Hadoop 企业级调优手册

本文档来自公众号：**五分钟学大数据**

微信扫码关注

目录

一、HDFS 核心参数

1.1 NameNode 内存生产配置

1. NameNode 内存计算

每个文件块大概占用 150byte，一台服务器 128G 内存为例，能存储多少文件块呢？

128*128*1024*1024/150Byte≈9.1 亿

2. Hadoop2.x 系列， 配置 NameNode 内存

NameNode 内存默认 2000m，如果服务器内存 4G， NameNode 内存可以配置 3G。在 hadoop-env.sh 文件中配置如下。

```
HADOOP_NAMENODE_OPTS= Xmx 3072 m
```

3. Hadoop3.x 系列，配置 NameNode 内存

hadoop-env.sh 中描述 Hadoop 的内存是动态分配的

```
# The maximum amount of heap to use (Java -Xmx).  If no unit
# is provided, it will be converted to MB.  Daemons will
# prefer any Xmx setting in their respective _OPT variable.
# There is no default; the JVM will autoscale based upon machine
# memory size.
# export HADOOP_HEAPSIZE_MAX=

# The minimum amount of heap to use (Java -Xms).  If no unit
# is provided, it will be converted to MB.  Daemons will
# prefer any Xms setting in their respective _OPT variable.
# There is no default; the JVM will autoscale based upon machine
# memory size.
# export HADOOP_HEAPSIZE_MIN=
HADOOP_NAMENODE_OPTS= Xmx102400m
```

查看 NameNode 占用内存

```
[Tom@hadoop102 hadoop-3.1.3]$ jps
3136 JobHistoryServer
```

```
3200 Jps
2947 NodeManager
2486 NameNode
2622 DataNode
[Tom@hadoop102 hadoop-3.1.3]$ jps -heap 2486
Heap Configuration:
   MinHeapFreeRatio         = 40
   MaxHeapFreeRatio         = 70
   MaxHeapSize              = 478150656 (456.0MB)
```

查看 DataNode 占用内存

```
[Tom@hadoop102 hadoop-3.1.3]$ jmap -heap 2622
Heap Configuration:
   MinHeapFreeRatio         = 40
   MaxHeapFreeRatio         = 70
   MaxHeapSize              = 478150656 (456.0MB)
```

查看发现 hadoop102 上的 NameNode 和 DataNode 占用内存都是自动分配的，且相等。不是很合理。

经验参考：

https://docs.cloudera.com/documentation/enterprise/6/release-notes/topics/rg_hardware_requirements.html#concept_fzz_dq4_gbb

NameNode	• Minimum: 1 GB (for proof-of-concept deployments) • Add an additional 1 GB for each additional 1,000,000 blocks Snapshots and encryption can increase the required heap memory. See Sizing NameNode Heap Memory Set this value using the **Java Heap Size of NameNode in Bytes** HDFS configuration property.

namenode最小值1G，每增加1000000个block，增加1G内存

DataNode	Minimum: 4 GB Increase the memory for higher replica counts or a higher number of blocks per DataNode. When increasing the memory, Cloudera recommends an additional 1 GB of memory for every 1 million replicas above 4 million on the DataNodes. For example, 5 million replicas require 5 GB of memory. Set this value using the **Java Heap Size of DataNode in Bytes** HDFS configuration property.

datanode最小值4G，block数，或者副本数升高，都应该调大datanode的值。

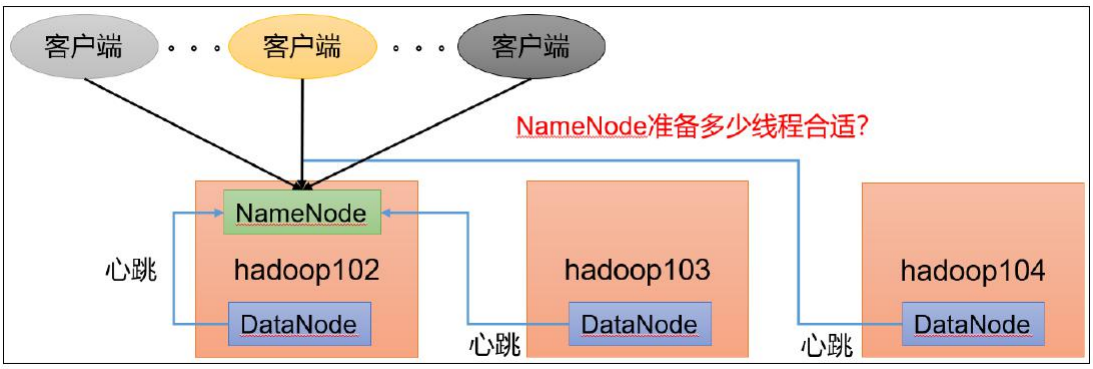
一个datanode上的副本总数低于4000000，调为4G，超过4000000，每增加1000000，增加1G

具体修改：hadoop-env.sh

```
export HDFS_NAMENODE_OPTS="-Dhadoop.security.logger=INFO,RFAS-Xmx1024m"
```

```
export HDFS_DATANODE_OPTS="-Dhadoop.security.logger=ERROR,RFAS-Xmx1024m"
```

1.2 NameNode 心跳并发配置



NameNode 有一个工作线程池，用来处理不同 DataNode 的并发心跳以及客户端并发的元数据操作 。对于大集群或者有大量客户端的集群来说，通常需要增大该参数 。 默认值是 10 。

```
<property>
 <name>dfs.namenode.handler.count</name>
 <value>21</value>
</property>
```

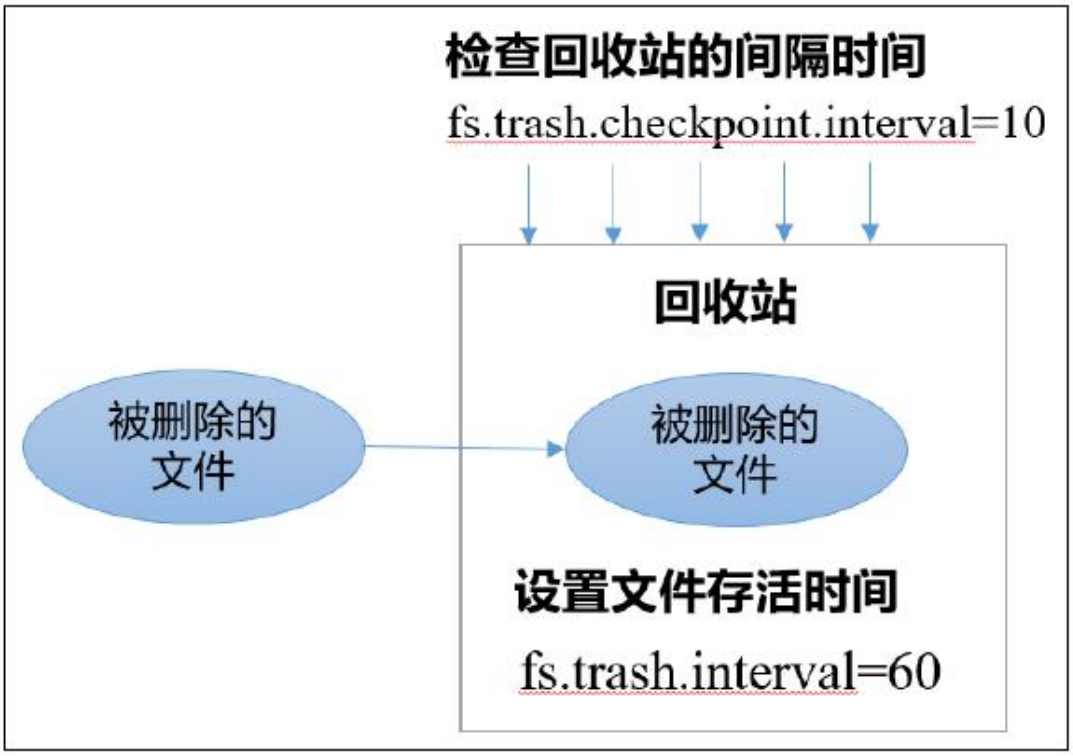
企业经验：dfs.namenode.handler.count=20 × logeClustersize ，比如集群规模（DataNode 台数）为 3 台时，此参数设置为 21。可通过简单的 python 代码计算该值，代码如下。

```
[Tom@hadoop102 hadoop-3.1.3]$ python
Python 2.7.5 (default, Oct 14 2020, 14:45:30)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> print int(20*math.log(3))
21
>>> quit()
```

1.3 开启回收站配置

开启回收站功能，可以将删除的文件在不超时的情况下，恢复原数据，起到防止误删除、备份等作用。

1. 回收站工作机制



2. 开启回收站功能参数说明

默认值 `fs.trash.interval = 0`，0 表示禁用回收站，其他值表示设置文件的存活时间。

默认值 fs.trash.checkpoint.interval = 0，检查回收站的间隔时间。如果该值为 0，则该值设置和 fs.trash.interval 的参数值相等。

要求 fs.trash.checkpoint.interval <= fs.trash.interval。

3. 启用回收站

修改 core-site.xml 配置，垃圾回收时间为 1 分钟。

```
<property>
 <name>fs.trash.interval</name>
 <value>1</value>
</property>
```

4. 查看回收站

回收站目录在 HDFS 集群中的路径：/user/Tom/.Trash/….

5. 注意：通过网页上直接删除的文件也不会走回收站。

6. 通过程序删除的文件不会经过回收站，需要调用 moveToTrash() 才进入回收站

```
Trash trash = N ew Trash(conf);

trash.moveToTrash(path);
```

7. 只有在命令行利用 hadoop fs -rm 命令删除的文件才会走回收站。

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -rm -r /input

2021-06-24 18:20:36,515 INFO fs.TrashPolicyDefault: Moved:
'hdfs://hadoop102:8020/input' to trash at:
hdfs://hadoop102:8020/user/Tom/.Trash/Current/input
```

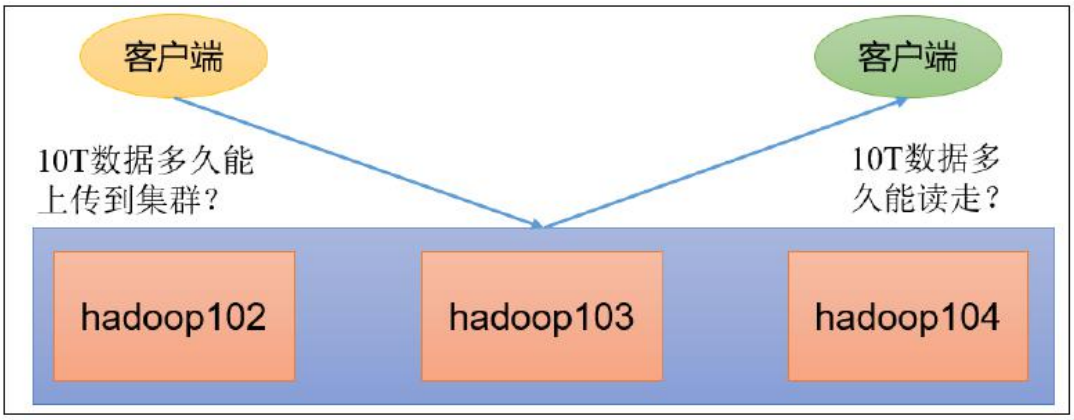
（8）恢复回收站数据

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -mv /user/Tom/.Trash/Current/input /input
```

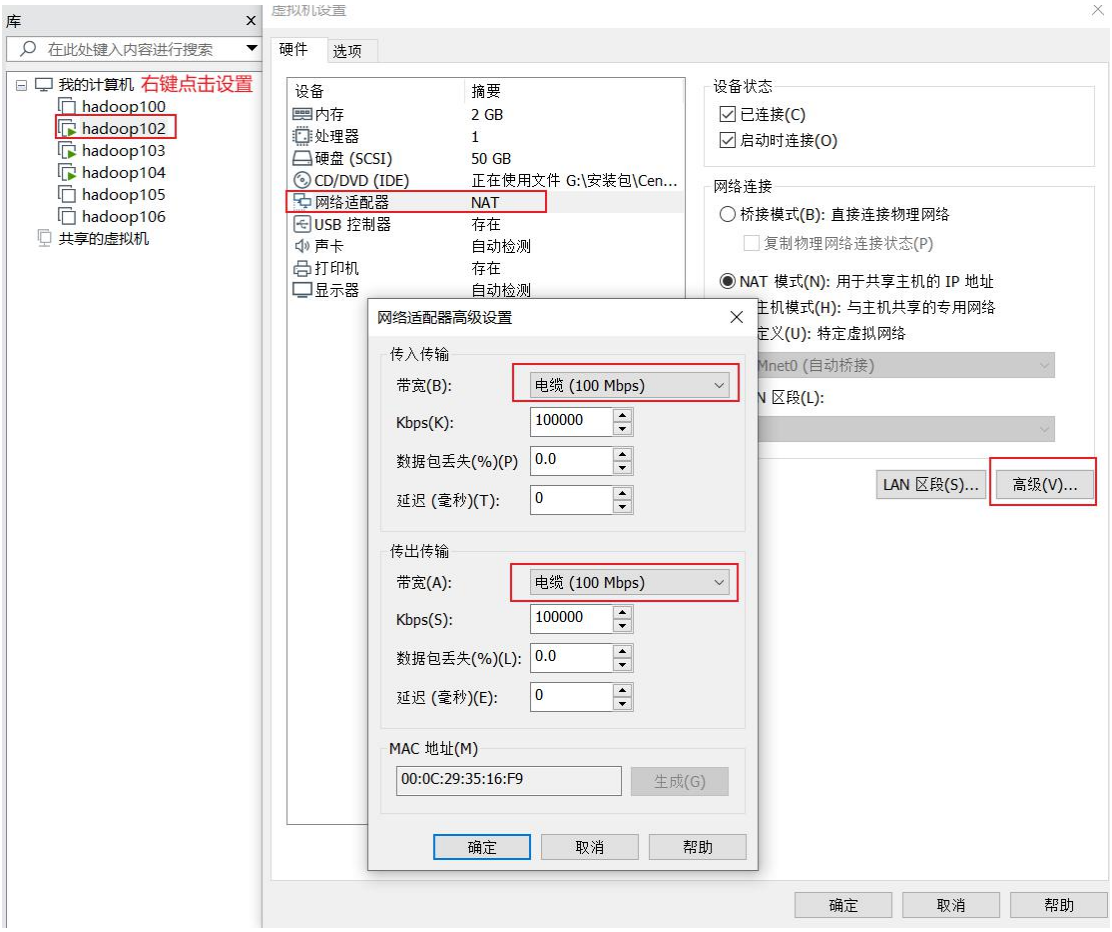
二、HDFS 集群压测

在企业中非常关心每天从 Java 后台拉取过来的数据，需要多久能上传到集群？消费者关心多久能从 HDFS 上拉取需要的数据？

为了搞清楚 HDFS 的读写性能，生产环境上非常需要对集群进行压测。



HDFS 的读写性能主要受网络和磁盘影响比较大。为了方便测试，将 hadoop102、hadoop103、 hadoop104 虚拟机网络都设置为 100mbps。



100Mbps 单位是 bit；10M/s 单位是 byte；1byte=8bit；100Mbps/8=12.5M/s。

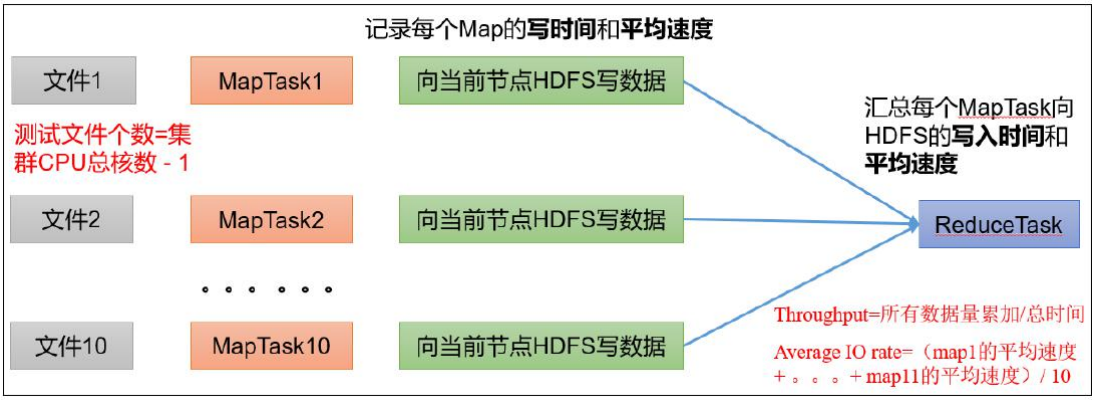
测试网速：来到 hadoop102 的 /opt/software 目录， 创建一个

```
[Tom@hadoop102 software]$ python -m SimpleHTTPServer
```

```
Serving HTTP on 0.0.0.0 port 8000 ...
```

2.1 测试 HDFS 写性能

写测试底层原理



测试内容： 向 HDFS 集群写 5 个 128M 的文件

注意： nrFiles n 为生成 mapTask 的数量，生产环境一般可通过 hadoop103:8088 查看 CPU 核数，设置为（CPU 核数 - 1）

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.3-tests.jar TestDFSIO
-write -nrFiles 5 -fileSize 128MB

2021-06-24 21:58:25,548 INFO sasl.SaslDataTransferClient: SASL encryption trust check:
localHostTrusted = false, remoteHostTrusted = false

2021-06-24 21:58:25,568 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write

2021-06-24 21:58:25,568 INFO fs.TestDFSIO:            Date & time: Thu Jun 24 21:58:25
CST 2021

2021-06-24 21:58:25,568 INFO fs.TestDFSIO:        Number of files: 5

2021-06-24 21:58:25,568 INFO fs.TestDFSIO: Total MBytes processed: 640

2021-06-24 21:58:25,568 INFO fs.TestDFSIO:      Throughput mb/sec: 0.88

2021-06-24 21:58:25,568 INFO fs.TestDFSIO: Average IO rate mb/sec: 0.88

2021-06-24 21:58:25,568 INFO fs.TestDFSIO:  IO rate std deviation: 0.04

2021-06-24 21:58:25,568 INFO fs.TestDFSIO:     Test exec time sec: 246.54
```

```
2021-06-24 21:58:25,568 INFO fs.TestDFSIO:
```

- Number of files：生成 mapTask 数量，一般是集群中 CPU 核数 -1，我们测试虚拟机就按照实际的物理内存 -1 分配即可
- Total MBytes processed：单个 map 处理的文件大小
- Throughput mb/sec：单个 mapTak 的吞吐量
  计算方式：处理的总文件大小 / 每一个 mapTask 写数据的时间累加
  集群整体吞吐量：生成 mapTask 数量 * 单个 mapTak 的吞吐量
- Average IO rate mb/sec：平均 mapTak 的吞吐量
  计算方式：每个 mapTask 处理文件大小 / 每一个 mapTask 写数据的时间全部相加除以 task 数量
- IO rate std deviation：方差、反映各个 mapTask 处理的差值，越小越均衡

注意：如果测试过程中，出现异常，可以在 yarn-site.xml 中设置虚拟内存检测为 false。然后分发配置并重启集群。

```
<!--是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是true -->
<property>
 <name>yarn.nodemanager.vmem-check-enabled</name>
 <value>false</value>
</property>
```

测试结果分析

由于副本 1 就在本地，所以该副本不参与测试

一共参与测试的文件：

5 个文件 * 2 个副本 = 10 个

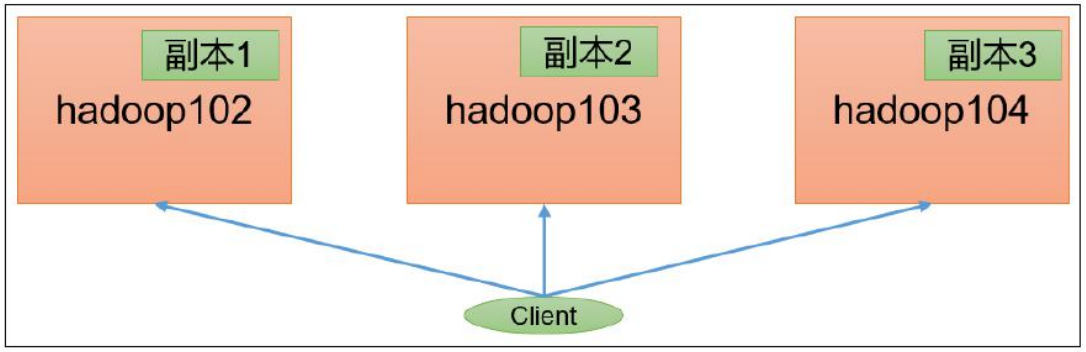
压测后的速度： 0.88

实测速度：0.88M/s * 10 个文件 ≈ 8.8M/s

三台服务器的带宽：12.5 + 12.5 + 12.5 ≈ 30m/s

所有网络资源没有用满。

如果实测速度远远小于网络，并且实测速度不能满足工作需求，可以考虑采用固态硬盘或者增加磁盘个数。



如果客户端不在集群节点，那就三个副本都参与计算

2.2 测试 HDFS 读性能

测试内容：读取 HDFS 集群 5 个 128M 的文件

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.3-tests.jar TestDFSIO
-read -nrFiles 5 -fileSize 128MB
```

```
2021-06-25 17:34:41,179 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
2021-06-25 17:34:41,181 INFO fs.TestDFSIO:            Date & time: Fri Jun 25 17:34:41
CST 2021
2021-06-25 17:34:41,182 INFO fs.TestDFSIO:        Number of files: 5
2021-06-25 17:34:41,182 INFO fs.TestDFSIO: Total MBytes processed: 640
2021-06-25 17:34:41,182 INFO fs.TestDFSIO:      Throughput mb/sec: 4.6
2021-06-25 17:34:41,182 INFO fs.TestDFSIO: Average IO rate mb/sec: 4.74
2021-06-25 17:34:41,182 INFO fs.TestDFSIO:  IO rate std deviation: 0.93
2021-06-25 17:34:41,182 INFO fs.TestDFSIO:     Test exec time sec: 82.47
2021-06-25 17:34:41,182 INFO fs.TestDFSIO:
```

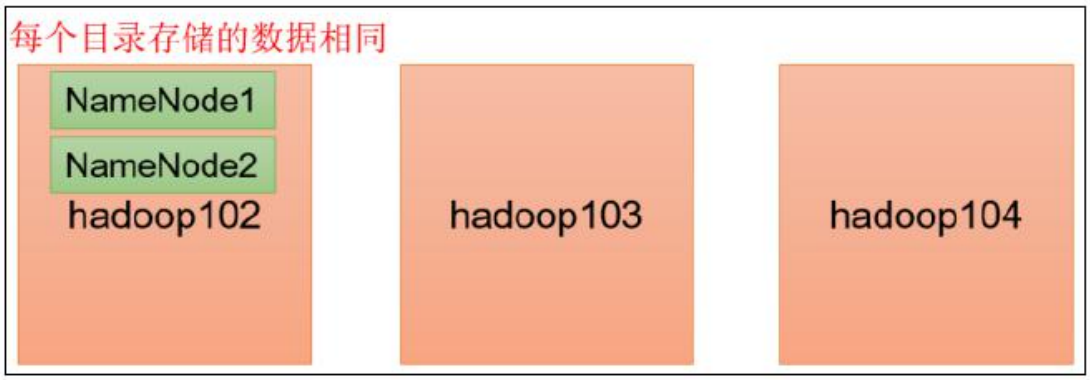
删除测试生成数据

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.1.3-tests.jar TestDFSIO
-clean
```

三、HDFS 多目录

3.1 NameNode 多目录配置

NameNode 的本地目录可以配置成多个，且每个目录存放内容相同，增加了可靠性。



具体配置如下

（1）在 hdfs-site.xml 文件中添加如下内容

```
<property>
 <name>dfs.namenode.name.dir</name>
 <value>file://${hadoop.tmp.dir}/dfs/name1,file://${hadoop.tmp.dir}/dfs/name2</valu
e>
</property>
```

注意：因为每台服务器节点的磁盘情况不同，所以这个配置配完之后，可以选择不分发。

（2）停止集群，删除三台节点的 data 和 logs 中所有数据。

```
[Tom@hadoop102 hadoop 3.1.3 ]$ rm rf data/ logs/
[Tom@hadoop103 hadoop 3.1.3 ]$ rm rf data/ logs/
[Tom@hadoop104 hadoop 3.1.3 ]$ rm rf data/ logs/
```

（3）格式化集群并启动。

```
[Tom@hadoop102 hadoop 3.1.3 ]$ bin/hdfs namenode format

[Tom@hadoop102 hadoop 3.1.3 ]$ sbin/start dfs.sh
```

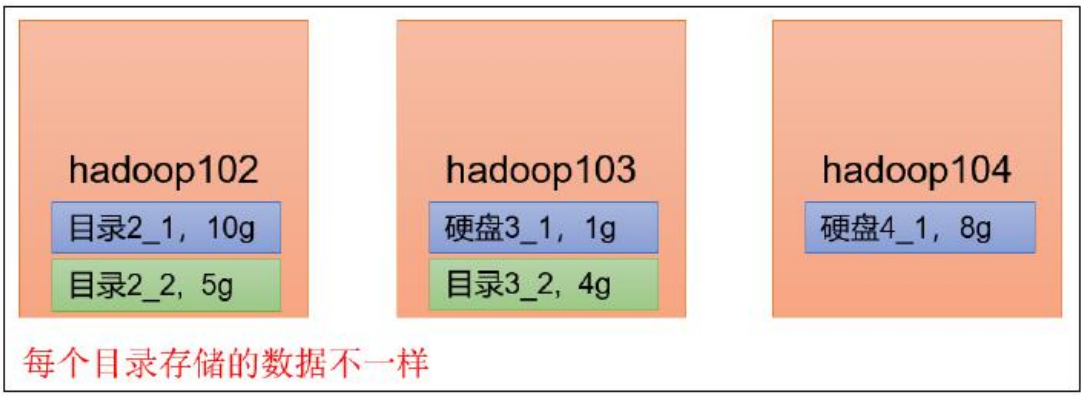
查看结果

```
[Tom@hadoop102 dfs]$ ll

总用量 12

drwx------. 3 Tom Tom 4096 12 月 11 08:03 data

drwxrwxr-x. 3 Tom Tom 4096 12 月 11 08:03 name1

drwxrwxr-x. 3 Tom Tom 4096 12 月 11 08:03 name2
```

检查 name1 和 name2 里面的内容，发现一模一样。

3.2 DataNode 多目录配置

DataNode 可以配置成多个目录，每个目录存储的数据不一样（数据不是副本）



具体配置如下

（1）在 hdfs-site.xml 文件中添加如下内容

```
<property>
 <name>dfs.datanode.data.dir</name>
 <value>file://${hadoop.tmp.dir}/dfs/data1,file://${hadoop.tmp.dir}/dfs/data2</valu
e>
</property>
```

查看结果

```
[Tom@hadoop102 dfs]$ ll
总用量 12
drwx------. 3 Tom Tom 4096 4 月 4 14:22 data1
drwx------. 3 Tom Tom 4096 4 月 4 14:22 data2
drwxrwxr-x. 3 Tom Tom 4096 12 月 11 08:03 name1
drwxrwxr-x. 3 Tom Tom 4096 12 月 11 08:03 name2
```

向集群上传一个文件，再次观察两个文件夹里面的内容发现不一致 （一个有数一个没有）

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -put wcinput/word.txt /
```

3.3 集群数据均衡之磁盘间数据均衡

生产环境，由于硬盘空间不足，往往需要增加一块硬盘。刚加载的硬盘没有数据时，可以执行磁盘数据均衡命令。（ Hadoop3.x 新特性）

（1）生成均衡计划（我只有一块磁盘，不会生成计划）

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs diskbalancer -plan hadoop102
```

（2）执行均衡计划

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs diskbalancer -execute hadoop102.plan.json
```

（3）查看当前均衡任务的执行情况

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs diskbalancer -query hadoop102
```

（4）取消均衡任务

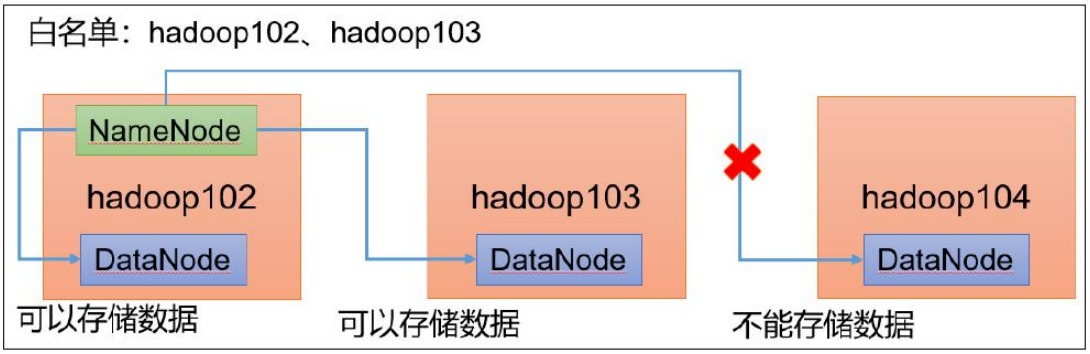
```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs diskbalancer -cancel hadoop102.plan.json
```

四、HDFS 集群扩容及缩容

4.1 添加白名单

白名单：表示在白名单的主机 IP 地址可以用来存储数据。

企业中：配置白名单，可以尽量防止黑客恶意访问攻击。



配置白名单步骤如下：

在 NameNode 节点的 /opt/module/hadoop-3.1.3/etc/hadoop 目录下分别创建 whitelist 和 blacklist 文件

（1）创建白名单

```
[Tom@hadoop102 hadoop]$ vim whitelist
```

在 whitelist 中添加下主机名称，假如集群正常工作的节点为 102 103

```
hadoop102
```

```
hadoop103
```

（2）创建黑名单，保持空的就可以

```
[Tom@hadoop102 hadoop]$ touch blacklist
```

在 hdfs-site.xml 配置文件中增加 dfs.hosts 配置参数

```
<!--白名单-->
<property>
 <name>dfs.hosts</name>
 <value>/opt/module/hadoop-3.1.3/etc/hadoop/whitelist</value>
</property>

<!--黑名单-->
<property>
 <name>dfs.hosts.exclude</name>
 <value>/opt/module/hadoop-3.1.3/etc/hadoop/blacklist</value>
</property>
```

分发配置文件 whitelist hdfs-site.xml

```
[Tom@hadoop104 hadoop]$ xsync hdfs site.xml whitelist
```

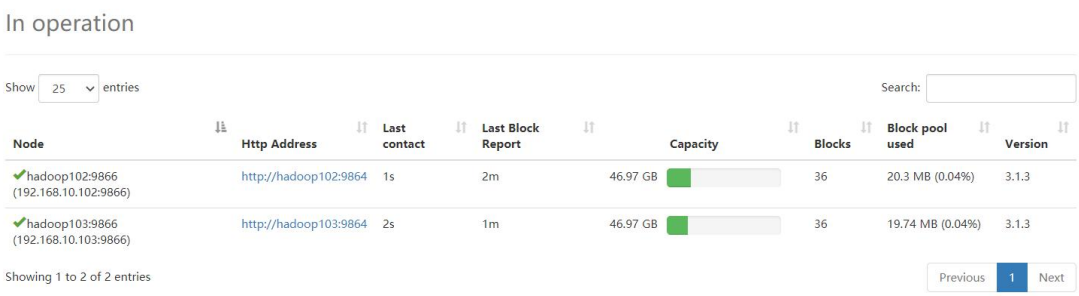
第一次添加白名单必须重启集群，不是第一次，只需要刷新 NameNode 节点即可

```
[Tom@hadoop102 hadoop 3.1.3]$ myhadoop.sh stop
```

```
[Tom@hadoop102 hadoop 3.1.3]$ myhadoop.sh start
```

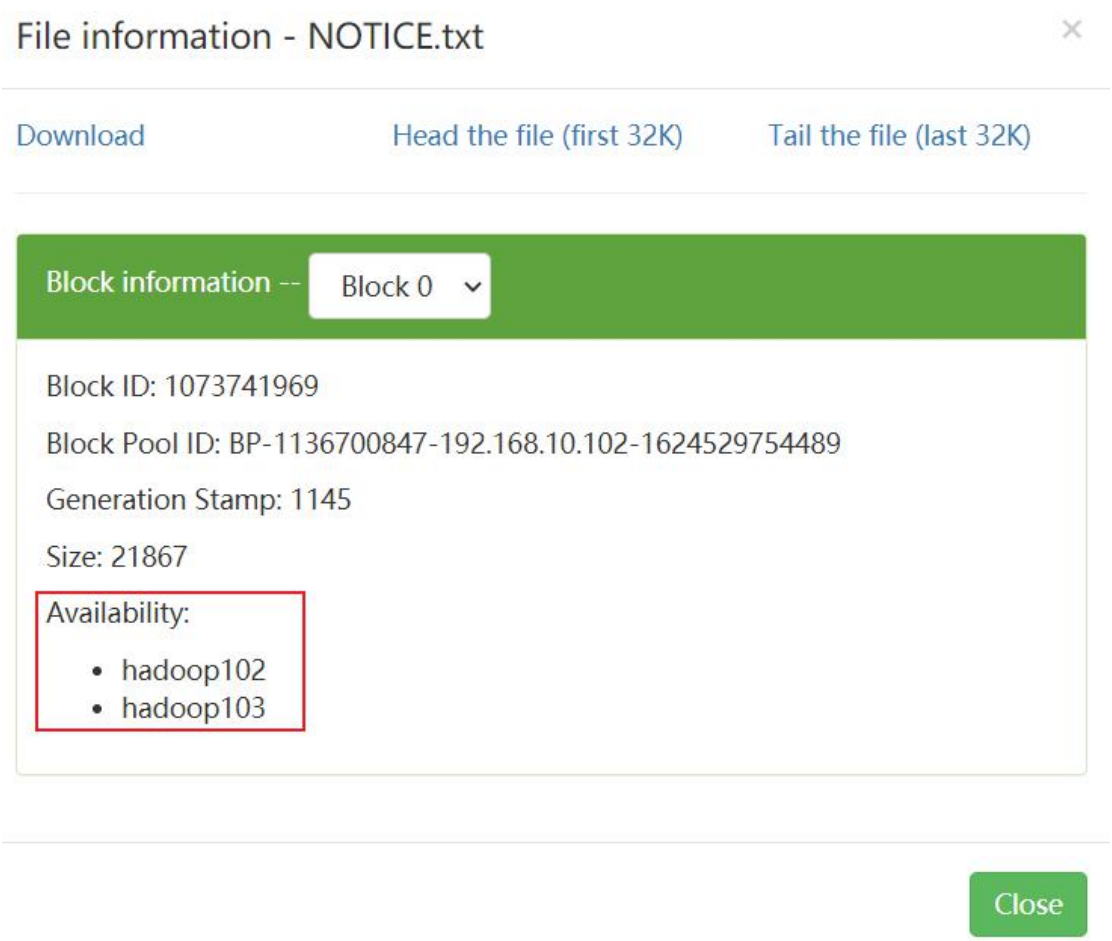
在 web 浏览器上查看 DN，

http://hadoop102:9870/dfshealth.html#tab-datanode



在 hadoop104 上执行上传数据数据失败

```
[Tom@hadoop104 hadoop-3.1.3]$ hadoop fs -put NOTICE.txt /
```



二次修改白名单，增加 hadoop104

```
[Tom@hadoop102 hadoop]$ vim whitelist
```

修改为如下内容

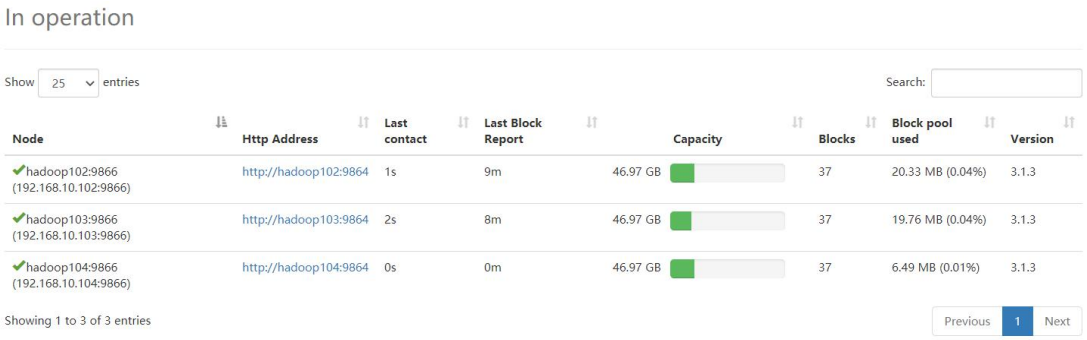
```
hadoop102

hadoop103

hadoo p104
```

刷新 NameNode

```
[Tom@hadoop102 hadoop 3.1.3]$ hdfs dfsadmin refreshNodes
```

```
Refresh nodes successful
```

在 web 浏览器上查看 DN http://hadoop102:9870/dfshealth.html#tab-datanode



4.2 服役新服务器

需求

随着公司业务的增长，数据量越来越大，原有的数据节点的容量已经不能满足存储数据的需求，需要在原有集群基础上动态添加新的数据节点。

环境准备

（1）在 hadoop100 主机上再克隆一台 hadoop105 主机

（2）修改 IP 地址和主机名称

```
[root@hadoop105 ~]# vim /etc/sysconfig/network-scripts/ifcfg-ens33
```

```
[root@hadoop105 ~]# vim /etc/hostname
```

（3）拷贝 hadoop102 的 /opt/module 目录和 /etc/profile.d/my_env.sh 到 hadoop105

```
[Tom@hadoop102 opt]$ scp-r module/* Tom@hadoop105:/opt/module/
```

```
[Tom@hadoop102 opt]$ sudo scp/etc/profile.d/my_env.sh
root@hadoop105:/etc/profile.d/my_env.sh
```

```
[Tom@hadoop105 hadoop-3.1.3]$ source /etc/profile
```

（4）删除 hadoop105 上 Hadoop 的历史数据， data 和 logs 数据

```
[Tom@hadoop105 hadoop 3.1.3]$ rm rf data/ logs/
```

（5）配置 hadoop102 和 hadoop103 到 hadoop105 的 ssh 无密登录

```
[ hadoop102 .ssh]$ ssh copy id hadoop105
```

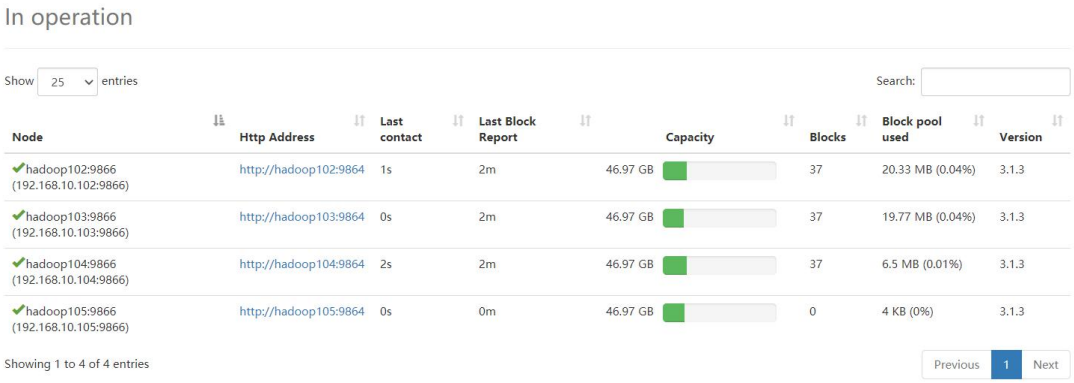
```
[ hadoop103 .ssh]$ ssh copy id hadoop105
```

服役新节点具体步骤

（1）直接启动 DataNode 即可关联到集群

```
[Tom@hadoop105 hadoop-3.1.3]$ hdfs --daemon start datanode
```

```
[Tom@hadoop105 hadoop-3.1.3]$ yarn --daemon start nodemanager
```



在白名单中增加新服役的服务器

（1）在白名单 whitelist 中增加 hadoop104、 hadoop105，并重启集群

```
[Tom@hadoop102 hadoop]$ vim whitelist
```

修改为如下内容

```
hadoop102
```

```
hadoop103
```

```
hadoop104
```

```
hadoop105
```

（2）分发

```
[Tom@hadoop102 hadoop]$ xsync whitelist
```

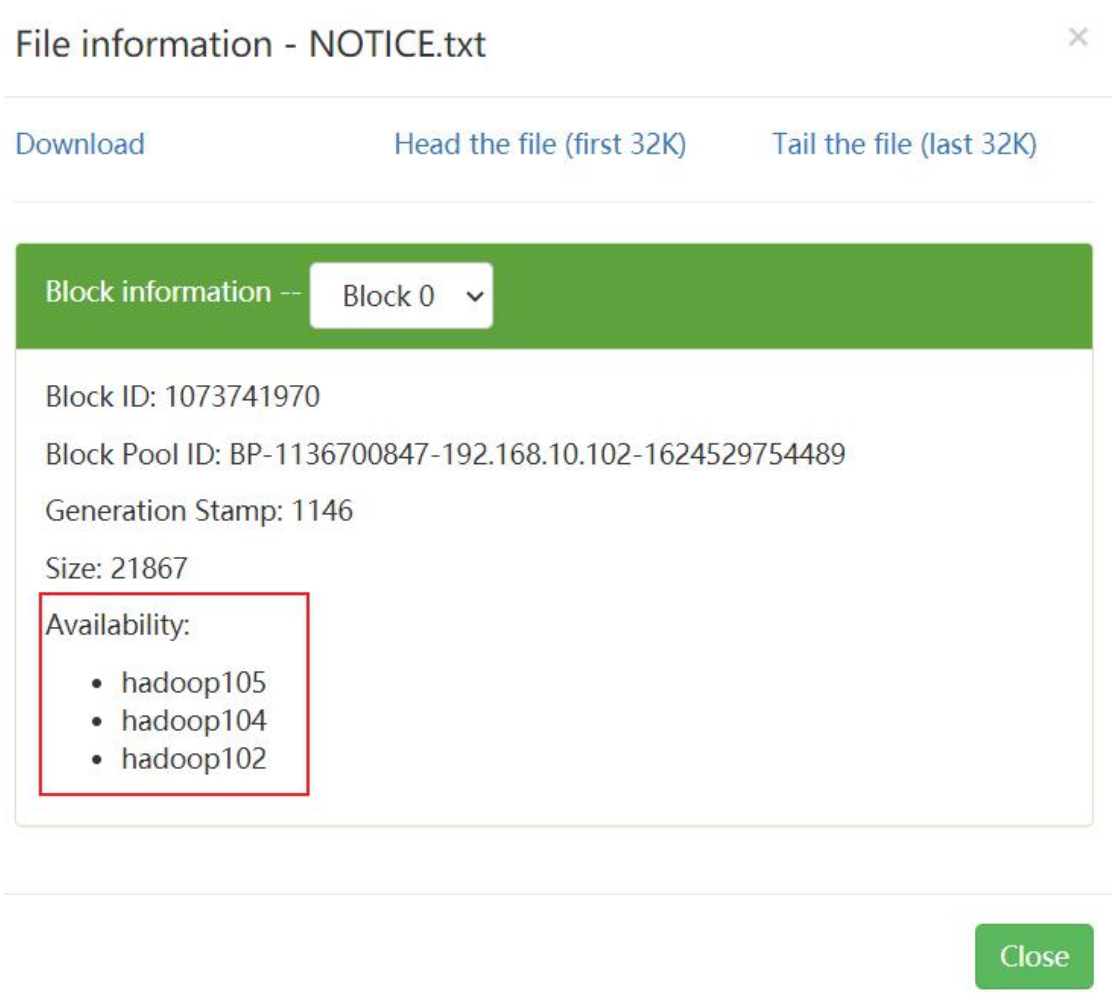
（3）刷新 NameNode

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs dfsadmin -refreshNodes
```

```
Refresh nodes successful
```

在 hadoop105 上上传文件

```
[Tom@hadoop105 hadoop-3.1.3]$ hadoop fs -put /opt/module/hadoop-3.1.3/LICENSE.txt /
```



4.3 服务器间数据均衡

企业经验

在企业开发中，如果经常在 hadoop102 和 hadoop104 上提交任务，且副本数为 2，由于数据本地性原则，就会导致 hadoop102 和 hadoop104 数据过多，hadoop103 存储的数据量小。

另一种情况，就是新服役的服务器数据量比较少，需要执行集群均衡命令。

开启数据均衡命令

```
[Tom@hadoop105 hadoop 3.1.3]$ sbin/start balancer.sh
```

```
threshold 10
```

对于参数 10，代表的是集群中各个节点的磁盘空间利用率相差不超过 10%，可根据实际情况进行调整。

停止数据均衡命令

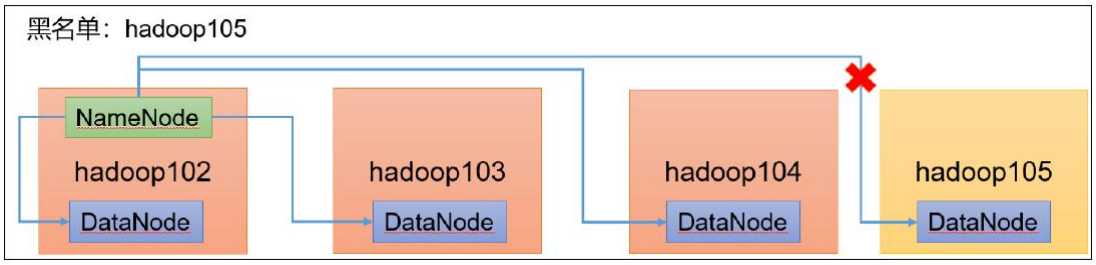
```
[Tom@hadoop105 hadoop-3.1.3]$ sbin/stop-balancer.sh
```

注意 ：由于 HDFS 需要启动单独的 Rebalance Server 来执行 Rebalance 操作，所以尽量不要在 NameNode 上执行 start-balancer.sh，而是找一台比较空闲的机器。

4.4 黑名单退役服务器

黑名单：表示在黑名单的主机 IP 地址不可以用来存储数据。

企业中：配置黑名单，用来退役服务器。



黑名单配置步骤如下：

编辑 /opt/module/hadoop-3.1.3/etc/hadoop 目录下的 blacklist 文件

```
[Tom@hadoop102 hadoop vim blacklist
```

添加如下主机名称（要退役的节点）

hadoop105

注意：如果白名单中没有配置，需要在 hdfs-site.xml 配置文件中增加 dfs.hosts 配置参数

```
<!--黑名单-->
<property>
 <name>dfs.hosts.exclude</name>
 <value>/opt/module/hadoop-3.1.3/etc/hadoop/blacklist</value>
</property>
```

分发配置文件 blacklist，hdfs-site.xml

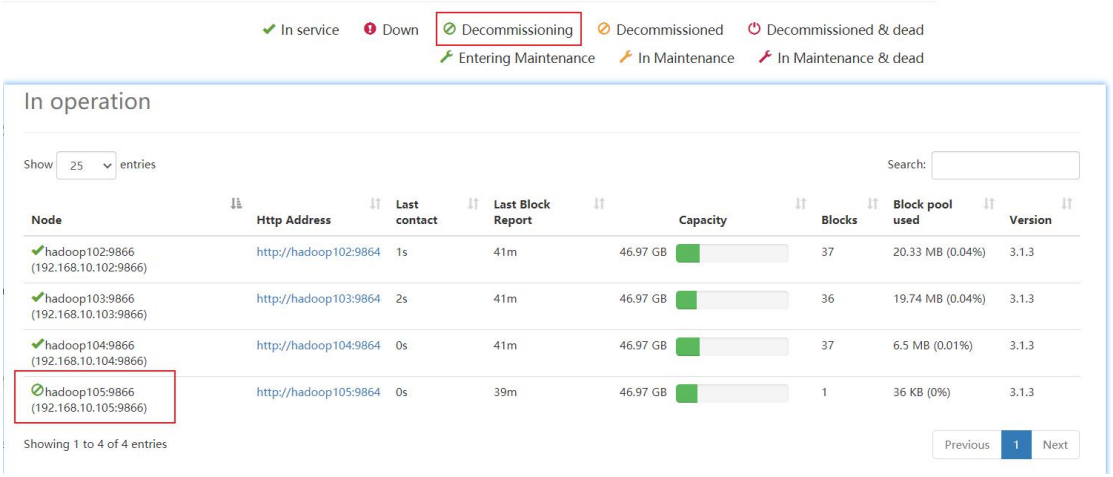
```
[Tom@hadoop104 hadoop]$ xsync hdfs-site.xml blacklist
```

第一次添加黑名单必须重启集群，不是第一次，只需要刷新 NameNode 节点即可

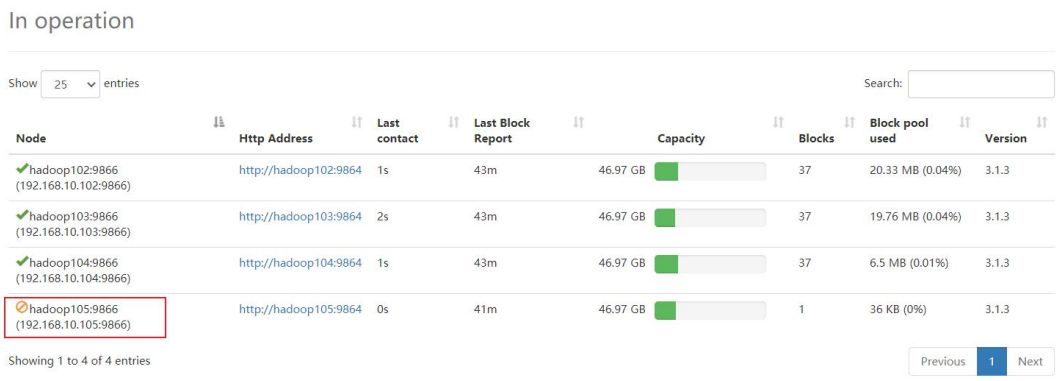
```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs dfsadmin -refreshNodes
```

```
Refresh nodes successful
```

检查 Web 浏览器 ，退役节点的状态为 decommission in progress（退役中，说明数据节点正在复制块到其他节点）



等待退役节点状态为 decommissioned（所有块已经复制完成 ），停止该节点及节点资源管理器。 注意 ：如果副本数是 3 服役的节点小于等于 3，是不能退役成功的，需要修改副本数后才能退役。




```
[Tom@hadoop105 hadoop-3.1.3]$ hdfs --daemon stop datanode
```

```
[Tom@hadoop105 hadoop-3.1.3]$ yarn--daemon stop nodemanager
```

如果数据不均衡，可以用命令实现集群的再平衡

```
[Tom@hadoop102 hadoop-3.1.3]$ sbin/start-balancer.sh -threshold 10
```

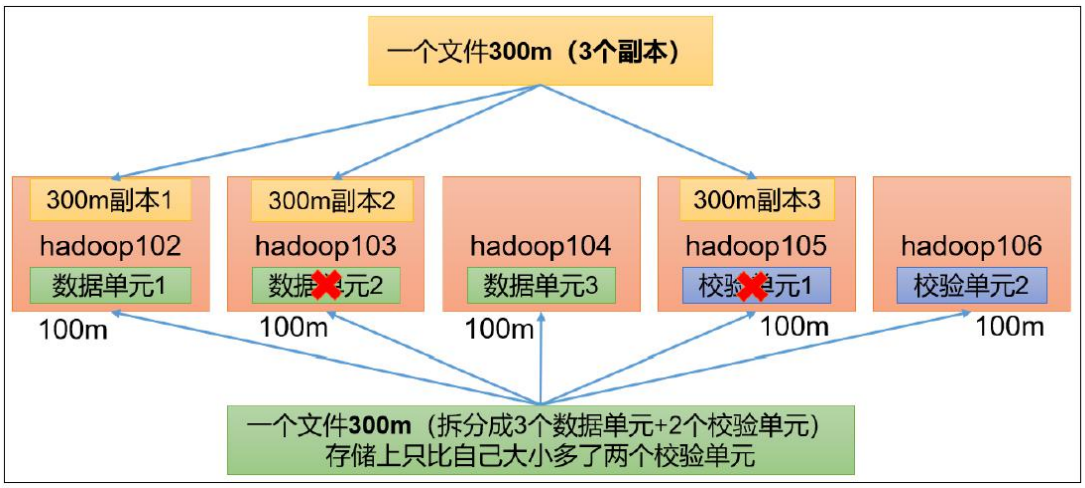
五、HDFS 存储优化

注：演示纠删码和异构存储需要一共 5 台虚拟机。尽量拿另外一套集群。提前准备 5 台服务器的集群。

5.1 纠删码

5.1.1 纠删码原理

HDFS 默认情况下，一个文件有 3 个副本，这样提高了数据的可靠性，但也带来了 2 倍的冗余开销。 Hadoop3.x 引入了纠删码， 采用计算的方式， 可以节省约 50%左右的存储空间。



纠删码操作相关的命令

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs ec
Usage: bin/hdfs ec [COMMAND]
          [-listPolicies]
          [-addPolicies -policyFile <file>]
          [-getPolicy -path <path>]
          [-removePolicy -policy <policy>]
          [-setPolicy -path <path> [-policy <policy>] [-replicate]]
          [-unsetPolicy -path <path>]
          [-listCodecs]
          [-enablePolicy -policy <policy>]
          [-disablePolicy -policy <policy>]
          [-help <command-name>]
```

查看当前支持的纠删码策略

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs ec -listPolicies
Erasure Coding Policies:
ErasureCodingPolicy=[Name=RS-10-4-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits=10,
numParityUnits=4]], CellSize=1048576, Id=5], State=DISABLED
```

```
ErasureCodingPolicy=[Name=RS-3-2-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits=3, numParityUnits=2]], CellSize=1048576, Id=2], State=DISABLED

ErasureCodingPolicy=[Name=RS-6-3-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits=6, numParityUnits=3]], CellSize=1048576, Id=1], State=ENABLED

ErasureCodingPolicy=[Name=RS-LEGACY-6-3-1024k, Schema=[ECSchema=[Codec=rs-legacy, numDataUnits=6, numParityUnits=3]], CellSize=1048576, Id=3], State=DISABLED

ErasureCodingPolicy=[Name=XOR-2-1-1024k,Schema=[ECSchema=[Codec=xor,numDataUnits=2, numParityUnits=1]], CellSize=1048576, Id=4], State=DISABLED
```

纠删码策略解释

（1）RS-3-2-1024k：使用 RS 编码，每 3 个数据单元，生成 2 个校验单元，共 5 个单元，也就是说：这 5 个单元中，只要有任意的 3 个单元存在（不管是数据单元还是校验单元，只要总数=3），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

（2）RS-10-4-1024k：使用 RS 编码，每 10 个数据单元（ cell），生成 4 个校验单元，共 14 个单元，也就是说：这 14 个单元中，只要有任意的 10 个单元存在 （不管是数据单元还是校验单元，只要总数 =10），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

（3）RS-6-3-1024k：使用 RS 编码，每 6 个数据单元，生成 3 个校验单元，共 9 个单元，也就是说：这 9 个单元中，只要有任意的 6 个单元存在（不管是数据单元还是校验单元，只要总数 =6），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

（4）RS-LEGACY-6-3-1024k：策略和上面的 RS-6-3-1024k 一样，只是编码的算法用的是 rs-legacy。

（5）XOR-2-1-1024k：使用 XOR 编码（速度比 RS 编码快），每 2 个数据单元，生成 1 个校验单元，共 3 个单元，也就是说：这 3 个单元中，只要有任意的 2 个单元存在（不管是数据单元还是校验单元，只要总数 =2），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

5.1.2 纠删码案例实操

纠删码策略是给具体一个路径设置 。所有往此路径下存储的文件，都会执行此策略。默认只开启对 RS-6-3-1024k 策略的支持 ，如要使用别的策略需要提前启用 。

需求：将 /input 目录设置为 RS-3-2-1024k 策略

具体步骤

（1）开启对 RS-3-2-1024k 策略的支持

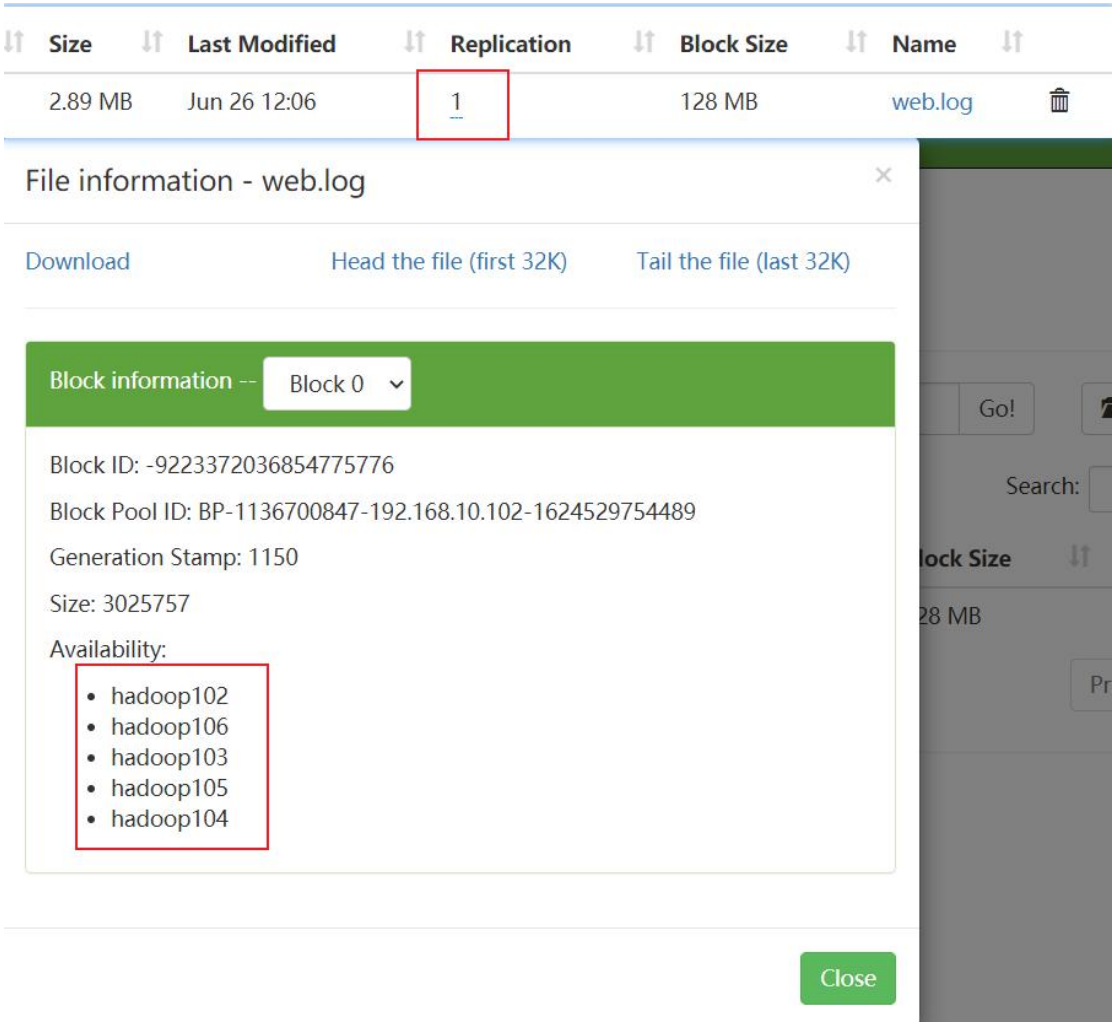
```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs ec -enablePolicy -policy RS-3-2-1024k
Erasure coding policy RS-3-2-1024k is enabled
```

（2）在 HDFS 创建目录，并设置 RS-3-2-1024k 策略

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs dfs -mkdir /input
[Tom@hadoop102 hadoop-3.1.3]$ hdfs ec -setPolicy -path /input -policy RS-3-2-1024k
Set RS-3-2-1024k erasure coding policy on /input
```

（3）上传文件，并查看文件编码后的存储情况

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs dfs -put web.log /input
```



（4）查看存储路径的数据单元和校验单元，并作破坏实验

5.2 异构存储（冷热数据分离）

异构存储主要解决，不同的数据，存储在不同类型的硬盘中，达到最佳性能的问题。

关于存储类型

RAM_DISK：（内存镜像文件系统）

SSD：（SSD 固态硬盘）

DISK：（普通磁盘，在 HDFS 中，如果没有主动声明数据目录存储类型默认都是

DISK）

ARCHIVE：（没有特指哪种存储介质，主要的指的是计算能力比较弱而存储密度比较高的存储介质，用来解决数据量的容量扩增的问题，一般用于归档）

关于存储策略 说明：从 Lazy_Persist 到 Cold，分别代表了设备的访问速度从快到慢

策略ID	策略名称	副本分布	
15	Lazy_Persist	RAM_DISK:1，DISK:n-1	一个副本保存在内存RAM_DISK中，其余副本保存在磁盘中。
12	All_SSD	SSD:n	所有副本都保存在SSD中。
10	One_SSD	SSD:1，DISK:n-1	一个副本保存在SSD中，其余副本保存在磁盘中。
7	Hot(default)	DISK:n	Hot：所有副本保存在磁盘中，这也是默认的存储策略。
5	Warm	DSIK:1，ARCHIVE:n-1	一个副本保存在磁盘上，其余副本保存在归档存储上。
2	Cold	ARCHIVE:n	所有副本都保存在归档存储上。

5.2.1 异构存储 Shell 操作

（1）查看当前有哪些存储策略可以用

```
[Tom@hadoop102 ~]$ hdfs storagepolicies -listPolicies

Block Storage Policies:

        BlockStoragePolicy{PROVIDED:1, storageTypes=[PROVIDED, DISK],
creationFallbacks=[PROVIDED, DISK], replicationFallbacks=[PROVIDED, DISK]}

        BlockStoragePolicy{COLD:2, storageTypes=[ARCHIVE], creationFallbacks=[],
replicationFallbacks=[]}

        BlockStoragePolicy{WARM:5, storageTypes=[DISK, ARCHIVE],
creationFallbacks=[DISK, ARCHIVE], replicationFallbacks=[DISK, ARCHIVE]}

        BlockStoragePolicy{HOT:7, storageTypes=[DISK], creationFallbacks=[],
replicationFallbacks=[ARCHIVE]}

        BlockStoragePolicy{ONE_SSD:10, storageTypes=[SSD, DISK],
creationFallbacks=[SSD, DISK], replicationFallbacks=[SSD, DISK]}

        BlockStoragePolicy{ALL_SSD:12, storageTypes=[SSD], creationFallbacks=[DISK],
replicationFallbacks=[DISK]}

        BlockStoragePolicy{LAZY_PERSIST:15, storageTypes=[RAM_DISK, DISK],
creationFallbacks=[DISK], replicationFallbacks=[DISK]}
```

（2）为指定路径 （数据存储目录） 设置指定的存储策略

```
hdfs storagepolicies -setStoragePolicy -path xxx -policy xxx
```

（3）获取指定路径（数据存储目录或文件）的存储策略

```
hdfs storagepolicies -getStoragePolicy -path xxx
```

（4）取消存储策略；执行改命令之后该目录或者文件，以其上级的目录为准，如果是根目录，那么就是 HOT

```
hdfs storagepolicies -unsetStoragePolicy -path xxx
```

（5）查看文件块的分布

```
bin/hdfs fsck xxx -files -blocks -locations
```

（6）查看集群节点

```
hadoop dfsadmin -report
```

5.2.2 测试环境准备

测试环境描述

服务器规模：5 台

集群配置：副本数为 2，创建好带有存储类型的目录（提前创建）

集群规划：

节点	存储类型分配
hadoop102	RAM_DISK，SSD
hadoop103	SSD，DISK
hadoop104	DISK，RAM_DISK
hadoop105	ARCHIVE
hadoop106	ARCHIVE

配置文件信息

（1）为 hadoop102 节点的 hdfs-site.xml 添加如下信息

```
<property>
 <name>dfs.replication</name>
 <value>2</value>
</property>
<property>
 <name>dfs.storage.policy.enabled</name>
 <value>true</value>
</property>
<property>
 <name>dfs.datanode.data.dir</name>
 <value>[SSD]file:///opt/module/hadoop-3.1.3/hdfsdata/ssd,[RAM_DISK]file:///opt/mod
```

```
ule/hadoop-3.1.3/hdfsdata/ram_disk</value>
</property>
```

（2）为 hadoop103 节点的 hdfs-site.xml 添加如下信息

```
<property>
 <name>dfs.replication</name>
 <value>2</value>
</property>
<property>
 <name>dfs.storage.policy.enabled</name>
 <value>true</value>
</property>
<property>
 <name>dfs.datanode.data.dir</name>
 <value>[SSD]file:///opt/module/hadoop-3.1.3/hdfsdata/ssd,[DISK]file:///opt/module/
hadoop-3.1.3/hdfsdata/disk</value>
</property>
```

（3）为 hadoop104 节点的 hdfs-site.xml 添加如下信息

```
<property>
 <name>dfs.replication</name>
 <value>2</value>
</property>
<property>
 <name>dfs.storage.policy.enabled</name>
 <value>true</value>
</property>
<property>
 <name>dfs.datanode.data.dir</name>
 <value>[RAM_DISK]file:///opt/module/hdfsdata/ram_disk,[DISK]file:///opt/module/had
oop-3.1.3/hdfsdata/disk</value>
</property>
```

（4）为 hadoop105 节点的 hdfs-site.xml 添加如下信息

```
<property>
 <name>dfs.replication</name>
 <value>2</value>
</property>
<property>
 <name>dfs.storage.policy.enabled</name>
 <value>true</value>
</property>
<property>
 <name>dfs.datanode.data.dir</name>
```

```
 <value>[ARCHIVE]file:///opt/module/hadoop-3.1.3/hdfsdata/archive</value>
</property>
```

（5）为 hadoop106 节点的 hdfs-site.xml 添加如下信息

```
<property>
 <name>dfs.replication</name>
 <value>2</value>
</property>
<property>
 <name>dfs.storage.policy.enabled</name>
 <value>true</value>
</property>
<property>
 <name>dfs.datanode.data.dir</name>
 <value>[ARCHIVE]file:///opt/module/hadoop-3.1.3/hdfsdata/archive</value>
</property>
```

数据准备

（1）启动集群

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs namenode -format
```

```
[Tom@hadoop102 hadoop-3.1.3]$ myhadoop.sh start
```

（2）在 HDFS 上创建文件目录

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -mkdir /hdfsdata
```

（3）将文件资料上传

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -put /opt/module/hadoop-3.1.3/NOTICE.txt
/hdfsdata
```

5.2.3 HOT 存储策略案例

（1）最开始我们未设置存储策略的情况下，我们获取该目录的存储策略

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs storagepolicies -getStoragePolicy -path /hdfsdata
```

（2）我们查看上传的文件块分布

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs fsck /hdfsdata-files -blocks -locations
```

```
[DatanodeInfoWithStorage[192.168.10.104:9866,DS-0b133854-7f9e-48df-939b-5ca6482c5af
b,DISK],
DatanodeInfoWithStorage[192.168.10.103:9866,DS-ca1bd3b9-d9a5-4101-9f92-3da5f1baa28b,
DISK]]
```

未设置存储策略，所有文件块都存储在 DISK 下。 所以， 默认存储策略为 HOT。

5.2.4 WARM 存储策略测试

（1）接下来我们为数据降温

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs storagepolicies -setStoragePolicy -path /hdfsdata
-policy WARM
```

（2）再次查看文件块分布，我们可以看到文件块依然放在原处。

```
[atguigu@hadoop102 hadoop-3.1.3]$ hdfs fsck /hdfsdata-files -blocks -locations
```

（3）我们需要让他 HDFS 按照存储策略自行移动文件块

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs mover /hdfsdata
```

（4）再次查看文件块分布

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs fsck /hdfsdata -files -blocks -locations

[DatanodeInfoWithStorage[192.168.10.105:9866,DS-d46d08e1-80c6-4fca-b0a2-4a3dd7ec745
9,ARCHIVE],
DatanodeInfoWithStorage[192.168.10.103:9866,DS-ca1bd3b9-d9a5-4101-9f92-3da5f1baa28b,
DISK]]
```

文件块一半在 DISK，一半在 ARCHIVE，符合我们设置的 WARM 策略

5.2.5 COLD 策略测试

（1）我们继续将数据降温为 cold

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs storagepolicies -setStoragePolicy -path /hdfsdata
-policy COLD
```

注意 ：当我们将目录设置为 COLD 并且我们未配置 ARCHIVE 存储目录的情况下，不可以向该目录直接上传文件，会报出异常。

（2）手动转移

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs mover /hdfsdata
```

（3）检查文件块的分布

```
[Tom@hadoop102 hadoop-3.1.3]$ bin/hdfs fsck /hdfsdata -files -blocks -locations

[DatanodeInfoWithStorage[192.168.10.105:9866,DS-d46d08e1-80c6-4fca-b0a2-4a3dd7ec745
9,ARCHIVE],
DatanodeInfoWithStorage[192.168.10.106:9866,DS-827b3f8b-84d7-47c6-8a14-0166096f919d,
ARCHIVE]]
```

所有文件块都在 ARCHIVE，符合 COLD 存储策略。

5.2.6 ONE_SSD 策略测试

（1）接下来我们将存储策略从默认的 HOT 更改为 One_SSD

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs storagepolicies -setStoragePolicy -path /hdfsdata -policy One_SSD
```

（2）手动转移文件块

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs mover /hdfsdata
```

（3）转移完成后，检查文件块的分布

```
[Tom@hadoop102 hadoop-3.1.3]$ bin/hdfs fsck /hdfsdata -files -blocks -locations

[DatanodeInfoWithStorage[192.168.10.104:9866,DS-0b133854-7f9e-48df-939b-5ca6482c5afb,DISK],
DatanodeInfoWithStorage[192.168.10.103:9866,DS-2481a204-59dd-46c0-9f87-ec4647ad429a,SSD]]
```

文件块分布为一半在 SSD，一半在 DISK，符合 One_SSD 存储策略。

5.2.7 ALL_SSD 策略测试

（1）接下来我们将存储策略从默认的 HOT 更改为 All_SSD

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs storagepolicies -setStoragePolicy -path /hdfsdata -policy All_SSD
```

（2）手动转移文件块

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs mover /hdfsdata
```

（3）转移完成后，检查文件块的分布

```
[Tom@hadoop102 hadoop-3.1.3]$ bin/hdfs fsck /hdfsdata -files -blocks -locations

[DatanodeInfoWithStorage[192.168.10.102:9866,DS-c997cfb4-16dc-4e69-a0c4-9411a1b0c1eb,SSD],
DatanodeInfoWithStorage[192.168.10.103:9866,DS-2481a204-59dd-46c0-9f87-ec4647ad429a,SSD]]
```

所有的文件块都存储在 SSD，符合 All_SSD 存储策略。

5.2.8 LAZY_PERSIST 策略测试

（1））继续改变策略，将存储策略改为 lazy_persist

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs storagepolicies -setStoragePolicy -path /hdfsdata
-policy policy lazy_persist
```

（2）手动转移文件块

```
[Tom@hadoop102 hadoop-3.1.3]$ hdfs mover /hdfsdata
```

（3）转移完成后，检查文件块的分布

```
[Tom@hadoop102 hadoop-3.1.3]$ bin/hdfs fsck /hdfsdata -files -blocks -locations

[DatanodeInfoWithStorage[192.168.10.104:9866,DS-0b133854-7f9e-48df-939b-5ca6482c5af
b,DISK],
DatanodeInfoWithStorage[192.168.10.103:9866,DS-ca1bd3b9-d9a5-4101-9f92-3da5f1baa28b,
DISK]]
```

这里我们发现所有的文件块都是存储在
DISK，按照理论一个副本存储在 RAM_DISK，其他副本存储在 DISK 中，这是因为，我们还需要配置" dfs.datanode.max.locked.memory"，"dfs.block.size"参数。

那么出现存储策略为 LAZY_PERSIST 时，文件块副本都存储在 DISK 上的原因有如下两点：

（1）当客户端所在的 DataNode 节点没有 RAM_DISK 时，则会写入客户端所在的 DataNode 节点的 DISK 磁盘，其余副本会写入其他节点的 DISK 磁盘。

（2）当客户端所在的 DataNode 有 RAM_DISK，但
dfs.datanode.max.locked.memory 参数值未设置或者设置过小（小于“ dfs.block.size”参数值）时，则会写入客户端所在的 DataNode 节点的 DISK 磁盘，其余副本会写入其他节点的 DISK 磁盘。

但是由于虚拟机的“max locked memory”为 64KB，所以，如果参数配置过大，还会报出错误：

```
ERROR org.apache .hadoop.hdfs.server.datanode.DataNode: Exception in
secureMainjava.lang.RuntimeException: Cannot start datanode because the configured max
locked memory size(dfs.datanode.max.locked.memory) of 209715200 bytes is more than the
datanode's available RLIMIT_ MEMLOCK ulimit of 65536 bytes.
```

我们可以通过该命令查询此参数的内存

```
[Tom@hadoop102 hadoop-3.1.3]$ ulimit -a

max locked memory       (kbytes, -l) 64
```

六、HDFS 故障排除

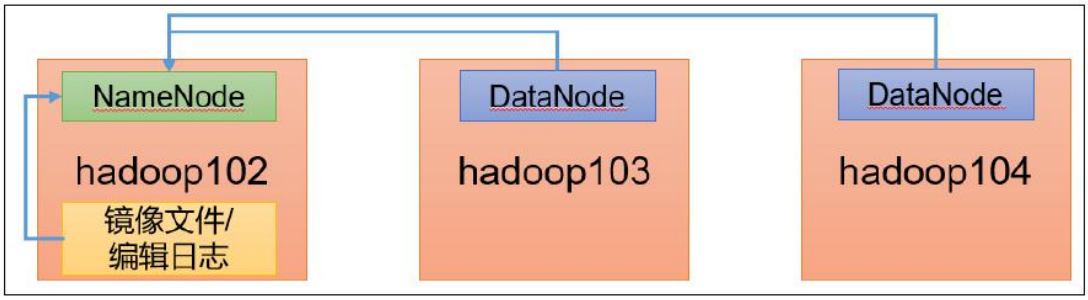
6.1 集群安全模式

安全模式：文件系统只接受读数据请求，而不接受删除、修改等变更请求

进入安全模式场景：

NameNode 在加载镜像文件和编辑日志期间处于安全模式；

NameNode 再接收 DataNode 注册时，处于安全模式。



退出安全模式条件

`dfs.namenode.safemode.min.datanodes`:最小可用 datanode 数量 ，默认 0

`dfs.namenode.safemode.threshold-pct`:副本数达到最小要求的 block 占系统总 block 数的百分比 ，默认 0.999f。（只允许丢一个块）

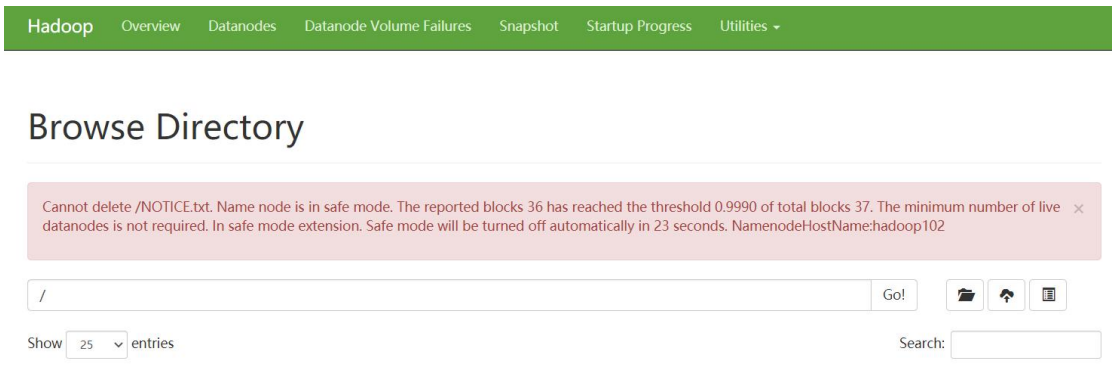
`dfs.namenode.safemode.extension`:稳定时间 ，默认值 30000 毫秒，即 30 秒

基本语法

集群处于安全模式，不能执行重要操作（写操作） 。 集群启动完成后，自动退出安全模式。

```
（1）bin/hdfs dfsadmin -safemode get（功能描述：查看安全模式状态）
（2）bin/hdfs dfsadmin -safemode enter （功能描述：进入安全模式状态）
（3）bin/hdfs dfsadmin -safemode leave（功能描述：离开安全模式状态）
（4）bin/hdfs dfsadmin -safemode wait（功能描述：等待安全模式状态）
```

案例：集群启动后，立即来到集群上删除数据，提示集群处于安全模式



6.2 慢磁盘监控

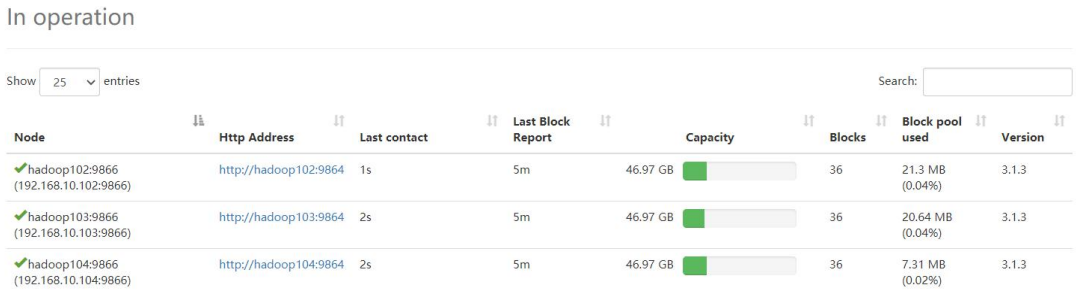
“慢磁盘”指的时写入数据非常慢的一类磁盘。其实慢性磁盘并不少见，当机器运行时间长了，上面跑的任务多了，磁盘的读写性能自然会退化，严重时就会出现写入数据延时的问题。

如何发现慢磁盘？

正常在 HDFS 上创建一个目录，只需要不到 1s 的时间。如果你发现创建目录超过 1 分钟及以上，而且这个现象并不是每次都有。只是偶尔慢了一下，就很有可能存在慢磁盘。可以采用如下方法找出是哪块磁盘慢：

通过心跳未联系时间

一般出现慢磁盘现象，会影响到 DataNode 与 NameNode 之间的心跳。正常情况心跳时间间隔是 3s。超过 3s 说明有异常。



fio 命令，测试磁盘的读写性能

（1）顺序读测试

```
[Tom@hadoop102 ~]#sudo yum install -y fio
```

```
[Tom@hadoop102 ~]# sudo fio -filename=/home/Tom/test.log -direct=1 -iodepth 1 -thread -rw=read -ioengine=psync-bs=16k -size=2G -numjobs=10 -runtime=60 -group_reporting -name=test_r
```

```
Run status group 0 (all jobs):
```

```
READ: bw=360MiB/s (378MB/s), 360MiB/s-360MiB/s (378MB/s-378MB/s), io=20.0GiB (21.5GB), run=56885-56885msec
```

结果显示，磁盘的总体顺序读速度为 360MiB/s

（2）顺序写测试

```
[Tom@hadoop102 ~]# sudofio -filename=/home/Tom/test.log -direct=1 -iodepth 1 -thread -rw=write -ioengine=psync -bs=16k -size=2G -numjobs=10 -runtime=60 -group_reporting -name=test_w
```

```
Run status group 0 (all jobs):
```

```
WRITE: bw=341MiB/s (357MB/s), 341MiB/s-341MiB/s (357MB/s-357MB/s), io=19.0GiB (21.4GB), run=60001-60001msec
```

结果显示，磁盘的总体顺序写速度为 341MiB/s

（3）随机写测试

```
[Tom@hadoop102 ~]#sudofio -filename=/home/Tom/test.log -direct=1 -iodepth 1 -thread
-rw=randwrite -ioengine=psync-bs=16k -size=2G -numjobs=10 -runtime=60 -group_reporting
-name=test_randw
```

```
Run status group 0 (all jobs):
```

```
WRITE: bw=309MiB/s (324MB/s), 309MiB/s-309MiB/s (324MB/s-324MB/s), io=18.1GiB (19.4GB),
run=60001-60001msec
```

结果显示，磁盘的总体随机写速度为 309MiB/s。

（4）顺序读测试

```
[Tom@hadoop102 ~]# sudo fio -filename=/home/Tom/test.log -direct=1 -iodepth 1 -thread
-rw=randrw -rwmixread=70 -ioengine=psync -bs=16k -size=2G -numjobs=10 -runtime=60
-group_reporting -name=test_r_w -ioscheduler=noop
```

```
Run status group 0 (all jobs):
```

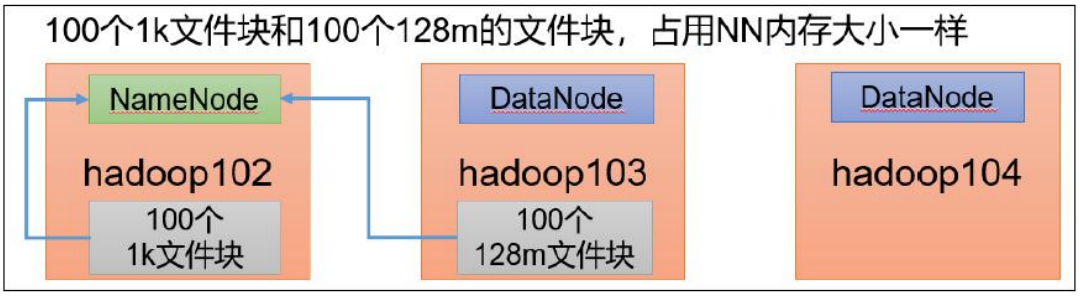
```
READ: bw=220MiB/s(231MB/s), 220MiB/s-220MiB/s (231MB/s-231MB/s), io=12.9GiB (13.9GB),
run=60001-60001msec
```

```
WRITE: bw=94.6MiB/s (99.2MB/s), 94.6MiB/s-94.6MiB/s (99.2MB/s-99.2MB/s), io=5674MiB
(5950MB), run=60001-60001msec
```

结果显示，磁盘的总体混合随机读写 ，读速度为 220MiB/s，写速度 94.6MiB/s。

6.3 小文件归档

HDFS 存储小文件弊端



每个文件均按块存储，每个块的元数据存储在 NameNode 的内存中，因此 HDFS 存储小文件会非常低效。因为大量的小文件会耗尽 NameNode 中的大部分内存。但注意，存储小文件所需要的磁盘容量和数据块的大小无关。例如，一个 1MB 的文件设置为 128MB 的块存储，实际使用的是 1MB 的磁盘空间，而不是 128MB。

解决存储小文件办法之一

HDFS 存档文件或 HAR 文件，是一个更高效的文件存档工具， 它将文件存入 HDFS 块，在减少 NameNode 内存使用的同时，允许对文件进行透明的访问。具

体说来， HDFS 存档文件对内还是一个一个独立文件，对 NameNode 而言却是一个整体，减少了 NameNode 的内存。

案例实操

（1）需要启动 YARN 进程

```
[Tom@hadoop102 hadoop 3 1 3 ]$ start-yarn.sh
```

（2）归档文件

把 /input 目录里面的所有文件归档成一个叫 input.har 的归档文件，并把归档 后文件存储到 /output 路径下。

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop archive -archiveName input.har -p /input /output
```

（3）查看文档

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -ls /output/input.har

Found 4 items

-rw-r--r--   3 Tom supergroup          0 2021-06-26 17:26 /output/input.har/_SUCCESS

-rw-r--r--   3 Tom supergroup        268 2021-06-26 17:26 /output/input.har/_index

-rw-r--r--   3 Tom supergroup         23 2021-06-26 17:26
/output/input.har/_masterindex

-rw-r--r--   3 Tom supergroup         74 2021-06-26 17:26 /output/input.har/part-0


[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -ls har:///output/input.har

2021-06-26 17:33:50,362 INFO sasl.SaslDataTransferClient: SASL encryption trust check:
localHostTrusted = false, remoteHostTrusted = false

Found 3 items

-rw-r--r--   3 Tom supergroup         38 2021-06-26 17:24
har:///output/input.har/shu.txt

-rw-r--r--   3 Tom supergroup         19 2021-06-26 17:24
har:///output/input.har/wei.txt

-rw-r--r--   3 Tom supergroup         17 2021-06-26 17:24
har:///output/input.har/wu.txt
```

（4）解归档文件

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop fs -cp har:///output/input.har/* /
```

七、MapReduce 生产经验

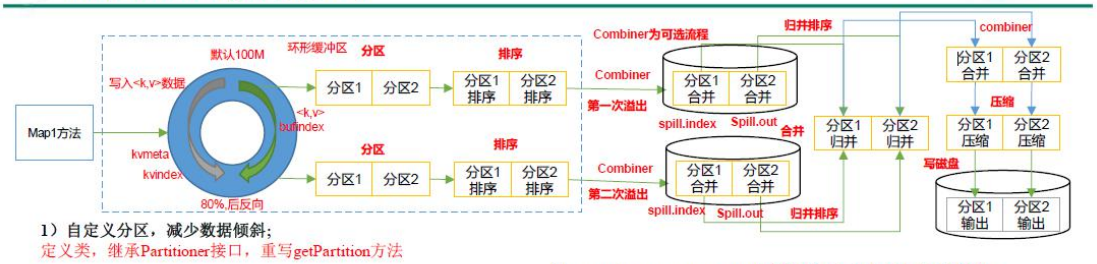
MapReduce 跑的慢的原因

（1）计算机性能：CPU、内存、磁盘、网络

（2）I/O 操作优化：数据倾斜；Map 运行时间太长，导致 Reduce 等待过久；小文件过多

MapReduce 常用调优参数

MapReduce优化（上）



1）自定义分区，减少数据倾斜；
定义类，继承Partitioner接口，重写getPartition方法

2）减少溢写的次数
mapreduce.task.io.sort.mb
Shuffle的环形缓冲区大小，默认100m，可以提高到200m
mapreduce.map.sort.spill.percent
环形缓冲区溢出的阈值，默认80%，可以提高的90%

3）增加每次Merge合并次数
mapreduce.task.io.sort.factor默认10，可以提高到20

4）在不影响业务结果的前提条件下可以提前采用Combiner
`job.setCombinerClass(xxxReducer.class);`

5）为了减少磁盘IO，可以采用Snappy或者LZO压缩
conf.setBoolean("mapreduce.map.output.compress", true);
conf.setClass("mapreduce.map.output.compress.codec", SnappyCodec.class,CompressionCodec.class);

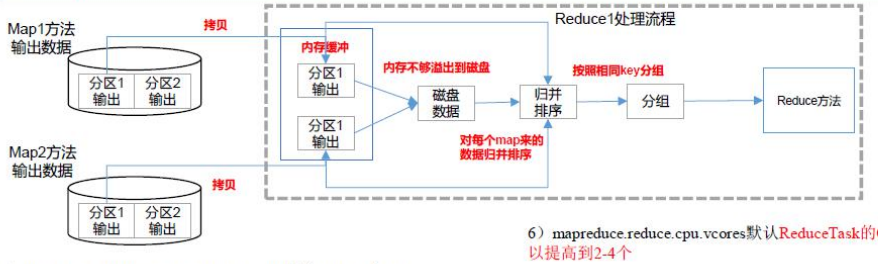
6）mapreduce.map.memory.mb 默认MapTask内存上限1024MB。可以根据128m数据对应1G内存原则提高该内存。

7）mapreduce.map.java.opts：控制MapTask堆内存大小。（如果内存不够，报：java.lang.OutOfMemoryError）

8）mapreduce.map.cpu.vcores 默认MapTask的CPU核数1。计算密集型任务可以增加CPU核数

9）异常重试
mapreduce.map.maxattempts每个Map Task最大重试次数，一旦重试次数超过该值，则认为Map Task运行失败，默认值：4。根据机器性能适当提高。

MapReduce优化（下）



1）mapreduce.reduce.shuffle.parallelcopies每个Reduce去Map中拉取数据的并行数，默认值是5。可以提高到10。

2）mapreduce.reduce.shuffle.input.buffer.percent
Buffer大小占Reduce可用内存的比例，默认值0.7。可以提高到0.8

3）mapreduce.reduce.shuffle.merge.percent Buffer中的数据达到多少比例开始写入磁盘，默认值0.66。可以提高到0.75

4）mapreduce.reduce.memory.mb 默认ReduceTask内存上限1024MB，根据128m数据对应1G内存原则，适当提高内存到4-6G

5）mapreduce.reduce.java.opts：控制ReduceTask堆内存大小。（如果内存不够，报：java.lang.OutOfMemoryError）

6）mapreduce.reduce.cpu.vcores默认ReduceTask的CPU核数1个。可以提高到2-4个

7）mapreduce.reduce.maxattempts每个Reduce Task最大重试次数，一旦重试次数超过该值，则认为Map Task运行失败，默认值：4。

8）mapreduce.job.reduce.slowstart.completedmaps当MapTask完成的比例达到该值后才会为ReduceTask申请资源。默认是0.05。

9）mapreduce.task.timeout如果一个Task在一定时间内没有任何进入，即不会读取新的数据，也没有输出数据，则认为该Task处于Block状态，可能是卡住了，也许永远会卡住，为了防止因为用户程序永远Block住不退出，则强制设置了一个该超时时间（单位毫秒），默认是600000（10分钟）。如果你的程序对每条输入数据的处理时间过长，建议将该参数调大。

10）如果可以不用Reduce，尽可能不用

MapReduce 数据倾斜问题

数据频率倾斜——某一个区域的数据量要远远大于其他区域。
数据大小倾斜——部分记录的大小远远大于平均值。
减少数据倾斜的方法：
（1）首先检查是否空值过多造成的数据倾斜。生产环境，可以直接过滤掉空值；如果想保留空值，就自定义分区，将空值加随机数打散。最后再二次聚合 。
（2)能在 map 阶段提前处理,最好先在 Map 阶段处理。如：Combiner、MapJoin
（3）设置多个 reduce 个数

八、Hadoop 综合调优

8.1 Hadoop 小文件优化方法

8.1.1 Hadoop 小文件弊端

HDFS 上每个文件都要在 NameNode 上创建对应的元数据，这个元数据的大小约为 150byte，这样当小文件比较多的时候，就会产生很多的元数据文件 一方面会大量占用 NameNode 的内存空间 另一方面就是元数据文件过多，使得寻址索引速度变慢。
小文件过多，在进行 MR 计算时，会生成过多切片，需要启动过多的 MapTask。每个 MapTask 处理的数据量小， 导致 MapTask 的处理时间比启动时间还小，白白消耗资源。

8.1.2 Hadoop 小文件解决方案

1）**在数据采集的时候，就将小文件或小批数据合成大文件再上传 HDFS（数据源头）**
2）**Hadoop Archive（存储方向）**
是一个高效的将小文件放入 HDFS 块中的文件存档工具,能够将多个小文件打包成一个 HAR 文件，从而达到减少 NameNode 的内存使用。
3）**CombineTextInputFormat（计算方向）**
CombineTextInputFormat 用于将多个小文件在切片过程中生成一个单独的切片或者少量的切片。
4）**开启 uber 模式，实现 JVM 重用 （计算方向）**
默认情况下，每个 Task 任务都需要启动一个 JVM 来运行，如果 Task 任务计

算的数据量很小，我们可以让同一个 Job 的多个 Task 运行在一个 JVM 中，不必为每个 Task 都开启一个 JVM。

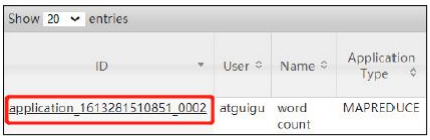
（1）未开启 uber 模式，在 /input 路径上上传多个小文件 并执行 wordcount 程序

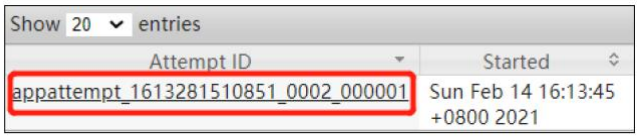
```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount /input /output2
```

（2）观察控制台

```
2021-06-26 16:18:07,607 INFO mapreduce.Job: Job job_1613281510851_0002 running in uber
mode : false
```

（3）观察 http://hadoop103:8088/cluster





（4）开启 uber 模式，在 mapred-site.xml 中添加如下配置

```
<!--开启 uber 模式，默认关闭-->
<property>
 <name>mapreduce.job.ubertask.enable</name>
 <value>true</value>
</property>
<!--uber 模式中最大的 mapTask 数量，可向下修改-->
<property>
 <name>mapreduce.job.ubertask.maxmaps</name>
 <value>9</value>
</property>
<!--uber 模式中最大的 reduce 数量，可向下修改-->
<property>
 <name>mapreduce.job.ubertask.maxreduces</name>
 <value>1</value>
</property>
<!--uber 模式中最大的输入数据量，默认使用 dfs.blocksize 的值，可向下修改-->
<property>
 <name>mapreduce.job.ubertask.maxbytes</name>
 <value></value>
</property>
```

（5）分发配置

```
[Tom@hadoop102 hadoop]$ xsync mapred-site.xml
```

（6）再次执行 wordcount 程序

```
[Tom@hadoop102 hadoop-3.1.3]$ hadoop jar
share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar wordcount /input /output2
```

（7）观察控制台

```
2021-06-27 16:28:36,198 INFO mapreduce.Job: Job job_1613281510851_0003 running in uber
mode : true
```

（8）观察 http://hadoop103:8088/cluster

8.2 测试 MapReduce 计算性能

使用 Sort 程序评测 MapReduce

注：一个虚拟机不超过 150G 磁盘尽量不要执行这段代码

（1）使用 RandomWriter 来产生随机数，每个节点运行 10 个 Map 任务，每个 Map 产生大约 1G 大小的二进制随机数

```
[Tom@hadoop102 mapreduce]$ hadoop jar
/opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar
randomwriter random-data
```

（2）执行 Sort 程序

```
[Tom@hadoop102 mapreduce]$ hadoop jar
/opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar
sortrandom-data sorted-data
```

（3）验证数据是否真正排好序了

```
[Tom@hadoop102 mapreduce]$ hadoop jar
/opt/module/hadoop-3.1.3/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3
1.3-tests.jar testmapredsort -sortInput random-data -sortOutput sorted-data
```

8.3 企业开发场景案例

8.3.1 需求

（1）需求：从 1G 数据中，统计每个单词出现次数。服务器 3 台，每台配置 4G 内存，4 核 CPU，4 线程。

（2）需求分析：

1G/128m=8 个 MapTask；1 个 ReduceTask；1 个 mrAppMaster，平均每个节点运行 10 个/3 台≈3 个任务（4 3 3）

8.3.2 HDFS 参数调优

（1）修改 hadoop-env.sh

```
export HDFS_NAMENODE_OPTS="-Dhadoop.security.logger=INFO,RFAS-Xmx1024m"
export HDFS_DATANODE_OPTS="-Dhadoop.security.logger=ERROR,RFAS-Xmx1024m"
```

（2）修改 hdfs-site.xml

```
<!--NameNode 有一个工作线程池，默认值是 10-->
<property>
 <name>dfs.namenode.handler.count</name>
 <value>21</value>
</property>
```

（3）修改 core-site.xml

```
<!--配置垃圾回收时间为 60 分钟-->
<property>
 <name>fs.trash.interval</name>
 <value>60</value>
</property>
```

（4）分发配置

```
[Tom@hadoop102 hadoop]$ xsync hadoop-env.sh hdfs-site.xml core-site.xml
```

8.3.3 MapReduce 参数调优

（1）修改 mapred-site.xml

```
<!--环形缓冲区大小，默认 100m-->
<property>
 <name>mapreduce.task.io.sort.mb</name>
 <value>100</value>
</property>

<!--环形缓冲区溢写阈值，默认 0.8-->
<property>
 <name>mapreduce.map.sort.spill.percent</name>
 <value>0.80</value>
</property>
```

```
<!--merge 合并次数，默认 10 个-->
<property>
 <name>mapreduce.task.io.sort.factor</name>
 <value>10</value>
</property>

<!--maptask 内存，默认 1g；maptask 堆内存大小默认和该值大小一致mapreduce.map.java.opts-->
<property>
 <name>mapreduce.map.memory.mb</name>
 <value>-1</value>
 <description>The amount of memory to request from the scheduler for each    map ta
sk. If this is not specified or is non-positive, it is inferred frommapreduce.map.j
ava.opts and mapreduce.job.heap.memory-mb.ratio. If java-opts are also not specifie
d, we set it to 1024.
</description>
</property>

<!--matask 的CPU 核数，默认 1 个-->
<property>
 <name>mapreduce.map.cpu.vcores</name>
 <value>1</value>
</property>

<!--matask 异常重试次数，默认 4 次-->
<property>
 <name>mapreduce.map.maxattempts</name>
 <value>4</value>
</property>

<!--每个 Reduce 去 Map 中拉取数据的并行数。默认值是 5-->
<property>
 <name>mapreduce.reduce.shuffle.parallelcopies</name>
 <value>5</value>
</property>

<!--Buffer 大小占 Reduce 可用内存的比例，默认值 0.7-->
<property>
 <name>mapreduce.reduce.shuffle.input.buffer.percent</name>
 <value>0.70</value>
</property>

<!--Buffer 中的数据达到多少比例开始写入磁盘，默认值 0.66。-->
<property>
```

```
 <name>mapreduce.reduce.shuffle.merge.percent</name>
 <value>0.66</value>
</property>

<!--reducetask 内存，默认 1g；reducetask 堆内存大小默认和该值大小一致
mapreduce.reduce.java.opts -->
<property>
 <name>mapreduce.reduce.memory.mb</name>
 <value>-1</value>
 <description>The amount of memory to request from the scheduler for each    reduce
 task. If this is not specified or is non-positive, it is inferred
from mapreduce.reduce.java.opts and mapreduce.job.heap.memory-mb.ratio.
If java-opts are also not specified, we set it to 1024.
</description>
</property>

<!--reducetask 的 CPU 核数，默认 1 个-->
<property>
 <name>mapreduce.reduce.cpu.vcores</name>
 <value>2</value>
</property>

<!--reducetask 失败重试次数，默认 4 次-->
<property>
 <name>mapreduce.reduce.maxattempts</name>
 <value>4</value>
</property>

<!--当 MapTask 完成的比例达到该值后才会为 ReduceTask 申请资源。默认是 0.05-->
<property>
 <name>mapreduce.job.reduce.slowstart.completedmaps</name>
 <value>0.05</value>
</property>

<!--如果程序在规定的默认 10 分钟内没有读到数据，将强制超时退出-->
<property>
 <name>mapreduce.task.timeout</name>
 <value>600000</value>
</property>
```

（2）分发配置

```
[Tom@hadoop102 hadoop]$ xsync mapred-site.xml
```

8.3.4 Yarn 参数调优

（1）修改 yarn-site.xml 配置参数如下：

```
<!--选择调度器，默认容量-->
<property>
 <description>The class to use as the resource scheduler.</description>
 <name>yarn.resourcemanager.scheduler.class</name>
 <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>
</property>

<!--ResourceManager 处理调度器请求的线程数量,默认 50；如果提交的任务数大于 50，可以增加该值，但是不能超过 3 台* 4 线程=12 线程（去除其他应用程序实际不能超过 8）-->
<property>
 <description>Number of threads to handle scheduler interface.</description>
 <name>yarn.resourcemanager.scheduler.client.thread-count</name>
 <value>8</value>
</property>

<!--是否让 yarn 自动检测硬件进行配置，默认是 false，如果该节点有很多其他应用程序，建议手动配置。如果该节点没有其他应用程序，可以采用自动-->
<property>
 <description>Enable auto-detection of node capabilities such as memory and CPU.</description>
 <name>yarn.nodemanager.resource.detect-hardware-capabilities</name>
 <value>false</value>
</property>

<!--是否将虚拟核数当作 CPU 核数，默认是 false，采用物理 CPU 核数-->
<property>
 <description>Flag to determine if logical processors(such as hyperthreads) should be counted as cores. Only applicable on Linux when yarn.nodemanager.resource.cpu-vcores is set to -1 and yarn.nodemanager.resource.detect-hardware-capabilities is true.</description>
 <name>yarn.nodemanager.resource.count-logical-processors-as-cores</name>
 <value>false</value>
</property>

<!--虚拟核数和物理核数乘数，默认是 1.0-->
<property>
 <description>Multiplier to determine how to convert phyiscal cores to vcores. This value is used if yarn.nodemanager.resource.cpu-vcores is set to -1(which implies auto-calculate vcores) and yarn.nodemanager.resource.detect-hardware-capabilities is
```

```
 set to true. Thenumber of vcores will be calculated asnumber of CPUs * multiplier.
</description>
 <name>yarn.nodemanager.resource.pcores-vcores-multiplier</name>
 <value>1.0</value>
</property>

<!--NodeManager 使用内存数，默认 8G，修改为 4G 内存-->
<property>
 <description>Amount of physical memory, in MB, that can be allocated for container
s. If set to -1 and yarn.nodemanager.resource.detect-hardware-capabilities is true,
 it is automatically calculated(in case of Windows and Linux).In other cases, the d
efault is 8192MB.</description>
 <name>yarn.nodemanager.resource.memory-mb</name>
 <value>4096</value>
</property>

<!--nodemanager 的 CPU 核数，不按照硬件环境自动设定时默认是 8 个，修改为 4 个-->
<property>
 <description>Number of vcores that can be allocated
for containers. This is used by the RM scheduler when allocating resources for cont
ainers. This is not used to limit the number of CPUs used by YARN containers. If it
 is set to -1 and yarn.nodemanager.resource.detect-hardware-capabilities is true, i
t is automatically determined from the hardware in case of Windows and Linux.In oth
er cases, number of vcores is 8 by default.</description>
 <name>yarn.nodemanager.resource.cpu-vcores</name>
 <value>4</value>
</property>

<!--容器最小内存，默认 1G -->
<property>
 <description>The minimum allocation for every container request at the RMin MBs. M
emory requests lower than this will be set to the value of thisproperty. Additional
ly, a node manager that is configured to have less memorythan this value will be sh
ut down by the resource manager.</description>
 <name>yarn.scheduler.minimum-allocation-mb</name>
 <value>1024</value>
</property>

<!--容器最大内存，默认 8G，修改为 2G -->
<property>
 <description>The maximum allocation for every container request at the RMin MBs. M
emory requests higher than this will throw anInvalidResourceRequestException.</desc
ription>
 <name>yarn.scheduler.maximum-allocation-mb</name>
```

```
 <value>2048</value>
</property>

<!--容器最小CPU 核数，默认 1 个-->
<property>
 <description>The minimum allocation for every container request at the RMin terms
of virtual CPU cores. Requests lower than this will be set to thevalue of this prop
erty. Additionally, a node manager that is configured tohave fewer virtual cores th
an this value will be shut down by the resourcemanager.</description>
 <name>yarn.scheduler.minimum-allocation-vcores</name>
 <value>1</value>
</property>

<!--容器最大CPU 核数，默认 4 个，修改为2 个-->
<property>
 <description>The maximum allocation for every container request at the RMin terms
of virtual CPU cores. Requests higher than this will throw an InvalidResourceReques
tException.</description>
 <name>yarn.scheduler.maximum-allocation-vcores</name>
 <value>2</value>
</property>

<!--虚拟内存检查，默认打开，修改为关闭-->
<property>
 <description>Whether virtual memory limits will be enforced for containers.</descr
iption>
 <name>yarn.nodemanager.vmem-check-enabled</name>
 <value>false</value>
</property>

<!--虚拟内存和物理内存设置比例,默认 2.1 -->
<property>
<description>Ratio between virtual memory to physical memory whensetting memory lim
its for containers. Container allocations areexpressed in terms of physical memory,
 and virtual memory usageis allowed to exceed this allocation by this ratio.</descr
iption>
 <name>yarn.nodemanager.vmem-pmem-ratio</name>
 <value>2.1</value>
</property>
```

（2）分发配置

```
[Tom@hadoop102 hadoop]$ xsync yarn-site.xml
```

8.3.5 执行程序

（1）重启集群

```
[Tom@hadoop102 hadoop-3.1.3]$ sbin/stop-yarn.sh
```

```
[Tom@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh
```

（2）执行 WordCount 程序

```
[Tom@hadoop102 hadoop 3.1.3]$ hadoop jar
```

```
share/hadoop/ mapreduce/hadoop mapreduce examples 3.1.3.jar
```

```
wordcount /input /output
```

（3）观察 Yarn 任务执行页面 http://hadoop103:8088/cluster/apps