重要的集群配置与优化方案

> https://www.elastic.co/guide/en/elasticsearch/reference/current/important-settings.html
> https://www.elastic.co/guide/en/elasticsearch/reference/current/bootstrap-checks.html
> https://cloud.tencent.com/developer/article/1156231
> https://segmentfault.com/a/1190000008796167

Linux优化：

1. 关闭交换分区，防止内存置换降低性能。 将/etc/fstab 文件中包含swap的行注释掉

```
sed -i '/swap/s/^/#/' /etc/fstab
swapoff -a

#或者在es配置文件中设置
bootstrap.memory_local: true
```

2. 磁盘挂载选项

```
noatime：禁止记录访问时间戳，提高文件系统读写性能
data=writeback： 不记录data journal，提高文件系统写入性能
barrier=0：barrier保证journal先于data刷到磁盘，上面关闭了journal，这里的barrier也就没必要开启了
nobh：关闭buffer_head，防止内核打断大块数据的IO操作
```

3.对于SSD磁盘，采用电梯调度算法，因为SSD提供了更智能的请求调度算法，不需要内核去做多余的调整 (仅供参考)

```
echo noop > /sys/block/sda/queue/scheduler
```

ES集群优化：

1. 适当增大写入buffer和bulk队列长度，提高写入性能和稳定性

```
indices.memory.index_buffer_size: 15%
thread_pool.bulk.queue_size: 1024
```

2. 计算disk使用量时，不考虑正在搬迁的shard

在规模比较大的集群中，可以防止新建shard时扫描所有shard的元数据，提升shard分配速度。

```
cluster.routing.allocation.disk.include_relocations: false
```

3. 禁用delete all

```
action.destructive_requires_name:true
```

index优化：

1. 开启最佳压缩

```
PUT /my_index/_settings
{
 "index.codec": "best_compression"
}
```

2. bulk批量写入

写入数据时尽量使用下面的bulk接口批量写入，提高写入效率。每个bulk请求的doc数量设定区间推荐为1k~1w，具体可根据业务场景选取一个适当的数量。

3. 调整translog同步策略

默认情况下，translog的持久化策略是，对于每个写入请求都做一次flush，刷新translog数据到磁盘上。这种频繁的磁盘IO操作是严重影响写入性能的，如果可以接受一定概率的数据丢失（这种硬件故障的概率很小），可以通过下面的命令调整 translog 持久化策略为异步周期性执行，并适当调整translog的刷盘周期。

```
PUT my_index
{
 "settings": {
 "index": {
 "translog": {
 "sync_interval": "5s",
 "durability": "async"
 }
 }
  }
}
```

4. 调整refresh_interval

写入Lucene的数据，并不是实时可搜索的，ES必须通过refresh的过程把内存中的数据转换成Lucene的完整segment后，才可以被搜索。默认情况下，ES每一秒会refresh一次，产生一个新的segment，这样会导致产生的segment较多，从而segment merge较为频繁，系统开销较大。如果对数据的实时可见性要求较低，可以通过下面的命令提高refresh的时间间隔，降低系统开销.

```
PUT my_index
{
```

```
  "settings": {
  "index": {
  "refresh_interval" : "30s"
  }
  }
}
```

5. merge并发控制

ES的一个index由多个shard组成，而一个shard其实就是一个Lucene的index，它又由多个segment组成，且Lucene会不断地把一些小的segment合并成一个大的segment，这个过程被称为merge。默认值是Math.max(1, Math.min(4, Runtime.getRuntime().availableProcessors() / 2))，当节点配置的cpu核数较高时，merge占用的资源可能会偏高，影响集群的性能，可以通过下面的命令调整某个index的merge过程的并发度

```
PUT /my_index/_settings
{
  "index.merge.scheduler.max_thread_count": 2
}
```

6. 使用routing

对于数据量较大的index，一般会配置多个shard来分摊压力。这种场景下，一个查询会同时搜索所有的shard，然后再将各个shard的结果合并后，返回给用户。对于高并发的小查询场景，每个分片通常仅抓取极少量数据，此时查询过程中的调度开销远大于实际读取数据的开销，且查询速度取决于最慢的一个分片。开启routing功能后，ES会将routing相同的数据写入到同一个分片中（也可以是多个，由index.routing_partition_size参数控制）。如果查询时指定routing，那么ES只会查询routing指向的那个分片，可显著降低调度开销，提升查询效率。

```
# 写入
PUT my_index/my_type/1?routing=user1
{
  "title": "This is a document"
}

# 查询
GET my_index/_search?routing=user1,user2
{
  "query": {
```

```
  "match": {
  "title": "document"
  }
  }
}
```

7. 为string类型的字段选取合适的存储方式

- 存为text类型的字段（string字段默认类型为text）： 做分词后存储倒排索引，支持全文检索，可以通过下面几个参数优化其存储方式:

  norms：用于在搜索时计算该doc的_score（代表这条数据与搜索条件的相关度），如果不需要评分，可以将其关闭。

  index_options：控制倒排索引中包括哪些信息（docs、freqs、positions、offsets）。对于不太注重_score/highlighting的使用场景，可以设为 docs来降低内存/磁盘资源消耗。

  fields: 用于添加子字段。对于有sort和聚合查询需求的场景，可以添加一个keyword子字段以支持这两种功能。

- 存为keyword类型的字段： 不做分词，不支持全文检索。text分词消耗CPU资源，冗余存储keyword子字段占用存储空间。如果没有全文索引需求，只是要通过整个字段做搜索，可以设置该字段的类型为keyword，提升写入速率，降低存储成本。设置字段类型的方法有两种：一是创建一个具体的index时，指定字段的类型；二是通过创建template，控制某一类index的字段类型。

8. 查询时，使用query-bool-filter组合取代普通query

默认情况下，ES通过一定的算法计算返回的每条数据与查询语句的相关度，并通过_score字段来表征。但对于非全文索引的使用场景，用户并不care查询结果与查询条件的相关度，只是想精确的查找目标数据。此时，可以通过query-bool-filter组合来让ES不计算_score，并且尽可能的缓存filter的结果集，供后续包含相同filter的查询使用，提高查询效率。

9. index建立管理

以日期的形式滚动建立，通过设置别名来切换索引，删除过期索引。

10. 控制分片数与副本数

shard数量（number_of_shards）设置过多或过低都会引发一些问题：shard数量过多，则批量写入/查询请求被分割为过多的子写入/查询，导致该index的写入、查询拒绝率上升；对

于数据量较大的inex，当其shard数量过小时，无法充分利用节点资源，造成机器资源利用率不高 或 不均衡，影响写入/查询的效率。

根据预估索引的大小保证每个shard的大小在20～50GB,document总数小于20亿，如果shard数量（不包括副本）超过50个，就很可能引发拒绝率上升的问题，此时可考虑把该index拆分为多个独立的index，分摊数据量，同时配合routing使用，降低每个查询需要访问的shard数量。

11. Segment Memory优化

前面提到，ES底层采用Lucene做存储，而Lucene的一个index又由若干segment组成，每个segment都会建立自己的倒排索引用于数据查询。Lucene为了加速查询，为每个segment的倒排做了一层前缀索引，这个索引在Lucene4.0以后采用的数据结构是FST (Finite State Transducer)。Lucene加载segment的时候将其全量装载到内存中，加快查询速度。这部分内存被称为SegmentMemory，常驻内存，占用heap，无法被GC。

前面提到，为利用JVM的对象指针压缩技术来节约内存，通常建议JVM内存分配不要超过32G。当集群的数据量过大时，SegmentMemory会吃掉大量的堆内存，而JVM内存空间又有限，此时就需要想办法降低SegmentMemory的使用量了，常用方法有下面几个：

- 定期删除不使用的index
- 对于不常访问的index，可以通过close接口将其关闭，用到时再打开
- 通过forcemerge接口强制合并segment，降低segment数量

forcemerge:https://www.elastic.co/guide/en/elasticsearch/reference/6.5/indices-forcemerge.html

12. 分场景优化

https://elasticsearch.cn/article/6191