勘误时间 20181130（第一次）

P337

1.将“某个节点”改为“每个节点”

2.命令中新增参数：name,ip,port,并更新了输出结果，命令和结果的完整信息如下：

```
curl -X GET "localhost:9200/_cat/nodes?v&h=name,ip,port,segments.memory"
name   ip        port segments.memory
node-1 127.0.0.1 9300          9.2kb
```

如下图所示：

P338

更新了命令参数，h= 后面的内容修改为： name,ip,port,fm

如下图：

P338

更新了命令参数，h= 后面的参数修改为：name,ip,port,rcm

可以使用下面的 API 来获取缓存使用量：

```
curl -X GET "localhost:9200/_stats/request_cache?pretty"
curl -X GET "localhost:9200/_nodes/stats/indices/request_cache?pretty"
 curl -X GET "localhost:9200/_cat/nodes?v&h=name,ip,port,rcm"
```

Node Query Cache　节点查询缓存由节点上的所有分片共享，也是一个 LRU 缓存，用于

张超
删除的内容：request_cache.memory_size

P339

更新了命令参数，h= 后面的参数修改为：name,ip,port,qcm

缓存查询结果，只缓存在过滤器上下文中使用的查询。该缓存默认开启，大小为堆大小的 10%。可以通过 indices.queries.cache.size 选项来配置大小，同时可以通过 index.queries.cache.enabled 选项在索引级启用或禁用该缓存。

该缓存的使用量可以通过下面的命令来获取：

```
curl -X GET "localhost:9200/_cat/nodes?v&h=name,ip,port,qcm"
```

此外，ES 进程的内存使用量还与 Lucene 以 mmap 方式加载段文件相关。mmap 加载的文件会被分配进程地址空间，因此它们同样算作 ES 占用的内存，我们可以通过 pmap 命令查看进程都有哪些文件被映射进来。通过 mmap 系统调用映射进来的段文件数据量通常都比较大，

张超
删除的内容：query_cache.memory_size

勘误时间 20181212（第 2 次）

P248

图中所示位置添加一个漏掉的乘号 *

ThreadPool 模块分析

每个节点都会创建一系列的线程池来执行任务，许多线程池都有与其相关任务队列，用来允许挂起请求，而不是丢弃它。下面列出目前 ES 版本中的线程池。

generic

用于通用的操作（例如，节点发现），线程池类型为 scaling。

index

用于 index/delete 操作，线程池类型为 fixed，大小为处理器的数量，队列大小为 200，允许设置的最大线程数为 1+处理器数量。

search

用于 count/search/suggest 操作。线程池类型为 fixed，大小为 int((处理器数量*3)/2)+1，队列大小为 1000。

勘误时间 20181219（第 3 次）

P22

增加图中所示的蓝色文字：“Bully 算法的”

原因：ES 实现选择 ID 最小的，Bully 选择最大的，此处增加明确说明，避免误解。

3.1 选举主节点

假设有若干节点正在启动，集群启动的第一件事是从已知的活跃机器列表中选择一个作为主节点，选主之后的流程由主节点触发。

ES 的选主算法是基于 Bully 算法的改进， Bully 算法的主要思路是对节点 ID 排序，取 ID 值最大的节点作为 Master，每个节点都运行这个流程。是不是非常简单？选主的目的是确定唯一的主节点，初学者可能认为选举出的主节点应该持有最新的元数据信息，实际上这个问题在实现上被分解为两步：先确定唯一的、大家公认的主节点，再想办法把最新的机器元数据复制到选举出的主节点上。

基于节点 ID 排序的简单选举算法有三个附加约定条件：

P216

原句改写为 “而是部分节点成功就算成功。其数量取决于配置项 :”

原因：原文表述不够精确

14.5 集群状态的发布过程

发布集群状态是一个分布式事务操作，分布式事务需要实现原子性：要么所有参与者都提交事务，要么都取消事务。ES 使用二段提交来实现分布式事务。二段提交可以避免失败回滚，其基本过程是：把信息发下去，但不应用，如果得到多数节点的确认，则再发一个请求出去要求节点应用。

ES 实现二段提交与标准二段提交有一些区别，发布集群状态到参与者的数量并非定义为全部，而是部分~~多数~~节点成功就算成功。其数量~~多数的定义~~取决于配置项：

```
discovery.zen.minimum_master_nodes
```

勘误时间 20190127（第 4 次）

P131

增加描述：“默认为 40mb,”

10.5 recovery 速度优化

众所周知，索引恢复是集群启动过程中最缓慢的过程，集群完全重启，或者 Master 节点挂掉后，新选出的 Master 也有可能执行这个过程。

官方也一直在优化索引恢复速度，陆续添加了 syncid 和 SequenceNumber。下面归纳一下有哪些方法可以提升索引恢复速度：

- 配置项 cluster.routing.allocation.node_concurrent_recoveries 决定了单个节点执行副分片 recovery 时的最大并发数（进/出），默认为 2，适当提高此值可以增加 recovery 并发数。
- 配置项 indices.recovery.max_bytes_per_sec 决定节点间复制数据时的限速，默认为 40mb，可以适当提高此值或取消限速。
- 配置项 cluster.routing.allocation.node_initial_primaries_recoveries 决定了单个节点执行主分片 recovery 时的最大并发数，默认为 4。由于主分片的恢复不涉及在网络上复制数据，仅在本地磁盘读写，所以在节点配置了多个数据磁盘的情况下，可以适当提高此值。

P339

增加描述：“该配置支持动态修改，但是修改前需要先对索引执行 _close 操作，修改完毕后重新 _open。”

如果 mmap 带的来难以控制的内存占用对系统来说是个麻烦，则可以考虑调整存储类型：

```
index.store.type: niofs
```

将默认的 mmapfs 修改为 niofs 等类型，该配置支持动态修改，但是修改前需要先对索引执行 _close 操作，修改完毕后重新 _open。虽然使用 mmap 可以少一次内存拷贝，但是由于目前 Lucene 使用 mmap 时不控制它的预读方式，mmap 会预读取 2MB 的数据，在随机 I/O 的场景中，其效率未必会高于 NIO。具体可以参考 https://github.com/elastic/elasticsearch/issues/27748。

22.5 Slow Logs

p301

增加描述：“并且由于这种方式计算出来的 JVM 内存一般较小， 在节点压力较大时进程更容易 OOM，因此不推荐。”

21.1.2 单节点还是多节点部署

ES 不建议为 JVM 配置超过 32GB 的内存，超过 32GB 时，Java 内存指针压缩失效，浪费一些内存，降低了 CPU 性能，GC 压力也较大。因此推荐设置为 31GB：

```
-Xmx31g -Xms31g
```

确保堆内存最小值（Xms）与最大值（Xmx）大小相同，防止程序在运行时动态改变堆内存大小，这是很耗系统资源的过程。

当物理主机内存在 64GB 以上，并且拥有多个数据盘，不做 raid 的情况下，部署 ES 节点时有多种选择：

（1）部署单个节点，JVM 内存配置不超过 32GB，配置全部数据盘。这种部署模式的缺点是多余的物理内存只能被 cache 使用，而且只要存在一个坏盘，节点重启会无法启动。

（2）部署单个节点，JVM 内存配置超过 32GB，配置全部数据盘。接受指针压缩失效和更长时间的 GC 等负面影响。

（3）有多少个数据盘就部署多少个节点，每个节点配置单个数据路径。优点是可以统一配置，缺点是节点数较多，集群管理负担大，只适用于集群规模较小的场景。并且由于这种方式计算出来的 JVM 内存一般较小， 在节点压力较大时进程更容易 OOM，因此不推荐。

P32

增加文字如下：

同时，在 CentOS 中，系统会按字母顺序加载 /etc/security/limits.d/ 下的配置文件，如果存在相同配置项，将会覆盖 /etc/security/limits.conf 中的配置，因此如果 /etc/security/limits.conf 中的配置没有生效，可以检查一下 /etc/security/limits.d/ 下的配置文件。

配置修改完毕后，重新登录系统，配置生效，无需重启操作系统。可以使用 ulimit –u 命令来验证是否生效。

- **4. 最大线程数检查**

ES 将请求分解为多个阶段执行，每个阶段使用不同的线程池来执行。因此 ES 进程需要创建很多线程，本项检查就是确保 ES 进程有创建足够多线程的权限。本项检查只对 Linux 系统进行。你需要调节进程可以创建的最大线程数，这个值至少是 2048。

要通过这项检查，可以修改/etc/security/limits.conf 文件的 nproc 来完成配置。同时，在 CentOS 中，系统会按字母顺序加载 /etc/security/limits.d/ 下的配置文件，如果存在相同配置项，将会覆盖 /etc/security/limits.conf 中的配置，因此如果 /etc/security/limits.conf 中的配置没有生效，可以检查一下 /etc/security/limits.d/ 下的配置文件。

配置修改完毕后，重新登录系统，配置生效，无需重启操作系统。可以使用 ulimit –u 命令来验证是否生效。

勘误时间 20190214（第 5 次）

P75，第 7.3 章末尾

原文描述的与本书版本不一致，因此更新描述如下：

在客户端收到成功响应时，意味着写操作已经在主分片和所有副分片都执行完成。

在 2.x 及之前的版本中，写一致性的默认策略是 quorum，即多数的分片（其中分片副本可以是主分片或副分片）在写入操作时处于可用状态。

```
quorum = int( (primary + number_of_replicas) / 2 ) + 1
```

在 5.x 及 6.x 中，写一致性策略由 wait_for_active_shards 参数控制，默认情况下，在执行写入操作前，只要主分片处于活跃状态就可以执行写入操作。wait_for_active_shards 用于指定在开始执行写入操作前，需要等待的活跃分片数量。但这和写入操作开始执行之后，有多少个分片写入成功没有关系。也就是说他仅是写入前的检查，不保证多少个分片副本写入成功。

7.4 Index/Bulk 详细流程

文字如下：

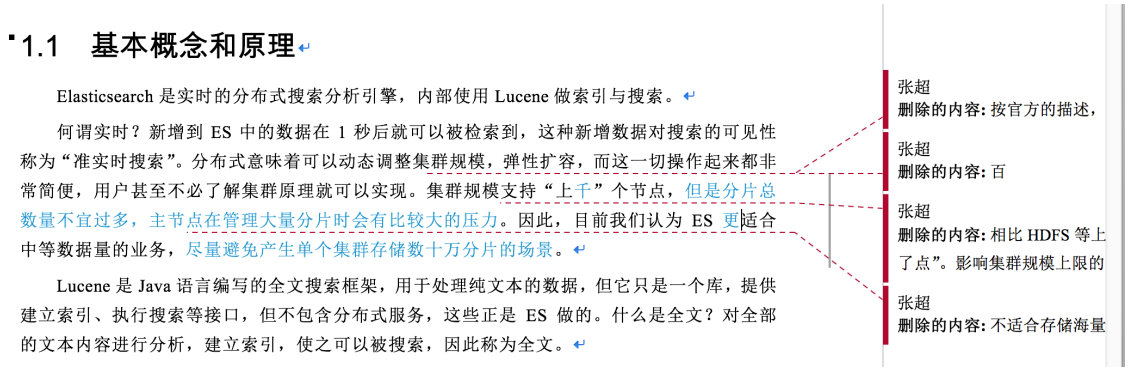
在 2.x 及之前的版本中，写一致性的默认策略是 quorum，即多数的分片（其中分片副本可以是主分片或副分片）在写入操作时处于可用状态。

```
quorum = int( (primary + number_of_replicas) / 2 ) + 1
```

在 5.x 及 6.x 中，写一致性策略由 wait_for_active_shards 参数控制，默认情况下，在执行写入操作前，只要主分片处于活跃状态就可以执行写入操作。wait_for_active_shards 用于指定在开始执行写入操作前，需要等待的活跃分片数量。但这和写入操作开始执行之后，有多少个分片写入成功没有关系。也就是说他仅是写入前的检查，不保证多少个分片副本写入成功。

P1

开篇就吐槽不太好，还是应该乐观的对待缺陷，因此把文字改得温和一些：



文字信息如下：

集群规模支持“上千”个节点，但是分片总数量不宜过多，主节点在管理大量分片时会有比较大的压力。因此，目前我们认为 ES 更适合中等数据量的业务，尽量避免产生单个集群存储数十万分片的场景。