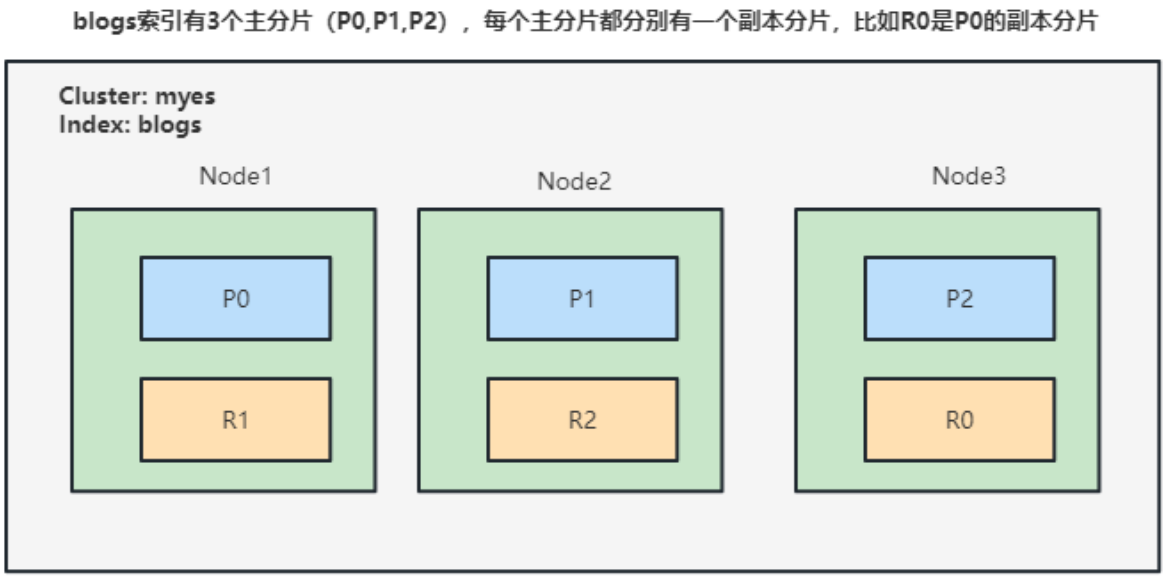
> 主讲老师：Fox
> 有道笔记链接：https://note.youdao.com/s/aOLNM6ud

1. ES底层读写工作原理分析

分片路由

如图所示：当我们想一个集群保存文档时，文档该存储到哪个节点呢？ 是随机吗？ 是轮询吗？



实际上，在ELasticsearch中，会采用计算的方式来确定存储到哪个节点，计算公式如下：

```
# es分片路由的规则
shard_num = hash(_routing) % num_primary_shards
# _routing字段的取值，默认是_id字段，可以自定义。
```

这就是为什么创建了主分片后，不能修改的原因。

写请求是写入 primary shard，然后同步给所有的 replica shard；读请求可以从 primary shard 或 replica shard 读取，采用的是随机轮询算法。

ES写入数据的过程

1. 客户端选择一个node发送请求过去，这个node就是coordinating node (协调节点)
2. coordinating node，对document进行路由，将请求转发给对应的node
3. node上的primary shard处理请求，然后将数据同步到replica node

4. coordinating node如果发现primary node和所有的replica node都搞定之后，就会返回请求到客户端

ES读取数据的过程

根据id查询数据的过程

根据 doc id 进行 hash，判断出来当时把 doc id 分配到了哪个 shard 上面去，从那个 shard 去查询。文档能够从主分片或任意一个复制分片被检索。

1. 客户端发送请求到任意一个 node，成为 coordinate node 。
2. coordinate node 对 doc id 进行哈希路由(hash(_id)%shards_size)，将请求转发到对应的 node，此时会使用 round-robin 随机轮询算法，在 primary shard 以及其所有 replica 中随机选择一个，让读请求负载均衡。
3. 接收请求的 node 返回 document 给 coordinate node 。
4. coordinate node 返回 document 给客户端。

根据关键词查询数据的过程

对于全文搜索而言，文档可能分散在各个节点上，那么在分布式的情况下，如何搜索文档呢?

- 客户端发送请求到一个 coordinate node 。
- 协调节点将搜索请求转发到所有的 shard 对应的 primary shard 或 replica shard ，都可以。
- query phase：每个 shard 将自己的搜索结果返回给协调节点，由协调节点进行数据的合并、排序、分页等操作，产出最终结果。
- fetch phase：接着由协调节点根据 doc id 去各个节点上拉取实际的 document 数据，最终返回给客户端。

写数据底层原理

核心概念

segment file: 存储倒排索引的文件，每个segment本质上就是一个倒排索引，每秒都会生成一个segment文件，当文件过多时es会自动进行segment merge（合并文件），合并时会同时将已经标注删除的文档物理删除。

commit point: 记录当前所有可用的segment，每个commit point都会维护一个.del文件，即每个.del文件都有一个commit point文件（es删除数据本质是不属于物理删除），当es做删改操作时首先会在.del文件中声明某个document已经被删除，文件内记录了在某个segment内某个文档已经被删除，当查询请求过来时在segment中被删除的文件是能够查出来的，但是当返回结果时会根据commit point维护的那个.del文件把已经删除的文档过滤掉

translog日志文件: 为了防止elasticsearch宕机造成数据丢失保证可靠存储，es会将每次写入数据同时写到translog日志中。

os cache：操作系统里面，磁盘文件其实都有一个东西，叫做os cache，操作系统缓存，就是说数据写入磁盘文件之前，会先进入os cache，先进入操作系统级别的一个内存缓存中去

Refresh

- 将文档先保存在Index buffer中，以refresh_interval为间隔时间，定期清空buffer，生成 segment,借助文件系统缓存的特性，先将segment放在文件系统缓存中，并开放查询，以提升搜索的实时性

Translog

- Segment没有写入磁盘，即便发生了宕机，重启后，数据也能恢复，从ES6.0开始默认配置是每次请求都会落盘

Flush

- 删除旧的translog 文件
- 生成Segment并写入磁盘 | 更新commit point并写入磁盘。ES自动完成，可优化点不多

2. 如何提升集群的读写性能

提升集群读取性能的方法

数据建模

- 尽量将数据先行计算，然后保存到Elasticsearch 中。尽量避免查询时的 Script计算

```
#避免查询时脚本
GET blogs/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {
          "title": "elasticsearch"
        }}
      ],

      "filter": {
        "script": {
          "script": {
            "source": "doc['title.keyword'].value.length()>5"
          }
        }
      }
    }
  }
}
```

- 尽量使用Filter Context，利用缓存机制，减少不必要的算分
- 结合profile，explain API分析慢查询的问题，持续优化数据模型
- 避免使用*开头的通配符查询

```
GET /es_db/_search
{
  "query": {
    "wildcard": {
      "address": {
        "value": "*白云*"
      }
    }
  }
}
```

优化分片

- 避免Over Sharing
  - 一个查询需要访问每一个分片，分片过多，会导致不必要的查询开销
- 结合应用场景，控制单个分片的大小
  - Search: 20GB
  - Logging: 50GB
- Force-merge Read-only索引
  - 使用基于时间序列的索引，将只读的索引进行force merge，减少segment数量

```
#手动force merge
POST /my_index/_forcemerge
```

提升写入性能的方法

- 写性能优化的目标: 增大写吞吐量，越高越好
- 客户端: 多线程，批量写
  - 可以通过性能测试，确定最佳文档数量
  - 多线程: 需要观察是否有HTTP 429（Too Many Requests）返回，实现 Retry以及线程数量的自动调节
- 服务器端: 单个性能问题，往往是多个因素造成的。需要先分解问题，在单个节点上进行调整并且结合测试，尽可能压榨硬件资源,以达到最高吞吐量
  - 使用更好的硬件。观察CPU / IO Block
  - 线程切换|堆栈状况

服务器端优化写入性能的一些手段

- 降低IO操作
  - 使用ES自动生成的文档Id
  - 一些相关的ES 配置，如Refresh Interval
- 降低 CPU 和存储开销
  - 减少不必要分词
  - 避免不需要的doc_values
  - 文档的字段尽量保证相同的顺序，可以提高文档的压缩率

- 尽可能做到写入和分片的均衡负载，实现水平扩展
  - Shard Filtering / Write Load Balancer
- 调整Bulk 线程池和队列

注意：ES 的默认设置，已经综合考虑了数据可靠性，搜索的实时性，写入速度，一般不要盲目修改。一切优化，都要基于高质量的数据建模。

建模时的优化

- 只需要聚合不需要搜索，index设置成false
- 不要对字符串使用默认的dynamic mapping。字段数量过多，会对性能产生比较大的影响
- Index_options控制在创建倒排索引时，哪些内容会被添加到倒排索引中。

如果需要追求极致的写入速度，可以牺牲数据可靠性及搜索实时性以换取性能：

- 牺牲可靠性: 将副本分片设置为0，写入完毕再调整回去
- 牺牲搜索实时性：增加Refresh Interval的时间
- 牺牲可靠性: 修改Translog的配置

降低 Refresh的频率

- 增加refresh_interval 的数值。默认为1s ，如果设置成-1，会禁止自动refresh
  - 避免过于频繁的refresh，而生成过多的segment 文件
  - 但是会降低搜索的实时性

```
PUT /my_index/_settings
{
    "index" : {
        "refresh_interval" : "10s"
    }
}
```

- 增大静态配置参数indices.memory.index_buffer_size
  - 默认是10%，会导致自动触发refresh

降低Translog写磁盘的频率，但是会降低容灾能力

- Index.translog.durability: 默认是request，每个请求都落盘。设置成async，异步写入
- Index.translog.sync_interval：设置为60s，每分钟执行一次
- Index.translog.flush_threshod_size: 默认512 m，可以适当调大。当translog 超过该值，会触发flush

分片设定

- 副本在写入时设为0，完成后再增加
- 合理设置主分片数，确保均匀分配在所有数据节点上
- Index.routing.allocation.total_share_per_node:限定每个索引在每个节点上可分配的主分片数

调整Bulk 线程池和队列

- 客户端
  - 单个bulk请求体的数据量不要太大，官方建议大约5-15m
  - 写入端的 bulk请求超时需要足够长，建议60s 以上
  - 写入端尽量将数据轮询打到不同节点。
- 服务器端
  - 索引创建属于计算密集型任务，应该使用固定大小的线程池来配置。来不及处理的放入队列，线程数应该配置成CPU核心数+1，避免过多的上下文切换
  - 队列大小可以适当增加，不要过大，否则占用的内存会成为GC的负担
  - ES线程池设置： https://blog.csdn.net/justlpf/article/details/103233215

```
DELETE myindex
PUT myindex
{
  "settings": {
    "index": {
      "refresh_interval": "30s",  #30s一次refresh
      "number_of_shards": "2"
    },
    "routing": {
      "allocation": {
        "total_shards_per_node": "3"  #控制分片，避免数据热点
      }
    },
    "translog": {
      "sync_interval": "30s",
      "durability": "async"    #降低translog落盘频率
    },
    "number_of_replicas": 0
  },
  "mappings": {
    "dynamic": false,     #避免不必要的字段索引，必要时可以通过update by query索引必要的字段
    "properties": {}
  }
}
```