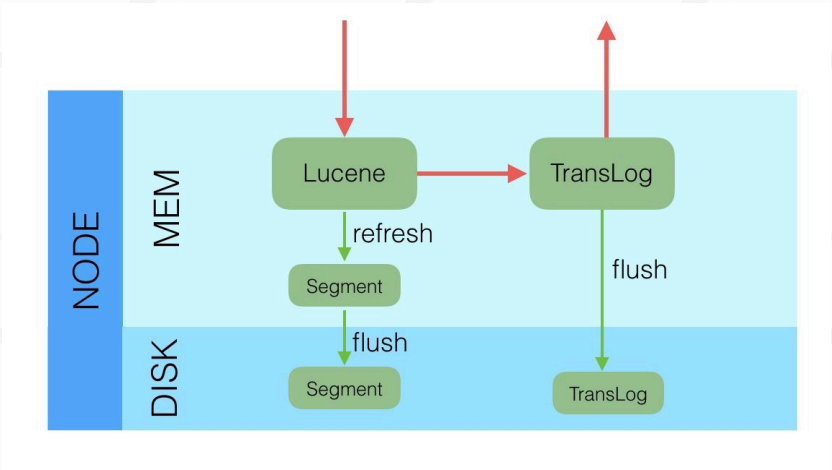
追求极致的写入速度时，很多是以牺牲可靠性和搜索实时性为代价的。



一、translog flush间隔调整

这是影响ES写入的最大因素。translog flush操作是将内存中的数据写入磁盘，典型的IO操作。ES默认为了写入的可靠性，采用的配置为:

> index.translog.durability: request
>
> 每个写入请求都flush到磁盘，确保写操作是可靠的

如果系统接受一定概率的数据丢失，可设置为根据固定周期和固定大小的flush操作，比如

> index.translog.durability: async
>
> index.translog.sync_interval: 120s
>
> or
>
> index.translog.flush_threshold_size: 1024mb

设置translog的flush频率可以控制可靠性，要么是按请求，每次请求都flush；要么是按时间，每隔一段时间flush一次，一般为了性能考虑，会设置为每隔5秒或者1分钟flush一次；flush间隔时间越长，可靠性就会越低。

二、索引刷新间隔refresh_interval

每次索引的refresh会产生一个新的Lucene段，增大refresh周期，可减少段的创建以及后续的Force Merge操作

三、段合并优化

segment merge操作对系统I/O和内存占用都比较高，需要调整参数，改变行为。（这点用的少）

四、indexing buffer

indexing buffer在为doc建立索引时使用，当缓冲满时会刷入磁盘，生成一个新的segment，这是除了refresh_interval刷新索引之外，另一个生成新segment的机会。

> indices.memory.index_buffer_size
>
> indices.memory.min_index_buffer_size
>
> indices.memory.max_index_buffer_size
>
> 当执行大量的索引操作时，indices.memory.index_buffer_size的默认设置可能不够，这和可用堆内存、单节点上的shard数量相关，可以考虑适当增大该值

五、使用bulk请求

批量写效率更高，每个请求最好避免超过几十兆，避免给集群带来压力

六、bulk线程池和队列

建立索引过程属于CPU密集型任务，应该使用固定大小的线程池配置，来不及处理的任务放入队列。线程池最大线程数量应配置为CPU核心数+1，队列可以适当的增加，但要控制大小，过大的队列会导致较高的GC压力

七、并发执行bulk请求

bulk写请求是个长任务，为了给系统增加足够的写入压力，写入过程应该多个客户端、多线程地并行执行，直至CPU打满。

```
# 索引bulk settings与mappings信息
{
  "bulk" : {
    "aliases" : { },
    "mappings" : {
      "doc" : {
        "properties" : {
          "age" : {
            "type" : "integer"
          },
          "name" : {
            "type" : "keyword"
          }
        }
      }
    },
    "settings" : {
      "index" : {
        "creation_date" : "1619512853526",
        "number_of_shards" : "1",
        "number_of_replicas" : "0",
        "uuid" : "TzpXptx5RJyr3ZZY68V1sw",
        "version" : {
          "created" : "6050499"
        },
        "provided_name" : "bulk"
      }
    }
  }
}

# 调用_bulk接口执行批量写入
POST _bulk
{"index":{"_index":"bulk","_type":"doc"}}
{"name":"19-tony","age":33}
{"index":{"_index":"bulk","_type":"doc"}}
{"name":"11-tom","age":45}


# elasticsearch client bulk的body参数为字符串类型的内容，比如
'{"index":{"_index":"bulk","_type":"doc"}}\n{"name":"19-tony","age":33}\n{"index":{"_index":"bulk",

```

八、磁盘间的任务均衡

ES多路径中磁盘的使用可能并不均匀，最好做成RAID0，提升性能

九、节点间的任务均衡

为了节点间的任务尽量均衡，数据写入客户端应该把bulk请求轮询发送到各个节点

十、自动生成doc ID

如果写入doc时指定了id，则ES会先尝试读取原来doc的版本号以判断是否需要更新，这会涉及一次读取磁盘的操作。总结下来两点：

1. 减少磁盘的IO操作
2. 自动生成的ID具有一定的规律，有利于FST的压缩。
3. Lucene从4.0版本开始大量使用FST（Finite State Transducer）；具有两个优点：1）空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间；2）查询速度快，O(len(str))的查询时间复杂度

十一、调整字段Mappings

1. 减少字段数量、对于不需要建立索引的字段，不写入ES（ES+HBase的组合使用）
2. 将不需要建立索引的字段index属性设置为not_analyzed或no。对字段不分词，或者不索引，可以减少很多运算操作，降低CPU使用
3. 减少字段内容长度
4. 使用不同的分词器，不同的分词器在索引过程中运算复杂度也有较大的差异

十二、调整_source字段

_source字段用于存储doc原始数据，对于部分不需要存储的字段，可以通过includes、excludes过滤。实际环境，一般不做调整

十三、禁用_all字段

ES 5.x _all默认开启，ES 6.x _all字段默认为不启用。_all字段中包含所有字段分词后的关键词，作用是可以在搜索的时候不指定特定字段，从所有字段中检索。ES 6.x默认禁用_all字段主要有以下几点原因：

1. 由于需要从其他的全部字段复制所有字段值，导致_all字段占用非常大的空间
2. _all字段有自己的分析器，在进行某些查询时，结果不符合预期
3. 由于数据重复引起的额外建立索引的开销
4. 想要调试时，其内容不容易检查
5. 有些用户甚至不知道存在这个字段，导致了查询混乱

可以通过mapping中将enabled设置为false来禁用_all字段，禁用_all字段可以明显降低对CPU和I/O的压力。

十四、对Analyzed的字段禁用Norms

Norms用于在搜索时计算doc的评分，如果不需要评分，则可以将其禁用：

```
"title": {"type": "text", "norms": {"enabled": false}}
```

十五、index_options设置

index_options用于控制在建立倒排索引过程中，哪些内容会被添加到倒排索引中；例如doc数量、词频、positions、offsets等，优化这些设置可以一定程度上降低索引过程中的运算任务；不过这个在实际环境中，很少用。

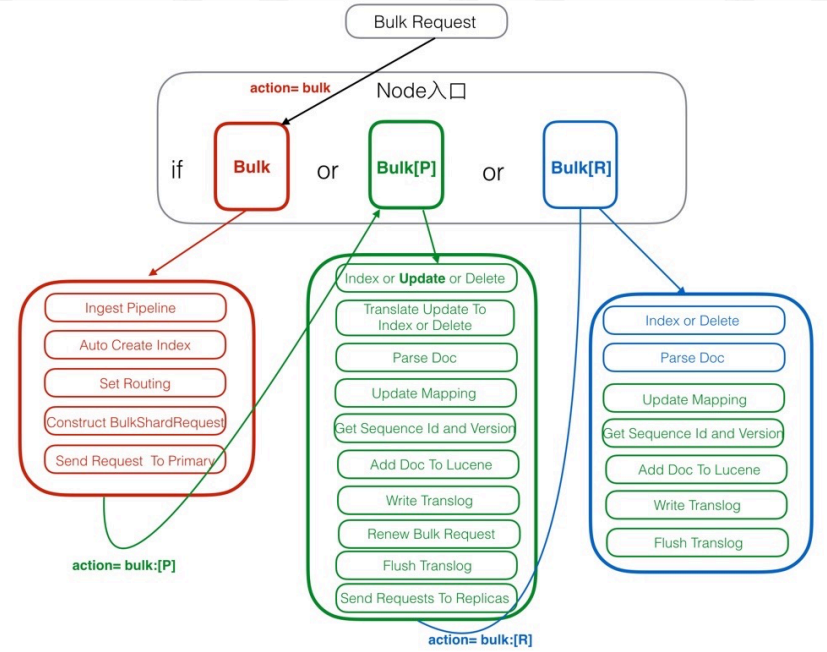
参考配置：

```
{
    "settings": {
        "index.merge.policy.max_merged_segment": "2gb",
```

```
        "index.merge.policy.segments_per_tier": "24",
        "index.number_of_replicas": "1",
        "index.number_of_shards": "24",
        "index.optimize_auto_generated_id": "true",
        "index.refresh_interval": "120s",
        "index.translog.durability": "async",
        "index.translog.flush_threshold_size": "1000mb",
        "index.translog.sync_interval": "120s",
        "index.unassigned.node_left.delayed_timeout": "5d"
    }
}
```

要点：

1. 写入时设置副本，有副本的索引创建操作，然后写入；如果是写入后，再设置副本，则此时采用的是复制操作
2. ES为了减少磁盘IO保证读写性能，一般是每隔一段时间（比如5分钟）才会把Lucene的segment写入磁盘持久化
3. 在每个shard中，写入流程分为两部分，先写入Lucene，后写入Translog



请求接着会发送给Primary Shard，在Primary Shard上执行成功后，再从Primary Shard上将请求同时发送给多个Replica Shard，请求在多个Replica Shard上执行成功并返回给Primary Shard后，写入请求执行成功，返回结果给客户端。

这种模式下，写入操作的延时就等于latency = Latency(Primary Write) + Max(Replicas Write)，只要有副本在，写入延时最小也是两次单Shard的写入时延总和，写入效率会较低，但是这样的好处也很明显，避免写入后，单机或磁盘故障导致数据丢失，在数据重要性和性能方面，一般都是优先选择数据，除非一些允许丢数据的特殊场景。

7. Write Translog

写完Lucene的Segment后，会以keyvalue的形式写TransLog，Key是_id，Value是Doc内容，当查询的时候，如果请求是GetDocByID，则可以直接根据_id从TransLog中读取到，满足NoSQL场景下的实时性要去。

6. 性能：性能是一个系统性工程，所有环节都要考虑对性能的影响，在Elasticsearch中，在很多地方的设计都考虑到了性能，一是不需要所有Replica都返回后才能返回给用户，只需要返回特定数目的就行；二是生成的Segment现在内存中提供服务，等一段时间后才刷新到磁盘，Segment在内存这段时间的可靠性由TransLog保证；三是TransLog可以配置为周期性的Flush，但这个会给可靠性带来伤害；四是每个线程持有一个Segment，多线程时相互不影响，相互独立，性能更好；五是系统的写入流程对版本依赖较重，读取频率较高，因此采用了versionMap，减少热点数据的多次磁盘IO开销。Lucene中针对性能做了大量的优化。后面我们