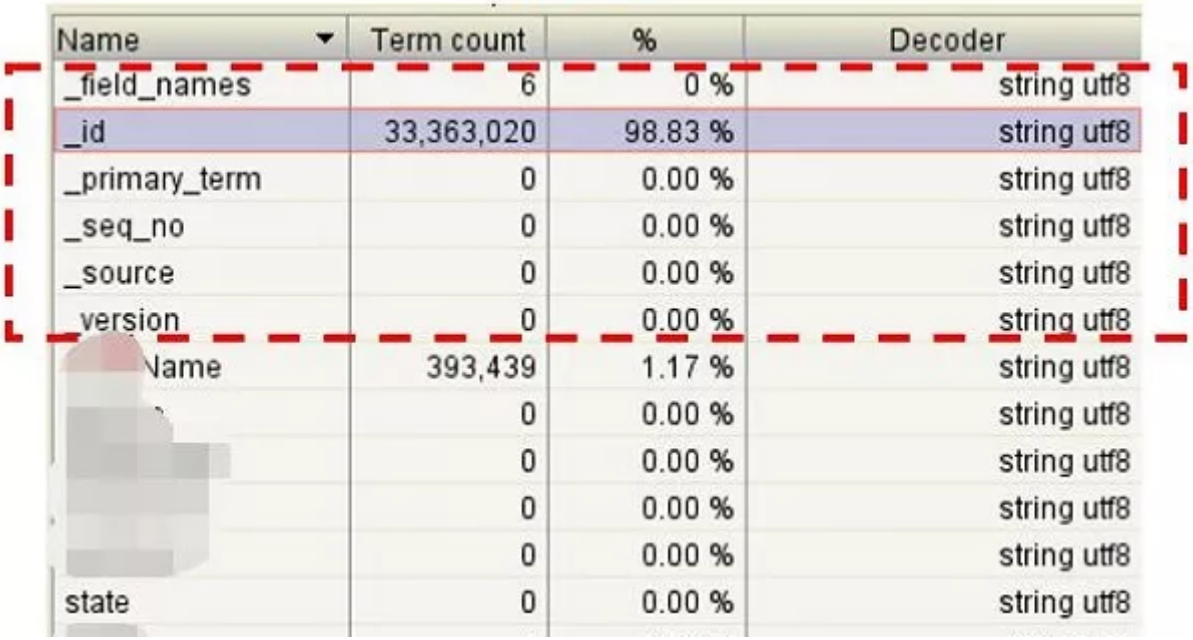
ES依赖一个重要的组件Lucene，关于数据结构的优化通常来说是对Lucene的优化，它是集群的一个存储于检索工作单元，结构如下图：



在Lucene中，分为索引(录入)与检索(查询)两部分，索引部分包含 **分词器**、**过滤器**、**字符映射器** 等，检索部分包含 **查询解析器** 等。

一个Lucene索引包含多个segments，一个segment包含多个文档，每个文档包含多个字段，每个字段经过分词后形成一个或多个term。

通过Luke工具查看ES的lucene文件如下，主要增加了_id和_source字段:

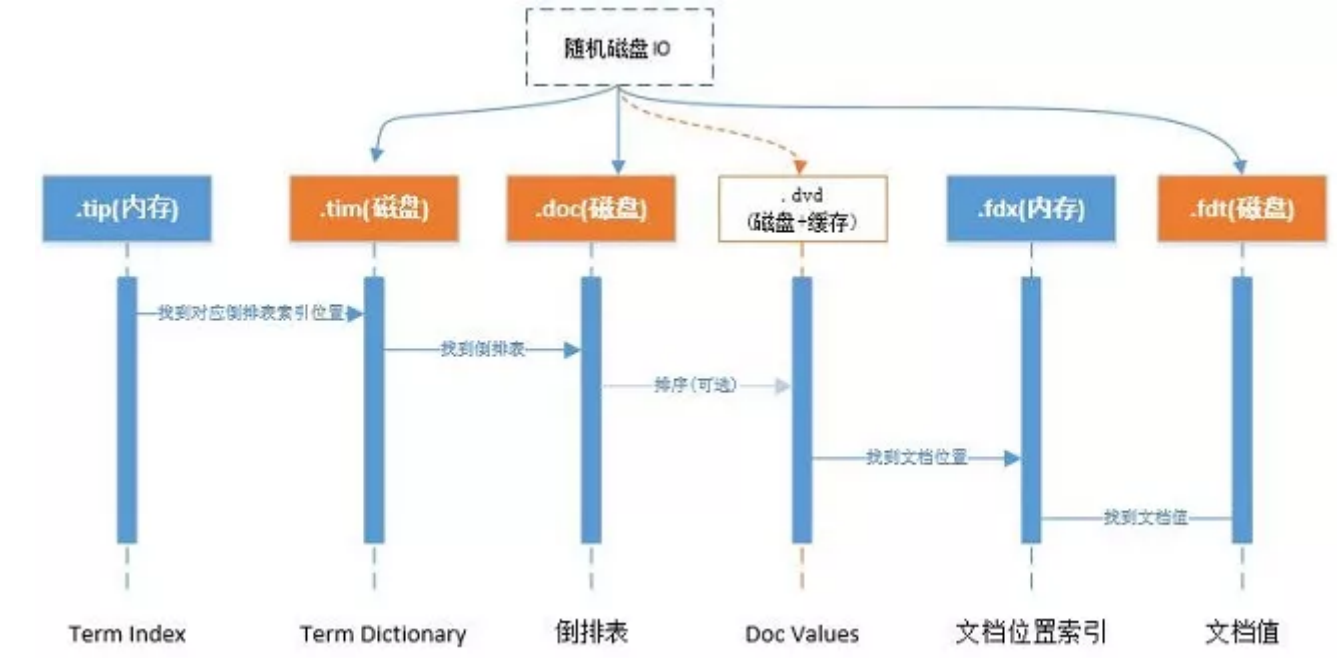


Name	Term count	%	Decoder
_field_names	6	0 %	string utf8
_id	33,363,020	98.83 %	string utf8
_primary_term	0	0.00 %	string utf8
_seq_no	0	0.00 %	string utf8
_source	0	0.00 %	string utf8
_version	0	0.00 %	string utf8
[illegible]Name	393,439	1.17 %	string utf8
[illegible]	0	0.00 %	string utf8
[illegible]	0	0.00 %	string utf8
[illegible]	0	0.00 %	string utf8
[illegible]	0	0.00 %	string utf8
state	0	0.00 %	string utf8

3.2 Lucene索引实现

Lucene 索引文件结构主要的分为：词典、倒排表、正向文件、DocValues等，如下图:

Name	Extension	Description
Term Index	.tip	词典索引（需加载进内存）
Term Dictionary	.tim	倒排表指针
Frequencies	.doc	包含 Term 和频率的文档列表(倒排表)
Fields	.fnm	Field 数据元信息
Field Index	.fdx	文档位置索引（需加载进内存）
Field Data	.fdt	文档值
Per-Document Values	.dvd, .dvm	.dvm 为 DocValues 元信息 .dvd 为 DocValue 值 (默认情况下 elasticsearch 开启该功能用于快速排序、聚合等操作)



注：整理来源于lucene官方:

> http://lucene.apache.org/core/7_2_1/core/org/apache/lucene/codecs/lucene70/package-summary.html#package.description

Lucene 随机三次磁盘读取比较耗时。其中.fdt文件保存数据值损耗空间大，.tim和.doc则需要SSD存储提高随机读写性能。

另外一个比较消耗性能的是打分流程，不需要则可屏蔽。

关于DocValues：

倒排索引解决从词快速检索到相应文档ID, 但如果需要对结果进行排序、分组、聚合等操作的时候则需要根据文档ID快速找到对应的值。

通过倒排索引代价缺很高：需迭代索引里的每个词项并收集文档的列里面 token。这很慢而且难以扩展：随着词项和文档的数量增加，执行时间也会增加。Solr docs对此的解释如下：

> For other features that we now commonly associate with search, such as sorting, faceting, and highlighting, this approach is not very efficient. The faceting engine, for example, must look up each term that appears in each document that will make up the result set and pull the document IDs in order to build the facet list. In Solr, this is maintained in memory, and can be slow to load (depending on the number of documents, terms, etc.)

在lucene 4.0版本前通过FieldCache，原理是通过按列逆转倒排表将（field value ->doc）映射变成（doc -> field value）映射，问题为逐步构建时间长并且消耗大量内存，容易造成OOM。

DocValues是一种列存储结构，能快速通过文档ID找到相关需要排序的字段。在ES中，默认开启所有(除了标记需analyzed的字符串字段)字段的doc values，**如果不需要对此字段做任何排序等工作，则可关闭以减少资源消耗。**

3.3 关于ES索引与检索分片

ES中一个索引由一个或多个lucene索引构成，一个lucene索引由一个或多个segment构成，其中segment是最小的检索域。

数据具体被存储到哪个分片上：

```
shard = hash(routing) % number_of_primary_shards
```

默认情况下 routing参数是文档ID (murmurhash3),可通过 URL中的 _routing 参数指定数据分布在同一个分片中，index和search的时候都需要一致才能找到数据，**如果能明确根据_routing进行数据分区，则可减少分片的检索工作，以提高性能。**

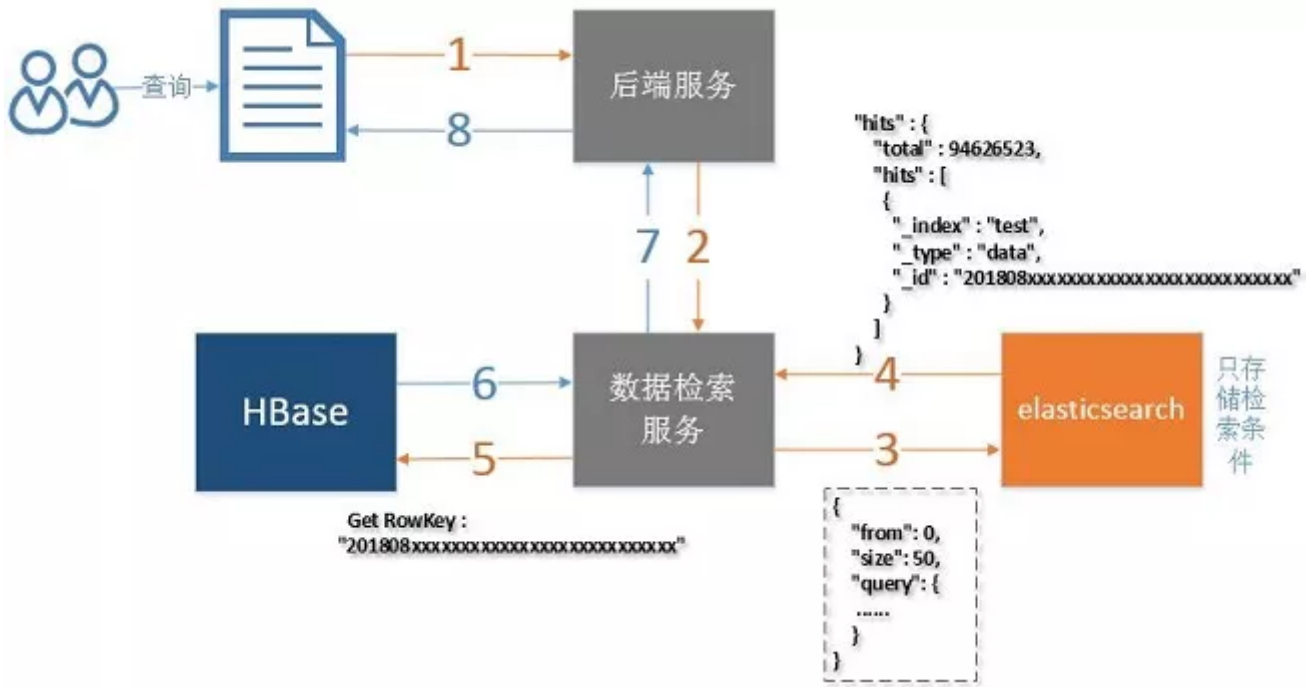
四、优化案例

在我们的案例中，查询字段都是固定的，不提供全文检索功能，这也是几十亿数据能秒级返回的一个大前提：

- ES仅提供字段的检索，仅存储HBase的Rowkey不存储实际数据。
- 实际数据存储在HBase中，通过Rowkey查询，如下图。
- 提高索引与检索的性能建议，可参考官方文档

> https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-indexing-speed.html

一些细节优化项官方与其他的一些文章都有描述，在此文章中仅提出一些本案例的重点优化项。



4.1 优化索引性能

1、**批量写入**，看每条数据量的大小，一般都是几百到几千。

2、**多线程写入**，写入线程数一般和机器数相当，可以配多种情况，在测试环境通过Kibana观察性能曲线。

3、**增加segments的刷新时间**，通过上面的原理知道，segment作为一个最小的检索单元，比如segment有50个，目的需要查10条数据，但需要从50个segment分别查询10条，共500条记录，再进行排序或者分数比较后，截取最前面的10条，丢弃490条。在我们的案例中将此 "refresh_interval": "-1" ，程序批量写入完成后进行手工刷新(调用相应的API即可)。

4、**内存分配方面**，很多文章已经提到，给系统50%的内存给Lucene做文件缓存，它任务很繁重，所以ES节点的内存需要比较多(比如每个节点能配置64G以上最好）。

5、**磁盘方面配置SSD**，机械盘做阵列RAID5 RAID10虽然看上去很快，但是随机IO还是SSD好。

6、 **使用自动生成的ID**，在我们的案例中使用自定义的KEY，也就是与HBase的ROW KEY，是为了能根据rowkey删除和更新数据，性能下降不是很明显。

7、关于**段合并**，合并在后台定期执行，比较大的segment需要很长时间才能完成，为了减少对其他操作的影响(如检索)，elasticsearch进行阈值限制，默认是20MB/s，

可配置的参数："indices.store.throttle.max_bytes_per_sec" : "200mb" （根据磁盘性能调整）

合并线程数默认是：Math.max(1, Math.min(4, Runtime.getRuntime().availableProcessors() / 2))，

如果是机械磁盘，可以考虑设置为1：index.merge.scheduler.max_thread_count: 1，在我们的案例中使用SSD，配置了6个合并线程。

4.2 优化检索性能

1、关闭不需要字段的doc values。

2、尽量使用keyword替代一些long或者int之类，term查询总比range查询好 (参考lucene说明)。

> http://lucene.apache.org/core/7_4_0/core/org/apache/lucene/index/PointValues.html

3、关闭不需要查询字段的_source功能，不将此存储仅ES中，以节省磁盘空间。

4、评分消耗资源，如果不需要可使用filter过滤来达到关闭评分功能，score则为0，如果使用constantScoreQuery则score为1。

5、关于分页：

- from + size: 每分片检索结果数最大为 from + size，假设from = 20, size = 20，则每个分片需要获取20 * 20 = 400条数据，多个分片的结果在协调节点合并(假设请求的分配数为5，则结果数最大为 400*5 = 2000条) 再在内存中排序后然后20条给用户。这种机制导致越往后分页获取的代价越高，达到50000条将面临沉重的代价，默认from + size默认如下：

  ```
  index.max_result_window : 10000
  ```

- search_after: 使用前一个分页记录的最后一条来检索下一个分页记录，在我们的案例中，首先使用from+size，检索出结果后再使用search_after，在页面上我们限制了用户只能跳5页，不能跳到最后一页。
- scroll: 用于大结果集查询，缺陷是需要维护scroll_id

6、关于排序：我们增加一个long字段，它用于存储时间和ID的组合(通过移位即可)，正排与倒排性能相差不明显。

7、关于CPU消耗，检索时如果需要做排序则需要字段对比，消耗CPU比较大，如果有可能尽量分配16cores以上的CPU，具体看业务压力。

8、关于合并被标记删除的记录，我们设置为0表示在合并的时候一定删除被标记的记录，默认应该是大于10%才删除："merge.policy.expunge_deletes_allowed": "0"。

```
{
  "mappings": {
    "data": {
      "dynamic": "false",
      "_source": {
        "includes": ["XXX"] -- 仅将查询结果所需的数据存储仅_source中
      },
      "properties": {
        "state": {
          "type": "keyword",  -- 虽然state为int值，但如果不需要做范围查询，尽量使用keyword，因为int需要比keyword增加额外的消耗。
          "doc_values": false -- 关闭不需要字段的doc values功能，仅对需要排序，汇聚功能的字段开启。
        },
        "b": {
          "type": "long"   -- 使用了范围查询字段，则需要用long或者int之类 （构建类似KD-trees结构）
        }
      }
    }
  },
  "settings": {......}
}
```

五、性能测试

优化效果评估基于基准测试，如果没有基准测试无法了解是否有性能提升，在这所有的变动前做一次测试会比较好。在我们的案例中：

- 单节点5千万到一亿的数据量测试，检查单点承受能力。
- 集群测试1亿-30亿的数量，磁盘IO/内存/CPU/网络IO消耗如何。
- 随机不同组合条件的检索，在各个数据量情况下表现如何。
- 另外SSD与机械盘在测试中性能差距如何。

性能的测试组合有很多，通常也很花时间，不过作为评测标准时间上的投入有必要，否则生产出现性能问题很难定位或不好改善。对于ES的性能研究花了不少时间，最多的关注点就是lucene的优化，能深入了解lucene原理对优化有很大的帮助。

六、生产效果

目前平台稳定运行，几十亿的数据查询100条都在3秒内返回，前后翻页很快，如果后续有性能瓶颈，可通过扩展节点分担数据压力。

推荐阅读(点击即可跳转阅读)

1. SpringBoot内容聚合
2. 面试题内容聚合
3. 设计模式内容聚合
4. Mybatis内容聚合
5. 多线程内容聚合

觉得不错？欢迎转发分享给更多人

Java知音

微信扫描二维码，关注我的公众号

我知道你 “在看”