Elasticsearch索引和查询性能调优

一、Elasticsearch部署

1. 选择合理的硬件配置：尽可能使用 SSD

Elasticsearch 最大的瓶颈往往是磁盘读写性能，尤其是随机读取性能。使用SSD（PCI-E接口SSD卡/SATA接口SSD盘）通常比机械硬盘（SATA盘/SAS盘）查询速度快5~10倍，写入性能提升不明显。

对于文档检索类查询性能要求较高的场景，建议考虑 SSD 作为存储，同时按照 1:10 的比例配置内存和硬盘。**对于日志分析类查询并发要求较低的场景**，可以考虑采用机械硬盘作为存储，同时按照 1:50 的比例配置内存和硬盘。单节点存储数据建议在2TB以内，不要超过5TB，避免查询速度慢、系统不稳定。

在单机存储 1TB 数据场景下，SATA 盘和 SSD 盘的全文检索性能对比（测试环境：Elasticsearch5.5.3，10亿条人口户籍登记信息，单机16核CPU、64GB内存，12块6TB SATA盘，2块1.5 TB SSD盘）。

磁盘类型	并发数	QPS	平均检索响应时间	50%请求响应时间	90%请求响应时间	IOPS
SATA盘	10并发	17	563ms	478ms	994ms	1200
SATA盘	50并发	64	773ms	711ms	1155ms	1800
SATA盘	100并发	110	902ms	841ms	1225ms	2040
SATA盘	200并发	84	2369ms	2335ms	2909ms	2400
SSD盘	10并发	94	105ms	90ms	200ms	25400
SSD盘	50并发	144	346ms	341ms	411ms	66000
SSD盘	100并发	152	654ms	689ms	791ms	60000
SSD盘	200并发	210	950ms	1179ms	1369ms	60000

2. 给JVM配置机器一半的内存，但是不建议超过32G

修改 **conf/jvm.options** 配置，-Xms 和 -Xmx 设置为相同的值，推荐设置为机器内存的一半左右，剩余一半留给操作系统缓存使用。JVM 内存建议不要低于 2G，否则有可能因为内存不足导致 ES 无法正常启动或内存溢出，JVM 建议不要超过 32G，否则 JVM 会禁用内存对象指针压缩技术，造成内存浪费。机器内存大于 64G 内存时，推荐配置 -Xms30g -Xmx30g。JVM 堆内存较大时，内存垃圾回收暂停时间比较长，建议配置 ZGC 或 G1 垃圾回收算法。

3. 规模较大的集群配置专有主节点，避免脑裂问题

Elasticsearch 主节点负责集群元信息管理、index 的增删操作、节点的加入剔除，定期将最新的集群状态广播至各个节点。在集群规模较大时，建议配置专有主节点只负责集群管理，不存储数据，不承担数据读写压力。

```
# 专有主节点配置(conf/elasticsearch.yml):
node.master:true
node.data: false
node.ingest:false


# 数据节点配置(conf/elasticsearch.yml):
node.master:false
node.data:true
node.ingest:true
```

Elasticsearch 默认每个节点既是候选主节点，又是数据节点。最小主节点数量参数 **minimum_master_nodes** 推荐配置为候选主节点数量一半以上，该配置告诉 Elasticsearch 当没有足够的 master 候选节点的时候，不进行 master 节点选举，等 master 节点足够了才进行选举。

例如对于 3 节点集群，最小主节点数量从默认值 1 改为 2。

```
# 最小主节点数量配置(conf/elasticsearch.yml):
discovery.zen.minimum_master_nodes: 2
```

4. Linux操作系统调优

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

# 单用户可以打开的最大文件数量，可以设置为官方推荐的65536或更大些
echo "* - nofile 655360" >> /etc/security/limits.conf

# 单用户线程数调大
echo "* - nproc 131072" >> /etc/security/limits.conf

# 单进程可以使用的最大map内存区域数量
echo "vm.max_map_count = 655360" >> /etc/sysctl.conf

# 参数修改立即生效
sysctl -p
```

二、索引性能调优

1. 设置合理的索引分片数和副本数

索引分片数建议设置为集群节点的整数倍，初始数据导入时副本数设置为 0，生产环境副本数建议设置为 1（设置 1 个副本，集群任意 1 个节点宕机数据不会丢失；设置更多副本会占用更多存储空间，操作系统缓存命中率会下降，检索性能不一定提升）。单节点索引分片数建议不要超过 3 个，每个索引分片推荐 10-40GB 大小，索引分片数设置后不可以修改，副本数设置后可以修改。

Elasticsearch6.X 及之前的版本默认索引分片数为 5、副本数为 1，从 Elasticsearch7.0 开始调整为默认索引分片数为 1、副本数为 1。

不同分片数对写入性能的影响（测试环境：7节点Elasticsearch6.3集群，写入30G新闻数据，单节点56核CPU、380G内存、3TB SSD卡，0副本，20线程写入，每批次提交10M左右数据）。

集群索引分片数	单节点索引分片数	写入耗时
2	0/1	600s
7	1	327s
14	2	258s
21	3	211s
28	4	211s
56	8	214s

2. 使用批量请求

使用批量请求将产生比单文档索引请求好得多的性能。写入数据时调用批量提交接口，推荐每批量提交 **5~15MB** 数据。例如单条记录 1KB 大小，每批次提交 10000 条左右记录写入性能较优；单条记录 5KB 大小，每批次提交 2000 条左右记录写入性能较优。

```
# 批量请求接口API
curl -XPOST "http://localhost:9200/_bulk" -H 'Content-Type: application/json'
-d'
{ "index" : { "_index" : "test", "_type" : "_doc", "_id" : "1" } }{ "field1"
: "value1" }
{ "delete" : { "_index" : "test", "_type" : "_doc", "_id" : "2" } }
{ "create" : { "_index" : "test", "_type" : "_doc", "_id" : "3" } }{ "field1"
: "value3" }
{ "update" : {"_id" : "1", "_type" : "_doc", "_index" : "test"} }{ "doc" :
{"field2" : "value2"} }'
```

3. 通过多进程/线程发送数据

单线程批量写入数据往往不能充分利用服务器 CPU 资源，可以尝试调整写入线程数或者在多个客户端上同时向 Elasticsearch 服务器提交写入请求。与批量调整大小请求类似，只有测试才能确定最佳的 worker 数量。可以通过逐渐增加工作任务数量来测试，直到集群上的 I/O 或 CPU 饱和。

4. 调大refresh interval

在 Elasticsearch 中，写入和打开一个新段的轻量的过程叫做 refresh 。 默认情况下每个分片会每秒自动刷新一次。这就是为什么我们说 Elasticsearch 是近实时搜索: 文档的变化并不是立即对搜索可见，但会在一秒之内变为可见。

并不是所有的情况都需要每秒刷新。可能你正在使用 Elasticsearch 索引大量的日志文件，你可能想优化索引速度而不是近实时搜索，可以通过设置 refresh_interval，降低每个索引的刷新频率。

```
# 设置 refresh interval API
curl -XPUT "http://localhost:9200/index" -H 'Content-Type: application/json'
-d'{
    "settings": {
        "refresh_interval": "30s"
    }
}'
```

refresh_interval 可以在已经存在的索引上进行动态更新，在生产环境中，当你正在建立一个大的新索引时，可以先关闭自动刷新，待开始使用该索引时，再把它们调回来。

```
curl -XPUT "http://localhost:9200/index/_settings" -H 'Content-Type:
application/json'
-d'{ "refresh_interval": -1 }'

curl -XPUT "http://localhost:9200/index/_settings" -H 'Content-Type:
application/json'
-d'{ "refresh_interval": "1s" }'
```

5. 配置事务日志参数

事务日志 translog 用于防止节点失败时的数据丢失。它的设计目的是帮助 shard 恢复操作，否则数据可能会从内存 flush 到磁盘时发生意外而丢失。事务日志 translog 的落盘(fsync)是 ES 在后台自动执行的，默认每 5 秒钟提交到磁盘上，或者当 translog 文件大小大于 512MB 提交，或者在每个成功的索引、删除、更新或批量请求时提交。

索引创建时，可以调整默认日志刷新间隔 5 秒，例如改为 60 秒，**index.translog.sync_interval: "60s"**。创建索引后，可以动态调整 translog 参数，**"index.translog.durability":"async"** 相当于关闭了 index、bulk 等操作的同步 flush translog 操作，仅使用默认的定时刷新、文件大小阈值刷新的机制。

```
# 动态设置 translog API
curl -XPUT "http://localhost:9200/index" -H 'Content-Type: application/json'
-d'{
    "settings": {
        "index.translog.durability": "async",
        "translog.flush_threshold_size": "2gb"
    }
}'
```

6. 设计mapping配置合适的字段类型

Elasticsearch 在写入文档时，如果请求中指定的索引名不存在，会自动创建新索引，并根据文档内容猜测可能的字段类型。但这往往不是最高效的，我们可以根据应用场景来设计合理的字段类型。

```
# 例如写入一条记录
curl -XPUT "http://localhost:9200/twitter/doc/1?pretty" -H 'Content-Type: application/json'
-d'{
    "user": "kimchy",
    "post_date": "2009-11-15T13:12:00",
    "message": "Trying out Elasticsearch, so far so good?"
}'
```

查询 Elasticsearch 自动创建的索引 mapping，会发现将 post_date 字段自动识别为 date 类型，但是 message 和 user 字段被设置为 text、keyword 冗余字段，造成写入速度降低、占用更多磁盘空间。

```
{
    "twitter": {
        "mappings": {
            "doc": {
                "properties": {
                    "message": {
                        "type": "text",
                        "fields": {
                            "keyword": {
                                "type": "keyword",
                                "ignore_above": 256
                            }
                        }
                    },
                    "post_date": {
                        "type": "date"
                    },
                    "user": {
                        "type": "text",
                        "fields": {
                            "keyword": {
                                "type": "keyword",
                                "ignore_above": 256
                            }
                        }
                    }
                }
            }
        },
        "settings": {
            "index": {
                "number_of_shards": "5",
                "number_of_replicas": "1"
            }
        }
    }
}
```

根据业务场景设计索引配置合理的分片数、副本数，设置字段类型、分词器。如果不需要合并全部字段，禁用 **_all** 字段，通过 **copy_to** 来合并字段。

```
curl -XPUT "http://localhost:9200/twitter?pretty" -H 'Content-Type:
application/json'
-d'{
    "settings": {
        "index": {
            "number_of_shards": "20",
            "number_of_replicas": "0"
        }
    }
}'

curl -XPOST "http://localhost:9200/twitter/doc/_mapping?pretty" -H 'Content-
Type: application/json'
-d'{
    "doc": {
        "_all": {
            "enabled": false
        },
        "properties": {
            "user": {
                "type": "keyword"
            },
            "post_date": {
                "type": "date"
            },
            "message": {
                "type": "text",
                "analyzer": "cjk"
            }
        }
    }
}'
```

三、查询性能调优

1. 配置合适的分词器

Elasticsearch 内置了很多分词器，包括 standard、cjk、nGram 等，也可以安装自研/开源分词器。根据业务场景选择合适的分词器，避免全部采用默认 standard 分词器。

常用分词器：

- **standard**：默认分词，英文按空格切分，中文按照单个汉字切分。
- **cjk**：根据二元索引对中日韩文分词，可以保证查全率。
- **nGram**：可以将英文按照字母切分，结合ES的短语搜索(match_phrase)使用。
- **IK**：比较热门的中文分词，能按照中文语义切分，可以自定义词典。
- **pinyin**：可以让用户输入拼音，就能查找到相关的关键词。
- **aliws**：阿里巴巴自研分词，支持多种模型和分词算法，词库丰富，分词结果准确，适用于电商等对查准要求高的场景。

```
# 分词效果测试API
curl -XPOST "http://localhost:9200/_analyze" -H 'Content-Type:
application/json'
-d'{
    "analyzer": "ik_max_word",
    "text": "南京市长江大桥"
}'
```

分词器	分词结果	查全率	查准率	查询性能
标准分词器standard	南/京/市/长/江/大/桥	高	低	低
中日韩文分词器cjk	南京/京市/市长/长江/江大/大桥	高	低	高
IK中文分词器ik_max_word	南京市/南京/市长/长江大桥/长江/大桥	高	低	高
IK中文分词器ik_smart	南京市/长江大桥	低	高	高
阿里中文分词器aliws	南京/市/长江/大桥	高	高	高

2. 设置查询读取记录条数和字段**

默认的查询请求通常返回排序后的前 10 条记录，最多一次读取 10000 条记录，通过 from 和 size 参数控制读取记录范围，避免一次读取过多的记录。通过 _source 参数可以控制返回字段信息，尽量避免读取大字段。

```
# 查询请求示例
curl -XGET http://localhost:9200/fulltext001/_search?pretty  -H 'Content-
Type: application/json'
-d '{
    "from": 0,
    "size": 10,
    "_source": "id",
    "query": {
        "bool": {
            "must": [
                {
                    "match": {
                        "content": "虎嗅"
                    }
                }
            ]
        }
    },
    "sort": [
        {
            "id": {
                "order": "asc"
            }
        }
    ]
}'
```

3. 设置 teminate_after 查询快速返回**

如果不需要精确统计查询命中记录条数，可以配 **teminate_after** 指定每个 shard 最多匹配 N 条记录后返回，设置查询超时时间 timeout。在查询结果中可以通过 “terminated_early” 字段标识是否提前结束查询请求。

```
# teminate_after 查询语法示例
curl -XGET "http://localhost:9200/twitter/_search" -H 'Content-Type: application/json'
-d'{
    "from": 0,
    "size": 10,
    "timeout": "10s",
    "terminate_after": 1000,
    "query": {
        "bool": {
            "filter": {
                "term": {
                    "user": "elastic"
                }
            }
        }
    }
}'
```

4. 避免前缀模糊匹配**

Elasticsearch 默认支持通过 *? 正则表达式来做模糊匹配，如果在一个数据量较大规模的索引上执行模糊匹配，尤其是前缀模糊匹配，通常耗时会比较长，甚至可能导致内存溢出。尽量避免在高并发查询请求的生产环境执行这类操作。

某客户需要对车牌号进行模糊查询，通过查询请求 **"车牌号:*A8848*"** 查询时，往往导致整个集群负载较高。通过对数据预处理，增加冗余字段 "车牌号.keyword"，并事先将所有车牌号按照1元、2元、3元...7元分词后存储至该字段，字段存储内容示例：沪,A,8,4,沪A,A8,88,84,48,沪A8...沪A88488。通过查询"**车牌号.keyword:A8848**"即可解决原来的性能问题。

5. 避免索引稀疏**

Elasticsearch6.X 之前的版本默认允许在一个 index 下面创建多个 type，Elasticsearch6.X 版本只允许创建一个 type，Elasticsearch7.X 版本只允许 type 值为 “**_doc**”。在一个索引下面创建多个字段不一样的 type，或者将几百个字段不一样的索引合并到一个索引中，会导致索引稀疏问题。

建议每个索引下只创建一个 type，字段不一样的数据分别独立创建 index，不要合并成一个大索引。每个查询请求根据需要去读取相应的索引，避免查询大索引扫描全部记录，加快查询速度。

6. 扩容集群节点个数，升级节点规格**

通常服务器节点数越多，服务器硬件配置规格越高，Elasticsearch 集群的处理能力越强。

在不同节点规模下的查询性能测试（测试环境：Elasticsearch5.5.3 集群，单节点16核CPU、64G内存、2T SSD盘，10亿条人口户籍登记信息，数据大小1TB, 20索引分片）。

集群节点数	副本数	10并发检索平均响应时间	50并发检索平均响应时间	100并发检索平均响应时间	200并发检索平均响应时间	200并发QPS	200并发CPU使用率	200并发CPUIO等待
1	0	77ms	459ms	438ms	1001ms	200	16%	52%
3	0	38ms	103ms	162ms	298ms	669	45%	34%
3	2	271ms	356ms	577ms	818ms	244	19%	54%
10	0	21ms	36ms	48ms	81ms	2467	40%	10%

不同集群节点规模写入性能测试（测试环境：Elasticsearch6.3.2 集群，单节点16核CPU、64G内存、2T SSD盘，10亿条人口户籍登记信息，单条记录1KB，数据集大小1TB，20个并发写入线程）。

集群节点数	副本数	写入TPS	耗时	集群CPU使用率
10	0	88945	11242s	50%
50	0	180638	5535s	20%