2015年12月13日 星期日

如何做Elasticsearch的优化

主题

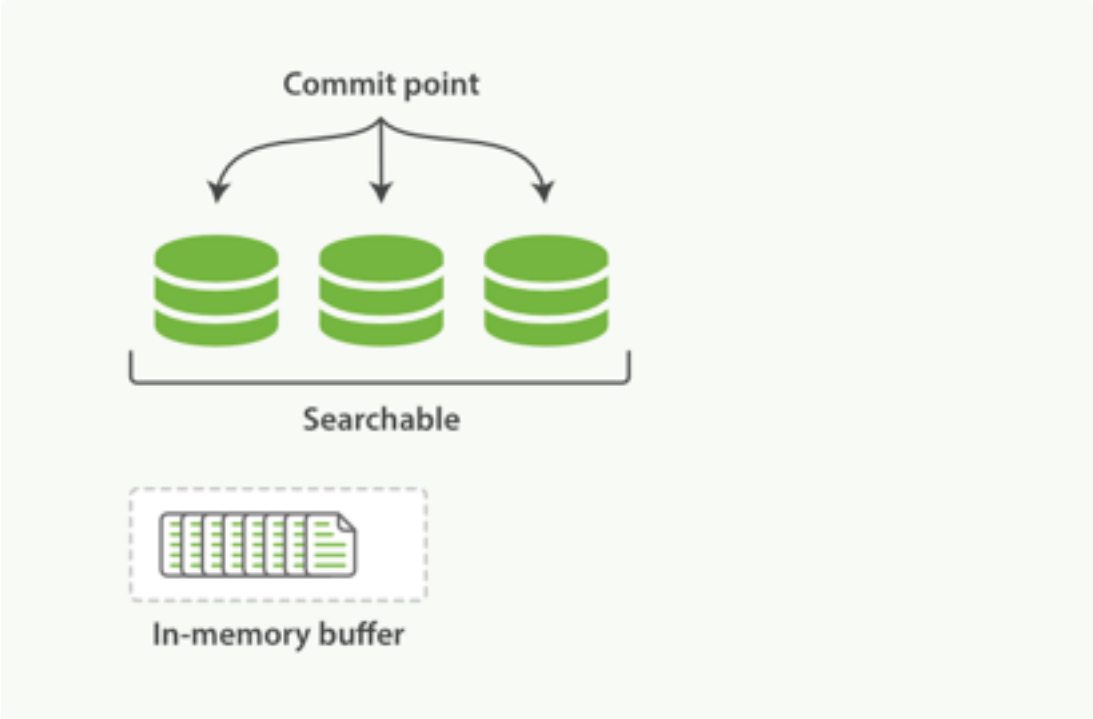
- segment、buffer和translog对实时性的影响
    - 动态更新的Lucene索引
    - 利用磁盘缓存实现的准实时检索
    - translog提供的磁盘同步控制
- segment merge对写入性能的影响
    - 归并线程配置
    - 归并策略
    - optimize接口
- routing和replica的读写过程
    - 路由计算
    - 副本一致性
- shard的allocate控制
    - reroute 接口
    - 冷热数据读写分离

动态更新的Lucene索引

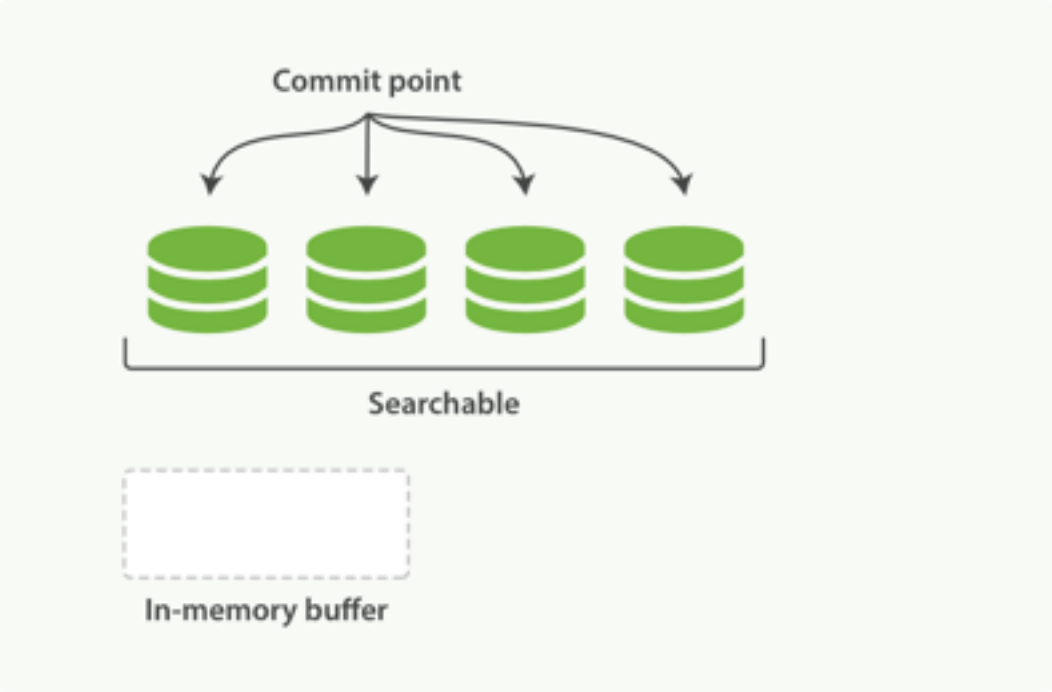
Lucene 把每次生成的倒排索引，叫做一个段(segment)。然后另外使用一个 commit 文件，记录索引内所有的 segment。而生成 segment 的数据来源，则是内存中的 buffer。也就是说，动态更新过程如下：

1 当前索引有 3 个 segment 可用

2 新接收的数据进入内存 buffer



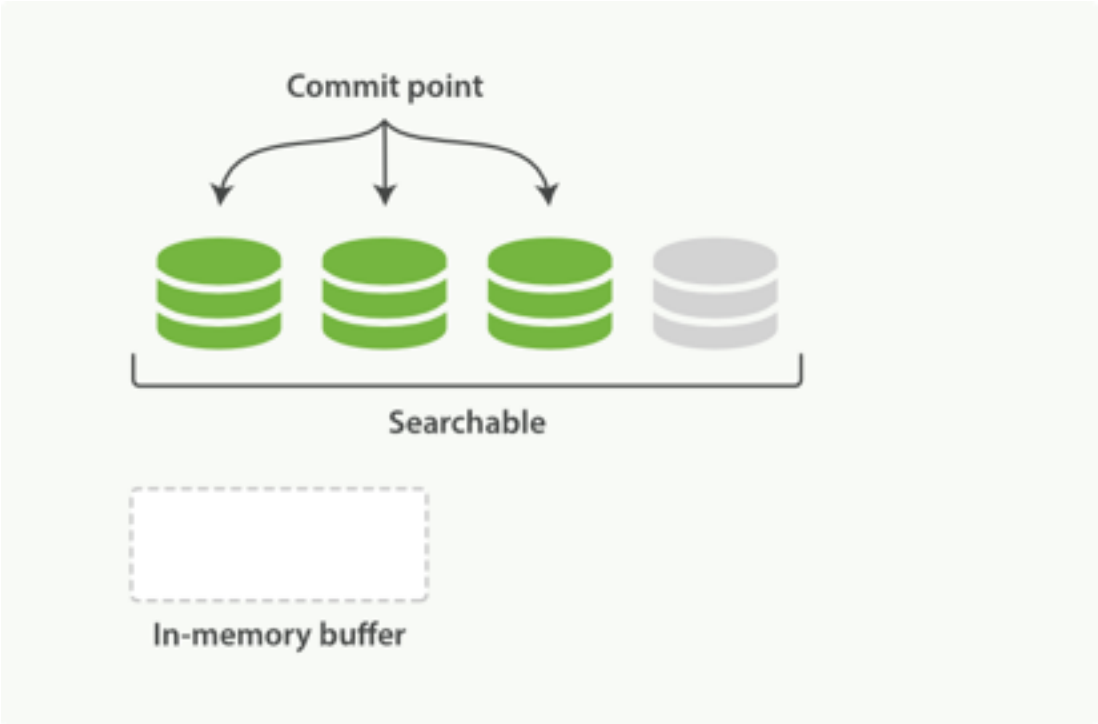
3 内存 buffer 刷到磁盘，生成一个新的 segment，commit 文件同步更新



利用磁盘缓存实现的准实时检索

既然涉及到磁盘，那么一个不可避免的问题就来了：磁盘太慢了！对我们要求实时性很高的服务来说，这种处理还不够。所以，在第 3 步的处理中，还有一个中间状态：

1 内存 buffer 生成一个新的 segment，刷到文件系统缓存中，Lucene 即可检索这个新 segment



2　文件系统缓存真正同步到磁盘上，commit 文件更新

这一步刷到文件系统缓存的步骤，在 ES 中，是默认设置为 1 秒间隔的，对于大多数应用来说，几乎就相当于是实时可搜索了。ES 也提供了单独的 /_refresh 接口，用户如果对 1 秒间隔还不满意的，可以主动调用该接口来保证搜索可见。

不过对于日志系统来说，恰恰相反，我们并不需要如此高的实时性，而是需要更快的写入性能。所以，一般来说，我们反而会通过 /_settings 接口或者定制 template 的方式，加大 refresh_interval 参数：

```
# curl -XPOST http://127.0.0.1:9200/logstash-2015.06.21/_settings -d'
        { "refresh_interval": "10s" }
'
```

如果是导入历史数据的场合，那甚至可以先完全关闭掉：

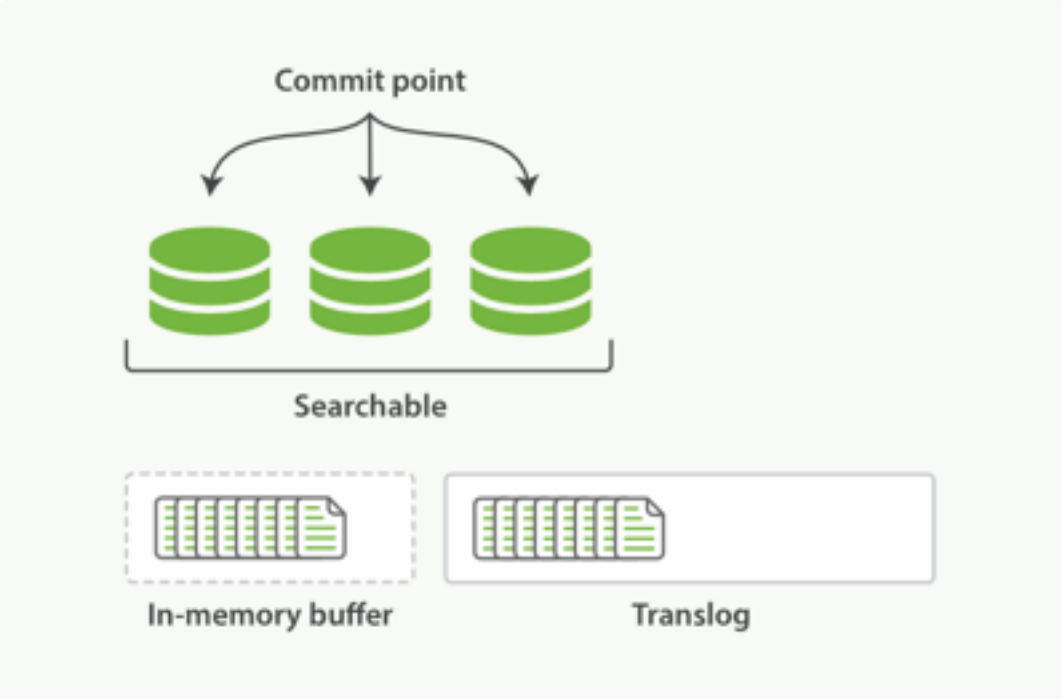
```
# curl -XPUT http://127.0.0.1:9200/logstash-2015.05.01 -d'
{
        "settings" : {
                "refresh_interval": "-1"
        }
}'
```

在导入完成以后，修改回来或者手动调用一次即可：

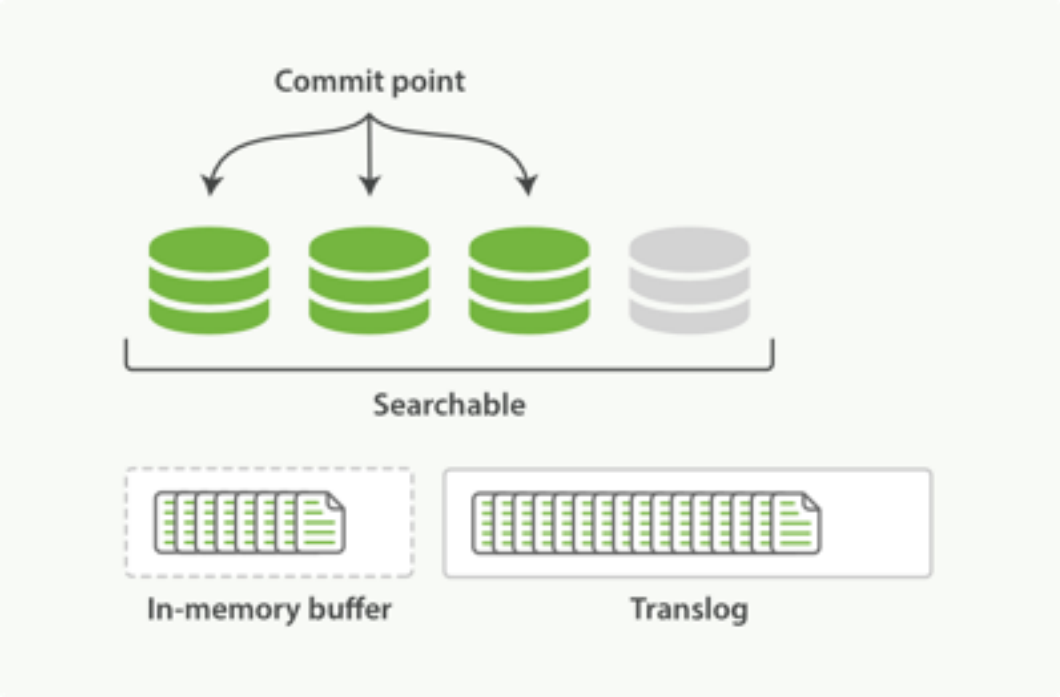
```
# curl -XPOST http://127.0.0.1:9200/logstash-2015.05.01/_refresh
```

translog提供的磁盘同步控制

既然 refresh 只是写到文件系统缓存，那么第 4 步写到实际磁盘又是有什么来控制的？如果这期间发生主机错误、硬件故障等异常情况，数据会不会丢失？
这里，其实有另一个机制来控制。ES 在把数据写入到内存 buffer 的同时，其实还另外记录了一个 translog 日志。也就是说，第 2 步并不是图 2 的状态，而是像图 5 这样：



在第3和第4步，refresh 发生的时候ranslog 日志文件依然保持原样，如图 6：



也就是说，如果在这期间发生异常，ES 会从 commit 位置开始，恢复整个 translog 文件中的记录，保证数据一致性。

等到真正把 segment 刷到磁盘，且 commit 文件进行更新的时候， translog 文件才清空。这一步，叫做 flush。同样，ES 也提供了 /_flush 接口。

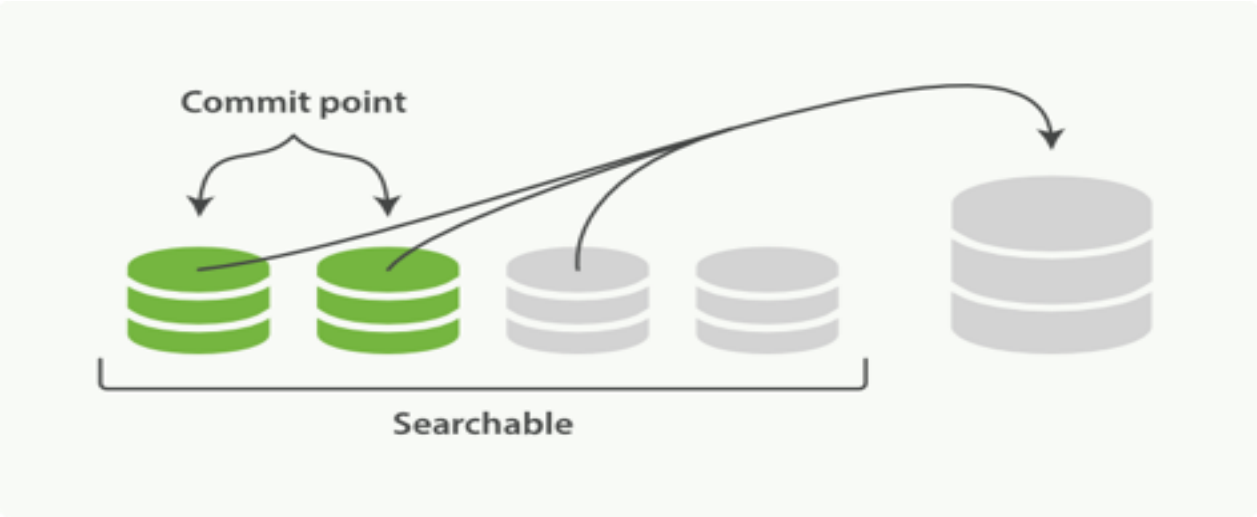
对于 flush 操作，ES 默认设置为：每 30 分钟主动进行一次 flush，或者当 translog 文件大小大于 512MB (老版本是 200MB)时，主动进行一次 flush。这两个行为，可以分别通过 index.translog.flush_threshold_period 和 index.translog.flush_threshold_size 参数修改。

如果对这两种控制方式都不满意，ES 还可以通过 index.translog.flush_threshold_ops 参数，控制每收到多少条数据后 flush 一次。

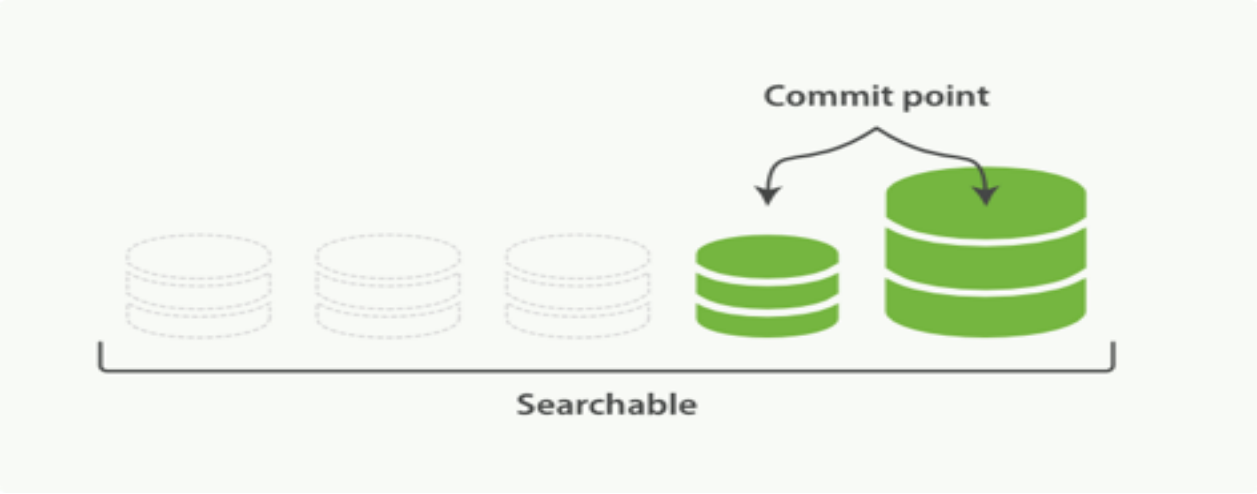
segment merge对写入性能的影响

数据怎么进入 ES 并且如何才能让数据更快的被检索使用。其中用一句话概括了 Lucene 的设计思路就是"开新文件"。从另一个方面看，开新文件也会给服务器带来负载压力。因为默认每 1 秒，都会有一个新文件产生，每个文件都需要有文件句柄，内存，CPU 使用等各种资源。一天有 86400 秒，设想一下，每次请求要扫描一遍 86400 个文件，这个响应性能绝对好不了！

为了解决这个问题，ES 会不断在后台运行任务，主动将这些零散的 segment 做数据归并，尽量让索引内只保有少量的，每个都比较大的，segment 文件。这个过程是有独立的线程来进行的，并不影响新 segment 的产生。归并过程中，索引状态如图7，尚未完成的较大的 segment 是被排除在检索可见范围之外的：



当归并完成，较大的这个 segment 刷到磁盘后，commit 文件做出相应变更，删除之前几个小 segment，改成新的大 segment。等检索请求都从小 segment 转到大 segment 上以后，删除没用的小 segment。这时候，索引里 segment 数量就下降了，状态如图8 所示：



归并线程配置

segment 归并的过程，需要先读取 segment，归并计算，再写一遍 segment，最后还要保证刷到磁盘。可以说，这是一个非常消耗磁盘 IO 和 CPU 的任务。所以，ES 提供了对归并线程的限速机制，确保这个任务不会过分影响到其他任务。
默认情况下，归并线程的限速配置 indices.store.throttle.max_bytes_per_sec 是 20MB。对于写入量较大，磁盘转速较高，甚至使用 SSD 盘的服务器来说，这个限速是明显过低的。对于日志统计分析应用，建议可以适当调大到 100MB或者更高。

```
# curl -XPUT http://127.0.0.1:9200/_cluster/settings -d'
{
   "persistent" : {
      "indices.store.throttle.max_bytes_per_sec" : "100mb"
   }
}'
```

归并线程的数目，ES 也是有所控制的。默认数目的计算公式是： Math.min(3, Runtime.getRuntime().availableProcessors() / 2)。即服务器 CPU 核数的一半大于 3 时，启动 3 个归并线程；否则启动跟 CPU 核数的一半相等的线程数。相信一般做日志系统的服务器 CPU 合数都会在 6 个以上。所以一般来说就是 3 个归并线程。如果你确定自己磁盘性能跟不上，可以降低 index.merge.scheduler.max_thread_count 配置，免得 IO 情况更加恶化。

归并策略

归并线程是按照一定的运行策略来挑选 segment 进行归并的。主要有以下几条：

- index.merge.policy.floor_segment 默认 2MB，小于这个大小的 segment，优先被归并。
- index.merge.policy.max_merge_at_once 默认一次最多归并 10 个 segment
- index.merge.policy.max_merge_at_once_explicit 默认 optimize 时一次最多归并 30 个 segment。
- index.merge.policy.max_merged_segment 默认 5 GB，大于这个大小的 segment，不用参与归并。optimize 除外。

根据这段策略，其实我们也可以从另一个角度考虑如何减少 segment 归并的消耗以及提高响应的办法：加大 flush 间隔，尽量让每次新生成的 segment 本身大小就比较大。

optimize 接口

既然默认的最大 segment 大小是 5GB。那么一个比较庞大的数据索引，就必然会有为数不少的 segment 永远存在，这对文件句柄，内存等资源都是极大的浪费。但是由于归并任务太消耗资源，所以一般不太选择加大 index.merge.policy.max_merged_segment 配置，而是在负载较低的时间段，通过 optimize 接口，强制归并 segment。

```
# curl -XPOST http://127.0.0.1:9200/logstash-2015-06.10/_optimize?max_num_segments=1
```

由于 optimize 线程对资源的消耗比普通的归并线程大得多，所以，绝对不建议对还在写入数据的热索引执行这个操作。

routing和replica的读写过程

路由计算

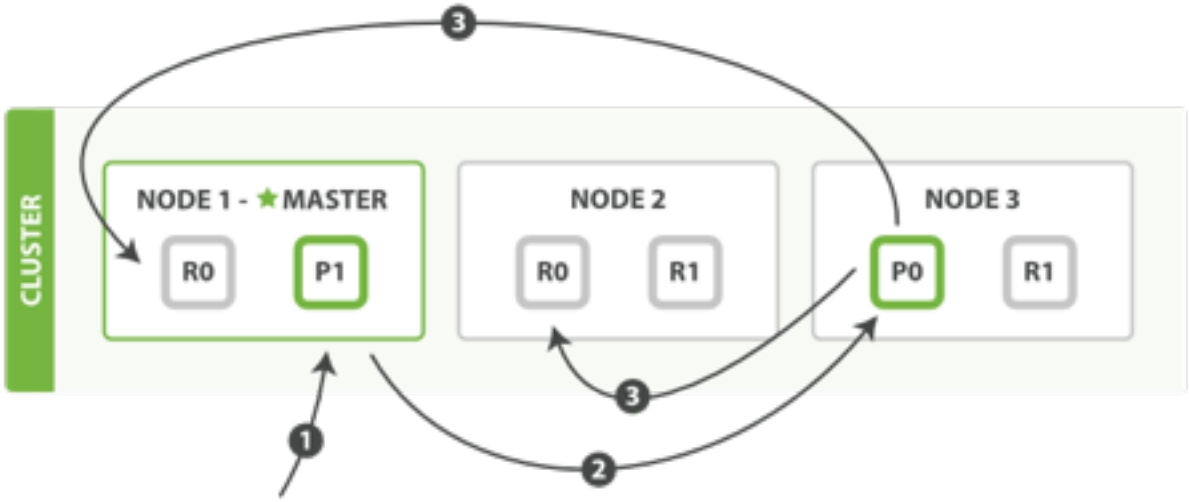
作为一个没有额外依赖的简单的分布式方案，ES 在这个问题上同样选择了一个非常简洁的处理方式，对任一条数据计算其对应分片的方式如下：

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

每个数据都有一个 routing 参数，默认情况下，就使用其 _id 值。将其 _id 值计算哈希后，对索引的主分片数取余，就是数据实际应该存储到的分片 ID。
由于取余这个计算，完全依赖于分母，所以导致 ES 索引有一个限制，索引的主分片数，不可以随意修改。因为一旦主分片数不一样，所以数据的存储位置计算结果都会发生改变，索引数据就完全不可读了。

副本一致性

作为分布式系统，数据副本可算是一个标配。ES 数据写入流程，自然也涉及到副本。在有副本配置的情况下，数据从发向 ES 节点，到接到 ES 节点响应返回，流向如下：
1、客户端请求发送给 Node1节点，注意图中 Node 1 是 Master 节点，实际完全可以不是。
2、Node 1 用数据的 _id 取余计算得到应该讲数据存储到 shard 0 上。通过 cluster state 信息发现 shard 0 的主分片已经分配到了 Node 3 上。Node 1 转发请求数据给 Node 3。
3、Node 3 完成请求数据的索引过程，存入主分片 0。然后并行转发数据给分配有 shard 0 的副本分片的 Node 1 和 Node 2。当收到任一节点汇报副本分片数据写入成功，Node 3 即返回给初始的接收节点 Node 1，宣布数据写入成功。Node 1 返回成功响应给客户端。



这个过程中，有几个参数可以用来控制或变更其行为：

- replication 通过在客户端发送请求的 URL 中加上 ?replication=async ，可以控制 Node 3 在完成本机主分片写入后，就返回给 Node 1 宣布写入成功。这个参数看似可以提高 ES 接收数据写入的性能，但事实上，由于 ES 的副本数据写入也是要经

过完成索引过程的，一旦由于发送过多数据，主机负载偏高导致某块数据写入有异常，可能整个主机的 CPU 都会飙高，导致分配到这台主机上其他主分片的数据都无法高性能完成，最终反而拖累了整体的写入性能。 从 ES 1.6 版本开始，该参数已经被标记为废弃，2.0 版预计将正式删除该参数。

- consistency 上面示例中，2 个副本分片只要有 1 个成功，就可以返回给客户端了。这点也是有配置项的。其默认值的计算来源如下：

  int((primary + number_of_replicas) / 2) + 1

  根据需要，也可以将参数设置为 one，表示仅写完主分片就返回，等同于 async；还可以设置为 all，表示等所有副本分片都写完才能返回。
- timeout 如果集群出现异常，有些分片当前不可用，ES 默认会等待 1 分钟看分片能否恢复。可以使用 ?timeout=30s 参数来缩短这个等待时间。

副本配置和分片配置不一样，是可以随时调整的。有些较大的索引，甚至可以在做 optimize 前，先把副本全部取消掉，等 optimize 完后，再重新开启副本，节约单个 segment 的重复归并消耗。

```
# curl -XPUT http://127.0.0.1:9200/logstash-mweibo-2015.05.02/_settings -d '{
   "index": { "number_of_replicas" : 0 }
}'
```

冷热数据的读写分离

Elasticsearch 集群一个比较突出的问题是: 用户做一次大的查询的时候, 非常大量的读 IO 以及聚合计算导致机器 Load 升高, CPU 使用率上升, 会影响阻塞到新数据的写入, 这个过程甚至会持续几分钟。所以，可能需要仿照 MySQL 集群一样，做读写分离。

实施方案：

1、N 台机器做热数据的存储, 上面只放当天的数据。这 N 台热数据节点上面的 elasticsearc.yml 中配置 node.tag: hot

2、之前的数据放在另外的 M 台机器上。这 M 台冷数据节点中配置 node.tag: stale

3、模板中控制对新建索引添加 hot 标签：

```
{
        "order" : 0,
        "template" : "*",
        "settings" : {
                "index.routing.allocation.require.tag" : "hot"
        }
}
```

4、每天计划任务更新索引的配置, 将 tag 更改为 stale, 索引会自动迁移到 M 台冷数据节点

```
# curl -XPUT http://127.0.0.1:9200/indexname/_settings -d'
{
        "index": {
                "routing": {
```

```
                "allocation": {
                        "require": {
                                "tag": "stale"
                                }
                        }
                }
        }
}'
```

这样，写操作集中在 N 台热数据节点上，大范围的读操作集中在 M 台冷数据节点上。避免了堵塞影响。

该方案运用的，是 Elasticsearch 中的 allocation filter 功能，详细说明见：

https://www.elastic.co/guide/en/elasticsearch/reference/master/shard-allocation-filtering.html

shard的allocate控制

某个 shard 分配在哪个节点上，一般来说，是由 ES 自动决定的。以下几种情况会触发分配动作：

1 新索引生成
2 索引的删除
3 新增副本分片
4 节点增减引发的数据均衡

ES 提供了一系列参数详细控制这部分逻辑：

- cluster.routing.allocation.enable 该参数用来控制允许分配哪种分片。默认是 all。可选项还包括 primaries 和 new_primaries。none 则彻底拒绝分片。该参数的作用，本书稍后集群升级章节会有说明。
- cluster.routing.allocation.allow_rebalance 该参数用来控制什么时候允许数据均衡。默认是 indices_all_active，即要求所有分片都正常启动成功以后，才可以进行数据均衡操作，否则的话，在集群重启阶段，会浪费太多流量了。
- cluster.routing.allocation.cluster_concurrent_rebalance 该参数用来控制集群内同时运行的数据均衡任务个数。默认是 2 个。如果有节点增减，且集群负载压力不高的时候，可以适当加大。
- cluster.routing.allocation.node_initial_primaries_recoveries 该参数用来控制节点重启时，允许同时恢复几个主分片。默认是 4 个。如果节点是多磁盘，且 IO 压力不大，可以适当加大。
- cluster.routing.allocation.node_concurrent_recoveries 该参数用来控制节点除了主分片重启恢复以外其他情况下，允许同时运行的数据恢复任务。默认是 2 个。所以，节点重启时，可以看到主分片迅速恢复完成，副本分片的恢复却很慢。除了副本分片本身数据要通过网络复制以外，并发线程本身也减少了一半。当然，这种设置也是有道理的——主分片一定是本地恢复，副本分片却需要走网络，带宽是有限的。从 ES 1.6 开始，冷索引的副本分片可以本地恢复，这个参数也就是可以适当加大了。
- indices.recovery.concurrent_streams 该参数用来控制节点从网络复制恢复副本分片时的数据流个数。默认是 3 个。可以配合上一条配置一起加大。
- indices.recovery.max_bytes_per_sec 该参数用来控制节点恢复时的速率。默认是 40MB。显然是比较小的，建议加大。

此外，ES 还有一些其他的分片分配控制策略。比如以 tag 和 rack_id 作为区分等。

一般来说，日志系统场景中使用不多。运维人员可能比较常见的策略有两种：

1 磁盘限额 为了保护节点数据安全，ES 会定时(cluster.info.update.interval，默认 30 秒)检查一下各节点的数据目录磁盘使用情况。在达到 cluster.routing.allocation.disk.watermark.low (默认 85%)的时候，新索引分片就不会再分配到这个节点上了。在达到 cluster.routing.allocation.disk.watermark.high (默认 90%)的时候，就会触发该节点现存分片的数据均衡，把数据挪到其他节点上去。这

两个值不但可以写百分比，还可以写具体的字节数。有些公司可能出于成本考虑，对磁盘使用率有一定的要求，需要适当抬高这个配置：

```
# curl -XPUT localhost:9200/_cluster/settings -d '{
        "transient" : {
                "cluster.routing.allocation.disk.watermark.low" : "85%",
                "cluster.routing.allocation.disk.watermark.high" : "10gb",
                 "cluster.info.update.interval" : "1m"
        }
}'
```

1 热索引分片不均 默认情况下，ES 集群的数据均衡策略是以各节点的分片总数(indices_all_active)作为基准的。这对于搜索服务来说无疑是均衡搜索压力提高性能的好办法。但是对于日志系统场景，一般压力集中在新索引的数据写入方面。正常运行的时候，也没有问题。但是当集群扩容时，新加入集群的节点，分片总数远远低于其他节点。这时候如果有新索引创建，ES 的默认策略会导致新索引的所有主分片几乎全分配在这台新节点上。整个集群的写入压力，压在一个节点上，结果很可能是这个节点直接被压死，集群出现异常。 所以，对于日志系统场景，强烈建议大家预先计算好索引的分片数后，配置好单节点分片的限额。比如，一个 5 节点的集群，索引主分片 10 个，副本 1 份。则平均下来每个节点应该有 4 个分片，那么就配置：

```
# curl -s -XPUT http://127.0.0.1:9200/logstash-2015.05.08/_settings -d '{
        "index": { "routing.allocation.total_shards_per_node" : "5" }
}'
```

注意，这里配置的是 5 而不是 4。因为我们需要预防有机器故障，分片发生迁移的情况。如果写的是 4，那么分片迁移会失败。

reroute 接口

上面说的各种配置，都是从策略层面，控制分片分配的选择。在必要的时候，还可以通过 ES 的 reroute 接口，手动完成对分片的分配选择的控制。

reroute 接口支持三种指令：allocate，move 和 cancel。常用的一半是 allocate 和 move：

- allocate 指令

因为负载过高等原因，有时候个别分片可能长期处于 UNASSIGNED 状态，我们就可以手动分配分片到指定节点上。默认情况下只允许手动分配副本分片，所以如果是主分片故障，需要单独加一个 allow_primary 选项：

```
# curl -XPOST 127.0.0.1:9200/_cluster/reroute -d '{
 "commands" : [ {
     "allocate" :
        {
          "index" : "lbf-2015.12.13", "shard" : 61, "node" : "10.19.0.77", "allow_primary" : true
        }
     }
 ]
```

```
}'
```

注意，如果是历史数据的话，请提前确认一下哪个节点上保留有这个分片的实际目录，且目录大小最大。然后手动分配到这个节点上。以此减少数据丢失。

- move 指令

因为负载过高，磁盘利用率过高，服务器下线，更换磁盘等原因，可以会需要从节点上移走部分分片：

```
curl -XPOST 127.0.0.1:9200/_cluster/reroute -d '{
  "commands" : [ {
      "move" :
         {
           "index" : "lbf-2015.12.13", "shard" : 0, "from_node" : "10.19.0.81", "to_node" :
"10.19.0.104"
         }
      }
  ]
}'
```