**一、参数优化**

**文件句柄**

Linux中，每个进程默认打开的最大文件句柄数是1000,对于服务器进程来说，显然太小，通过修改/etc/security/limits.conf来增大打开最大句柄数

|  |  |
| --- | --- |
| 1 | \* - nofile 65535 |

**虚拟内存设置**

max\_map\_count定义了进程能拥有的最多内存区域

|  |  |
| --- | --- |
| 1 | sysctl -w vm.max\_map\_count=262144 |

修改/etc/elasticsearch/elasticsearch.yml

|  |  |
| --- | --- |
| 1 | bootstrap.mlockall: true |

修改/etc/security/limits.conf, 在limits.conf中添加如下内容

|  |  |
| --- | --- |
| 1  2 | \* soft memlock unlimited  \* hard memlock unlimited |

memlock 最大锁定内存地址空间， 要使limits.conf文件配置生效，必须要确保pam\_limits.so文件被加入到启动文件中。

确保/etc/pam.d/login文件中有如下内容

|  |  |
| --- | --- |
| 1 | session required /lib/security/pam\_limits.so |

验证是否生效

|  |  |
| --- | --- |
| 1 | curl localhost:9200/\_nodes/stats/process?pretty |

**磁盘缓存相关参数**

*vm.dirty\_background\_ratio* 这个参数指定了当文件系统缓存脏页数量达到系统内存百分之多少时（如5%）就会触发pdflush/flush/kdmflush等后台回写进程运行，将一定缓存的脏页异步地刷入外存；

*vm.dirty\_ratio*

* 该参数指定了当文件系统缓存脏页数量达到系统内存百分之多少时（如10%），系统不得不开始处理缓存脏页（因为此时脏页数量已经比较多，为了避免数据丢失需要将一定脏页刷入外存）；在此过程中很多应用进程可能会因为系统处理文件IO而阻塞
* 把该参数适当调小。如果cached的脏数据所占比例（这里是占MemTotal的比例）超过这个设置，系统会停止所有的应用层的IO写操作，等待刷完数据后恢复IO。所以万一触发了系统的这个操作，对于用户来说影响非常大的

|  |  |
| --- | --- |
| 1  2 | sysctl -w vm.dirty\_ratio=10  sysctl -w vm.dirty\_background\_ratio=5 |

为了将设置永久保存，将上述配置项写入**/etc/sysctl.conf**文件中

|  |  |
| --- | --- |
| 1  2 | vm.dirty\_ratio = 10  vm.dirty\_background\_ratio = 5 |

**swap调优**

swap空间是一块磁盘空间，操作系统使用这块空间保存从内存中换出的操作系统不常用page数据，这样可以分配出更多的内存做page cache。这样通常会提升系统的吞吐量和IO性能，但同样会产生很多问题。页面频繁换入换出会产生IO读写、操作系统中断，这些都很影响系统的性能。这个值越大操作系统就会更加积极的使用swap空间。

调节swappniess方法如下

|  |  |
| --- | --- |
| 1 | sudo sh -c 'echo "0">/proc/sys/vm/swappiness' |

**io sched**

如果集群中使用的是SSD磁盘，那么可以将默认的io sched由cfq设置为noop

|  |  |
| --- | --- |
| 1 | sudo sh -c 'echo "noop">/sys/block/sda/queue/scheduler' |

**在bin/elasticsearch.in.sh中进行配置**

修改配置项为尽量大的内存：

|  |  |
| --- | --- |
| 1  2 | ES\_MIN\_MEM=8g  ES\_MAX\_MEM=8g |

两者最好改成一样的，否则容易引发长时间GC（stop-the-world）

elasticsearch默认使用的GC是CMS GC，如果你的内存大小超过6G，CMS是不给力的，容易出现stop-the-world，建议使用G1 GC

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | JAVA\_OPTS=”$JAVA\_OPTS -XX:+UseParNewGC”  JAVA\_OPTS=”$JAVA\_OPTS -XX:+UseConcMarkSweepGC”    JAVA\_OPTS=”$JAVA\_OPTS -XX:CMSInitiatingOccupancyFraction=75″  JAVA\_OPTS=”$JAVA\_OPTS -XX:+UseCMSInitiatingOccupancyOnly”    #修改为：    JAVA\_OPTS=”$JAVA\_OPTS -XX:+UseG1GC”  JAVA\_OPTS=”$JAVA\_OPTS -XX:MaxGCPauseMillis=200″ |

G1 GC优点是减少stop-the-world的几率，但是CPU占有率高

**二、服务器配置**

**1、节点配置**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | cluster.name: es-cluster  //这个是配置集群的名字，为了能进行自动查找  node.name: "node-100" //这个是配置当前节点的名字，当然每个节点的名字都应该是唯一的  node.master: false  node.data: true  //这两个配置有4种配置方法，表示这个节点是否可以充当主节点，这个节点是否充当数据节点。  //如果你的节点数目只有两个的话，为了防止脑裂的情况，需要手动设置主节点和数据节点。  //其他情况建议直接不设置，默认两个都为true    network.host: "0.0.0.0" //绑定host，0.0.0.0代表所有IP，为了安全考虑，建议设置为内网IP'  transport.tcp.port: 9300 //节点到节点之间的交互是使用tcp的，这个设置设置启用的端口  http.port: 9200  //这个是对外提供http服务的端口，安全考虑，建议修改，不用默认的9200 |

* 当master为false，而data为true时，会对该节点产生严重负荷；
* 当master为true，而data为false时，该节点作为一个协调者；
* 当master为false，data也为false时，该节点就变成了一个负载均衡器。

**2、自动发现**

es提供了四种选择，一种是默认实现，其他都是通过插件实现。

* Azure discovery 插件方式，多播
* EC2 discovery 插件方式，多播
* Google Compute Engine (GCE)discovery 插件方式多播
* zen discovery默认实现 多播/单播

elasticsearch的集群是内嵌自动发现功能的。

单播配置下，节点向指定的主机发送单播请求，配置如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | discovery.zen.ping.multicast.enabled: false  discovery.zen.fd.ping\_timeout: 100s  discovery.zen.ping.timeout: 100s  discovery.zen.minimum\_master\_nodes: 2  discovery.zen.ping.unicast.hosts: ["172.31.26.200", "172.31.26.222"] |

多播配置下，意思就是说，你只需要在每个节点配置好了集群名称，节点名称，互相通信的节点会根据es自定义的服务发现协议去按照多播的方式来寻找网络上配置在同样集群内的节点。和其他的服务发现功能一样，es是支持多播和单播的。多播和单播的配置分别根据这几个参数：

|  |  |
| --- | --- |
| 1  2  3  4  5 | discovery.zen.ping.multicast.enabled: true  discovery.zen.fd.ping\_timeout: 100s  discovery.zen.ping.timeout: 100s  discovery.zen.minimum\_master\_nodes: 2  discovery.zen.ping.unicast.hosts: ["172.31.26.200"] |

* discovery.zen.ping.multicast.enabled  //这个设置把组播的自动发现给关闭了，为了防止其他机器上的节点自动连入。
* discovery.zen.fd.ping\_timeout和discovery.zen.ping.timeout是设置了节点与节点之间的连接ping时长
* discovery.zen.minimum\_master\_nodes //这个设置为了避免脑裂。比如3个节点的集群，如果设置为2，那么当一台节点脱离后，不会自动成为master。
* discovery.zen.ping.unicast.hosts //这个设置了自动发现的节点。
* action.auto\_create\_index: false //这个关闭了自动创建索引。为的也是安全考虑，否则即使是内网，也有很多扫描程序，一旦开启，扫描程序会自动给你创建很多索引。

多播是需要看服务器是否支持的，由于其安全性，其实现在基本的云服务（比如阿里云）是不支持多播的，所以即使你开启了多播模式，你也仅仅只能找到本机上的节点。单播模式安全，也高效，但是缺点就是如果增加了一个新的机器的话，就需要每个节点上进行配置才生效了。

**3、自动选举**

elasticsearch集群一旦建立起来以后，会选举出一个master，其他都为slave节点。但是具体操作的时候，每个节点都提供写和读的操作，你不论往哪个节点中做写操作，这个数据也会分配到集群上的所有节点中。

**如果是从节点挂掉了**

那么首先关心，数据会不会丢呢？不会。如果你开启了replicate，那么这个数据一定在别的机器上是有备份的。别的节点上的备份分片会自动升格为这份分片数据的主分片。

*这里要注意的是这里会有一小段时间的yellow状态时间*

**如果是主节点挂掉了**

从节点发现和主节点连接不上了，那么他们会自己决定再选举出一个节点为主节点。但是这里有个脑裂的问题，假设有5台机器，3台在一个机房，2台在另一个机房，当两个机房之间的联系断了之后，每个机房的节点会自己聚会，推举出一个主节点。这个时候就有两个主节点存在了，当机房之间的联系恢复了之后，这个时候就会出现数据冲突了

解决的办法就是设置参数：discovery.zen.minimum\_master\_nodes为3(超过一半的节点数)，那么当两个机房的连接断了之后，就会以大于等于3的机房的master为主，另外一个机房的节点就停止服务了

对于自动服务这里不难看出，如果把节点直接暴露在外面，不管怎么切换master，必然会有单节点问题。所以一般我们会在可提供服务的节点前面加一个负载均衡。

**4、Too many open files**

查看max\_file\_descriptors

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | curl http://localhost:9200/\_nodes/process\?pretty  {    "cluster\_name" : "elasticsearch",    "nodes" : {      "qAZYd8jbSWKxFAcOu9Ax9Q" : {        "name" : "Masque",        "transport\_address" : "127.0.0.1:9300",        "host" : "127.0.0.1",        "ip" : "127.0.0.1",        "version" : "2.2.1",        "build" : "d045fc2",        "http\_address" : "127.0.0.1:9200",        "process" : {          "refresh\_interval\_in\_millis" : 1000,          "id" : 31917,          "mlockall" : true        }      }    }  } |

然而并没有

|  |  |
| --- | --- |
| 1  2  3  4  5 | # ps -ef | grep 'Xms' | grep -v grep | awk '{print $2}'  31917    # cat /proc/31917/limits  | grep 'Max open files'  Max open files            102400               102400               files |

官方文档建议

Make sure to increase the number of open files descriptors on the machine (or for the user running elasticsearch). Setting it to 32k or even 64k is recommended.

最简单的做法, 在bin/elasticsearch文件开始的位置加入

|  |  |
| --- | --- |
| 1 | ulimit -n 102400 |

**5、设置合理的刷新时间**

建立的索引，不会立马查到，这是因为elasticsearch为near-real-time，需要配置index.refresh\_interval参数，默认是1s

|  |  |
| --- | --- |
| 1 | index.refresh\_interval：1s |

这样所有新建的索引都使用这个刷新频率

**6、大量unassigned shards**

其实刚搭完运行时就是status: yellow(所有主分片可用，但存在不可用的从分片), 只有一个节点, 主分片启动并运行正常, 可以成功处理请求, 但是存在unassigned\_shards, 即存在没有被分配到节点的从分片.(只有一个节点.....)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | curl -XGET http://localhost:9200/\_cluster/health\?pretty  {    "cluster\_name" : "elasticsearch",    "status" : "yellow",    "timed\_out" : false,    "number\_of\_nodes" : 1,    "number\_of\_data\_nodes" : 1,    "active\_primary\_shards" : 15,    "active\_shards" : 15,    "relocating\_shards" : 0,    "initializing\_shards" : 0,    "unassigned\_shards" : 15,    "delayed\_unassigned\_shards" : 0,    "number\_of\_pending\_tasks" : 0,    "number\_of\_in\_flight\_fetch" : 0,    "task\_max\_waiting\_in\_queue\_millis" : 0,    "active\_shards\_percent\_as\_number" : 50.0  } |

处理方式: 找台内网机器, 部署另一个节点， 再次检查集群健康, 发现unassigned\_shards减少, active\_shards增多，操作完后, 集群健康从yellow恢复到 green

**7、fix unassigned shards**

找出UNASSIGNED分片

|  |  |
| --- | --- |
| 1  2  3  4  5 | curl -s "http://localhost:9200/\_cat/shards" | grep UNASSIGNED  index                3 p UNASSIGNED  index                3 r UNASSIGNED  index                1 p UNASSIGNED  index                1 r UNASSIGNED |

查询得到master节点的唯一标识

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | curl 'localhost:9200/\_nodes/process?pretty'    {    "cluster\_name" : "elasticsearch",    "nodes" : {      "AfUyuXmGTESHXpwi4OExxx" : {        "name" : "Master", |

执行reroute(分多次, 变更shard的值为UNASSIGNED查询结果中编号, 上一步查询结果是1和3)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | curl -XPOST 'localhost:9200/\_cluster/reroute' -d '{          "commands" : [ {                "allocate" : {                    "index" : "pv-2015.05.22",                    "shard" : 1,                    "node" : "AfUyuXmGTESHXpwi4OExxx",                    "allow\_primary" : true                }              }          ]      }' |

**三、插件的安装**

集群安装成功之后，需要对集群中的索引数据、运行情况等信息进行查看，索引需要安装一些插件，方面后续工作

**1、head**

通过head，可以查看集群几乎所有信息，还能进行简单的搜索查询，观察自动恢复的情况等等

|  |  |
| --- | --- |
| 1 | ES\_HOME/bin/plugin -install mobz/elasticsearch-head |

安装成功之后访问 ： http://ip:9200/\_plugin/head/

比如：cluster health:green (2, 20) : 表示该集群目前处于健康状态，集群包含2台机器，索引总共20个分片。粗线绿框表示主分片，细线绿框为备份分片

**2、bigdesk**

 bigdesk是集群监控插件，通过该插件可以查看整个集群的资源消耗情况，cpu、内存、http链接等等

|  |  |
| --- | --- |
| 1 | ES\_HOME/bin/plugin -install lukas-vlcek/bigdesk |

安装完成之后输入：http://ip:9200/\_plugin/bigdesk/#nodes即可

可以查看单个节点的资源使用情况，包括JVM、Thread Pools、OS、Process、HTTP&Transport、Indice、File system

插件大全：http://www.searchtech.pro/elasticsearch-plugins