CDH平台重要指标及优化建议

目录

[引言 3](#_Toc25557)

[1 HDFS优化 4](#_Toc30290)

[1.1 HDFS重要参数及设置 4](#_Toc16427)

[1.1.1 Namenode的RPC调用线程数 4](#_Toc25249)

[1.1.2 数据节点的服务器线程数 5](#_Toc12145)

[1.1.3 DataNode数据传输线程数 5](#_Toc20441)

[1.1.4 失效数据节点的读写 5](#_Toc12229)

[1.1.5平衡带宽 6](#_Toc2952)

[1.1.6 系统预留磁盘空间 6](#_Toc2267)

[1.1.7 允许失败的卷数 7](#_Toc5503)

[1.1.8 垃圾自动清除时间 7](#_Toc32358)

[1.1.9 HDFS数据块大小 7](#_Toc301)

[1.1.10 HDFS平衡并行移动块数 7](#_Toc17048)

[1.1.11 HDFS副本数 8](#_Toc23176)

[1.1.12 Namenode的Java堆栈 8](#_Toc4565)

[1.1.13 Datanode的Java堆栈 8](#_Toc29576)

[1.1.14 HDFS访问控制 9](#_Toc26831)

[2 Yarn优化 9](#_Toc26184)

[2.1 Yarn的重要参数 9](#_Toc4256)

[2.1.1 Yarn的resource manager堆空间 9](#_Toc14654)

[2.1.2Yarn的nodemanager的堆空间 10](#_Toc22690)

[2.2 Yarn的容器性能调优 10](#_Toc15918)

[2.2.1 Yarn所有容器可使用的内存 10](#_Toc21521)

[2.2.2 单个容器最小内存 10](#_Toc18885)

[2.2.3 单个容器最大内存 11](#_Toc25022)

[2.2.4 Yarn任务CPU用量 11](#_Toc4353)

[2.2.5 Yarn任务的最小虚拟CPU用量 11](#_Toc13918)

[2.2.6 Yarn任务最大虚拟CPU使用量 11](#_Toc8905)

[3 Hive优化 12](#_Toc9777)

[3.1 Hive重要参数及设置 12](#_Toc26474)

[3.1.1 HiveServer2的Java堆空间配置 12](#_Toc4384)

[3.1.2 Hive metastore server的Java堆空间配置 13](#_Toc7375)

[3.1.3 Beeline客户端的堆空间与垃圾收集配置 15](#_Toc11389)

[3.1.4 Hive Server2的垃圾收集设置 15](#_Toc15795)

[3.1.5 Hive Metastore的高可用 16](#_Toc6)

[3.1.6 HiveServer2高可用的负载均衡 16](#_Toc29465)

[3.1.7 hive查询过滤 16](#_Toc22506)

[4 Impala优化 17](#_Toc27342)

[4.1 Impala硬件优化 17](#_Toc15307)

[4.2 Impala用户与相关目录 18](#_Toc713)

[4.3 Impala重要参数及设置 19](#_Toc15136)

[4.3.1 Catalog Server的java堆 19](#_Toc30873)

[4.3.2 Impala短路径读取 19](#_Toc8549)

[4.3.3 HDFS块位置追踪 20](#_Toc1353)

[4.4 Impala 查询准入控制（精细优化项） 21](#_Toc30964)

[4.5 Impala后续调优案例 22](#_Toc20832)

[5 Kudu优化 23](#_Toc14720)

[5.1 Kudu重要参数及设置 23](#_Toc18921)

[5.1.1 Kudu Tablet Server 最大内存用量限制 23](#_Toc10115)

[5.1.2 Kudu Tablet server的并发线程数 24](#_Toc26275)

[5.1.3 Kudu Tablet Server块缓存 24](#_Toc21546)

[5.1.4 Kudu Tablet复制因子 24](#_Toc13002)

[6 Hbase优化 25](#_Toc27005)

[6.1 Hbase的JVM GC 25](#_Toc13594)

[7 OS相关优化项 27](#_Toc3280)

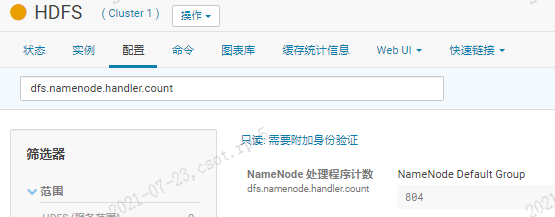
[7.1 交换分区临时解决方案 27](#_Toc30972)

# 1 HDFS优化

## 1.1 HDFS重要参数及设置

### 1.1.1 Namenode的RPC调用线程数

-dfs.namenode.handler.count：指定NameNode 的服务器线程的数量，如果该值设的太小，会经常发现DataNode在连接NameNode的时候总是超时或者连接被拒绝，但该数值过大时，调用延时就会加大。根据相关操作经验，可设置为集群节点数的10倍。



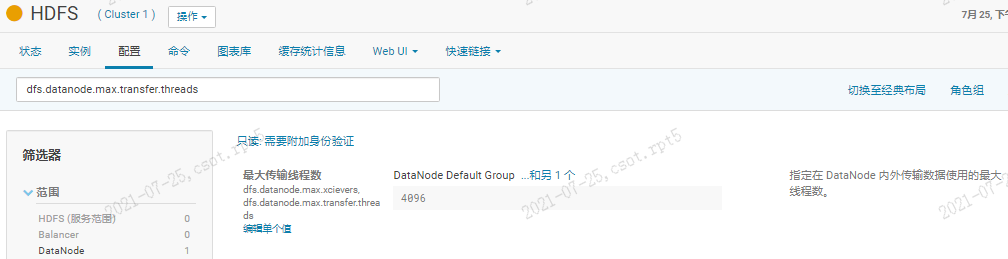
### 1.1.2 数据节点的服务器线程数

-dfs.datanode.handler.count： CDH中默认为3，可适当增加这个数值来提升DataNode RPC服务的并发度。在DataNode上设定,取决于系统的繁忙程度,设置太小会导致性能下降甚至报错。线程数的提高将增加DataNode的内存需求，通常5~10为宜。



### 1.1.3 DataNode数据传输线程数

-dfs.datanode.max.transfer.threads： DataNode可以同时处理的数据传输连接数,即指定在DataNode内外传输数据使用的最大线程数，默认值为4096。



\*推荐将该值调整为8192，充分利用数据节点性能

### 1.1.4 失效数据节点的读写

-dfs.namenode.avoid.read.stale.datanode： 指示是否避免读取“过时”的数据节点（DataNode），这些数据节点（DataNode）的心跳消息在指定的时间间隔内未被名称节点（NameNode）接收。过时的数据节点（DataNode）将移动到返回供读取的节点列表的末尾。默认值是flase，推荐设置为true。

-dfs.namenode.avoid.write.stale.datanode：指示超过失效DataNode时间间隔 NameNode 未收到检测信号信息时是否避免写入失效DataNode。写入应避免使用失效 DataNode。默认值是flase，推荐设置为true。



\*推荐修改以上两参数为“true”，开启功能。

### 1.1.5平衡带宽

-dfs.datanode.balance.bandwidthPerSec：HDFS平衡器检测集群中使用过度或者使用不足的DataNode，并在这些DataNode之间移动数据块来保证负载均衡。如果不对平衡操作进行带宽限制，那么它会很快就会抢占所有的网络资源，不会为Mapreduce作业或者数据输入预留资源。该参数定义了每个DataNode平衡操作所允许的最大使用带宽，这个值的单位是byte。默认是10M，当前服务器为万兆网卡，也就是说我们最大可以设置1.25G带宽。考虑其他服务的传输效率，不被DataNode 平衡数据抢占，推荐设置为64M～512M。



\*可适当提升该值，满足HDFS平衡需求，建议修改设置为64M。

### 1.1.6 系统预留磁盘空间

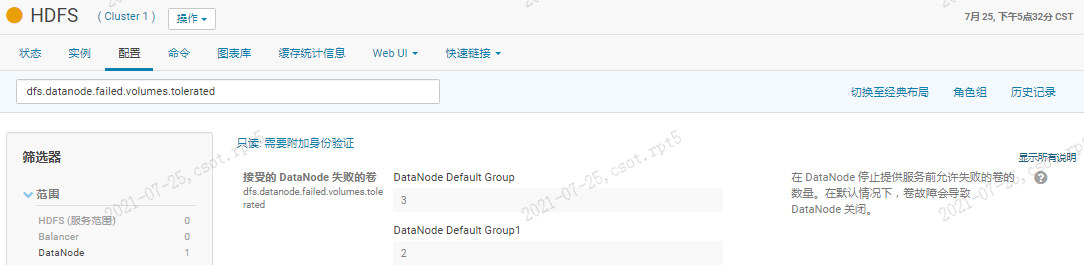
-dfs.datanode.du.reserved： 定义了每个dfs.data.dir所定义的硬盘空间需要保留的大小，以byte为单位。



\*HDFS用量宽裕，且同时运行有Kudu服务，建议保持当前值。

### 1.1.7 允许失败的卷数

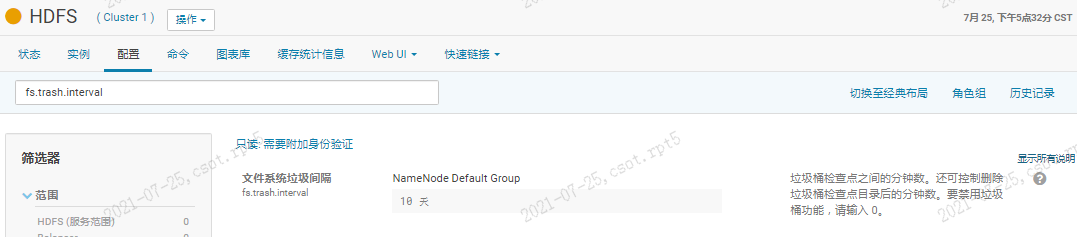
-dfs.datanode.failed.volumes.tolerated： 指定数据节点（DataNode）停止提供服务之前允许失败的卷数。一个DataNode的丢失会导致一些数据块备份数下降，因此，NameNode会命令其他DataNode复制这些丢失的数据块来增加副本数。该参数定义整个DataNode声明失败前允许多少个硬盘出现故障。



\*正常，无需修改

### 1.1.8 垃圾自动清除时间

-fs.trash.interval：垃圾桶检查点之间的分钟数。可控制删除垃圾桶检查点目录后的分钟数。删除HDFS文件时，被移到用户的HDFS主目录中一个名为.Trash目录中，来保留被删除的文件，而不是立即彻底删除。该参数定义.Trash目录下文件被永久删除前保留的时间。在文件被从HDFS永久删除前，用户可以自由地把文件从该目录下移出来并立即还原。



\*正常，该参数无需修改

### 1.1.9 HDFS数据块大小

-dfs.block.size, dfs.blocksize： 该值设置HDFS副本切片大小



\*正常，当前无需修改

### 1.1.10 HDFS平衡并行移动块数

-maximum concurrent moves：这个参数的数值要考虑现有集群的带宽，还有当前设置的副本大小，过大容易引起网络阻塞，过小平衡耗时长。



\*当前值正常，无需修改

### 1.1.11 HDFS副本数

-dfs.replicatio：集群当前值为默认值，为3副本。



\*当前值正常，且HDFS剩余容量充裕，无需修改。

### 1.1.12 Namenode的Java堆栈

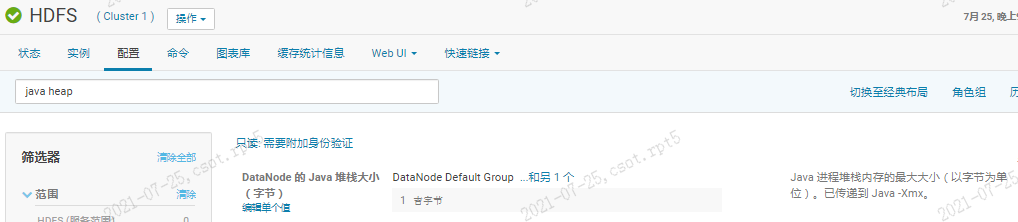
-该值根据Namenode节点的硬件配置及HDFS的工作负荷确定



\*该值设置符合当前配置，无需修改

### 1.1.13 Datanode的Java堆栈

-该值根据Datanode节点的硬件配置及HDFS的工作负荷确定



\*当前为1G，符合当前集群环境，如后续Datanode工作负荷增加可调整为2G。

### 1.1.14 HDFS访问控制

-dfs.namenode.acls.enabled：设置此参数，可避免多个使用者误删其他人数据，同时又具备一定隔离作用。



\*后续如有更高的安全需求，可考虑配置此参数

# 2 Yarn优化

## 2.1 Yarn的重要参数

### 2.1.1 Yarn的resource manager堆空间

-该值决定Yarn任务分配角色resource manager的可使用Java堆内存大小，默认值为1G。生产中根据resource manager压力，通常设置为2~4G。



\*当前值已修改过，符合集群当前环境，如遇resource manager角色GC时间长，导致Yarn任务排队，可调整为4G。

### 2.1.2Yarn的nodemanager的堆空间

-该值决定Yarn任务执行角色node manager的可使用Java堆内存大小，默认值为1G。生产中根据Yarn工作压力，通常设置为1~2G。



## 2.2 Yarn的容器性能调优

### 2.2.1 Yarn所有容器可使用的内存

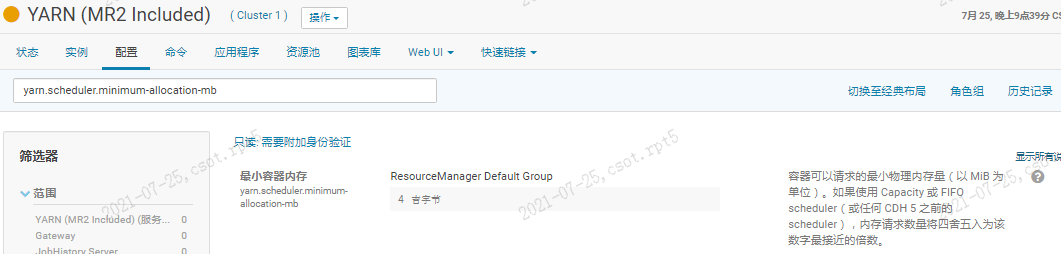
-yarn.nodemanager.resource.memory-mb：M/R任务容器的总可用内存，根据M/R任务量设置该值。



\*集群当前任务不以M/R为主，可根据业务量适当调整，降低该值。

### 2.2.2 单个容器最小内存

-yarn.scheduler.minimum-allocation-mb: M/R任务单个容器最小初始内存，根据单个任务复杂度设置该值。



\*可调整该值至2G，以并行更多小M/R任务

### 2.2.3 单个容器最大内存

-yarn.scheduler.maximum-allocation-mb：M/R任务单个容器最大内存，超出此值的任务会执行失败。



\*当前值正常，无需调整

### 2.2.4 Yarn任务CPU用量

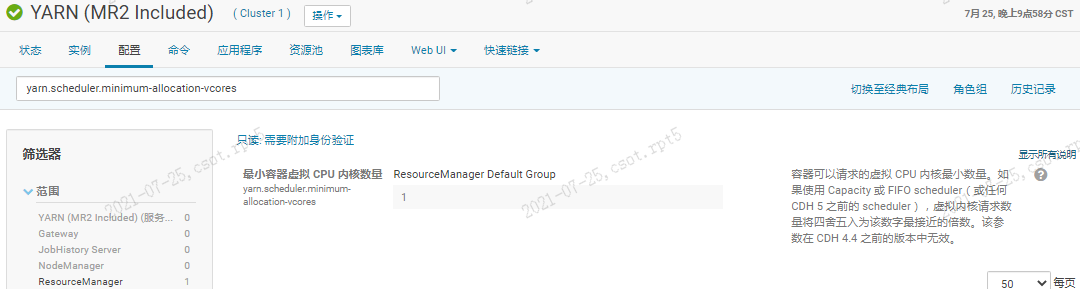
-yarn.nodemanager.resource.cpu-vcores：Yarn的任务节点服务器上可以使用的虚拟CPU个数，默认是8，推荐将值配置与物理核心个数相同，如果节点CPU核心不足8个，要调小这个值。



\*当前集群CPU资源充裕，该值可以适当增加。

### 2.2.5 Yarn任务的最小虚拟CPU用量

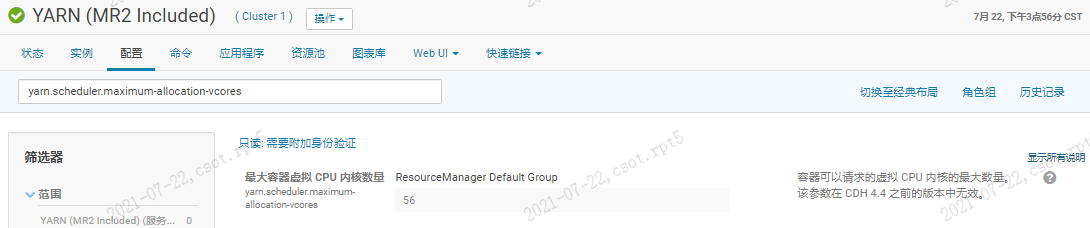
-yarn.scheduler.minimum-allocation-vcores：单个任务最小可申请的虚拟核心数。



\*该值正常，无需调整

### 2.2.6 Yarn任务最大虚拟CPU使用量

-yarn.scheduler.maximum-allocation-vcores：一般就设置成4个，根据cloudera公司性能测试，如果cpu大于等于5之后，cpu利用率反而不是很好 。



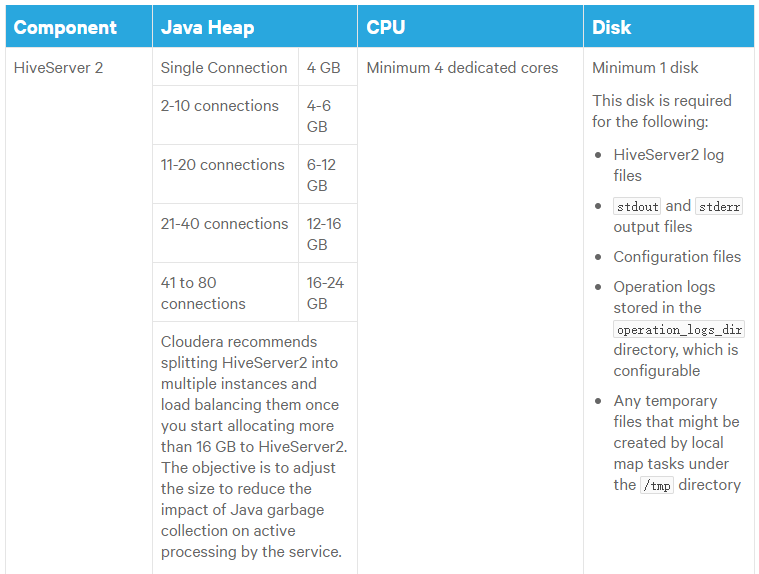
\*该值建议更改为官方推荐值“4”。

# 3 Hive优化

## 3.1 Hive重要参数及设置

### 3.1.1 HiveServer2的Java堆空间配置

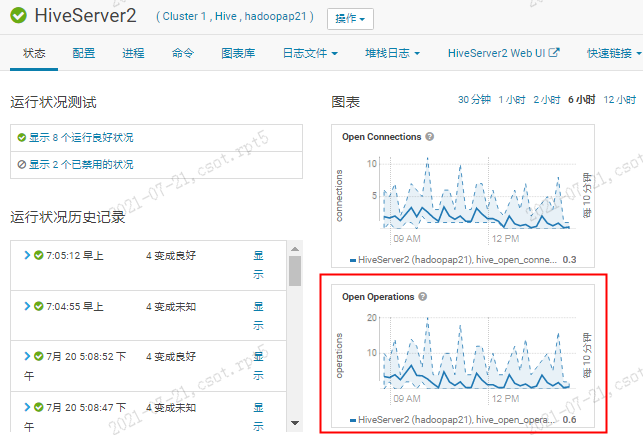
官方文档有如下推荐配置：



\*这些数值仅作为一般指导，可能受列数、分区、复杂连接和客户端活动等因素的影响。根据您的预期进行初期部署，通过测试进行改进，以达到适合您环境的最佳值。

连接数的查看方法：

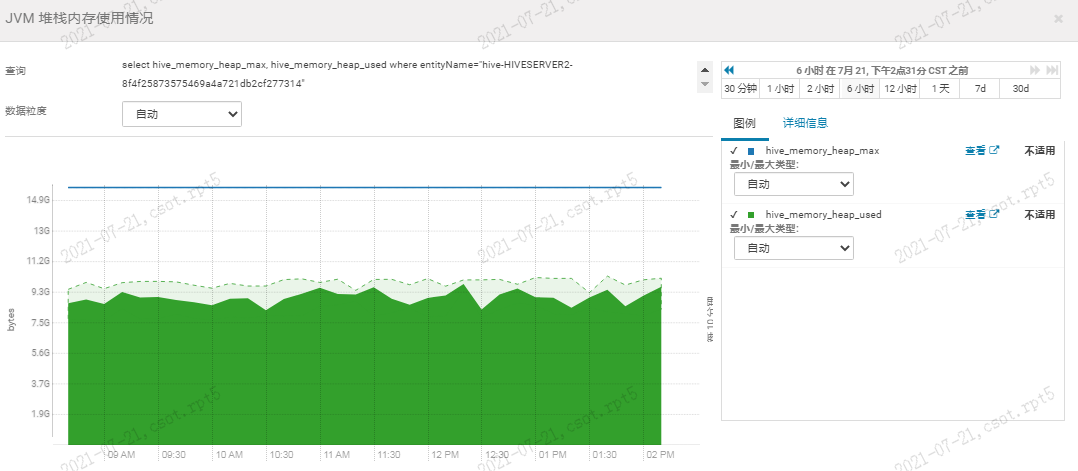
进入HiveServer2角色的状态，即可可视化的查看Hive打开链接数，如下图所示：



根据观测到的峰值连接数，对比官方推荐值进行堆大小的设置，当前设置为：

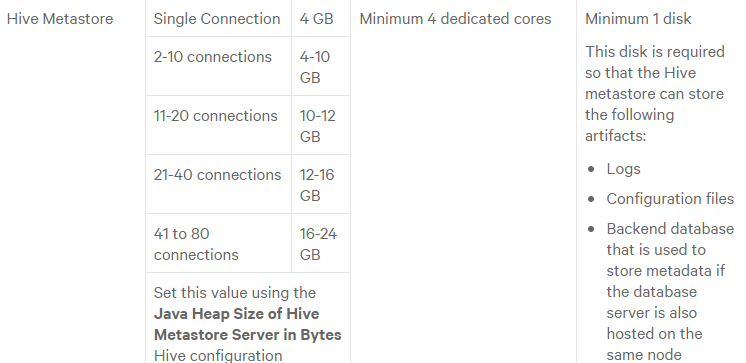


可以观察到HiveServer2当前配置的堆内存使用情况，比较充裕

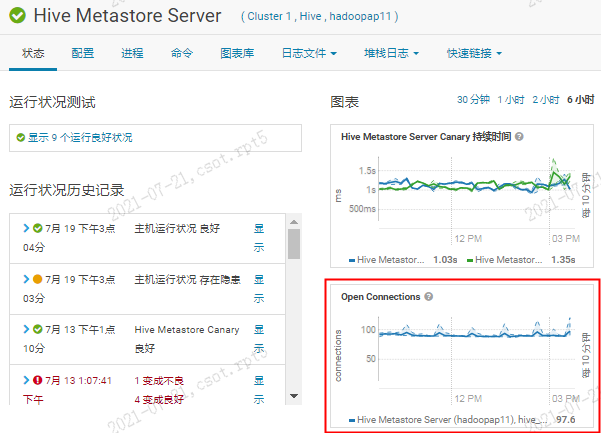


### 3.1.2 Hive metastore server的Java堆空间配置

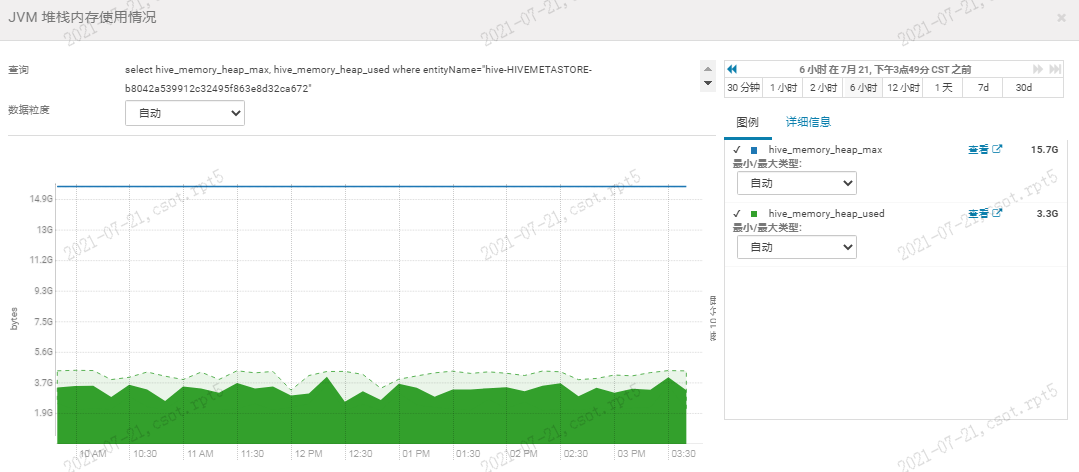
官方文档推荐配置：



当前活跃Hive metastore server的连接数



当前活跃的Hive metastore server的JVM堆使用情况



当前的Hive metastore server的JVM堆大小配置情况及GC配置



\*当前连接数较高，但JVM堆空间使用量不高，推测存在大量空闲连接，并持续观察。关于JVM的GC配置Hbase章节的JVM GC优化，此处不再赘述。

### 3.1.3 Beeline客户端的堆空间与垃圾收集配置

根据Cloudera官方推荐BeeLine CLI堆空间至少分配2G，在CM界面如下位置设置：



\*该值配置满足集群需求，BeeLine服务未有工作状态不良及故障。

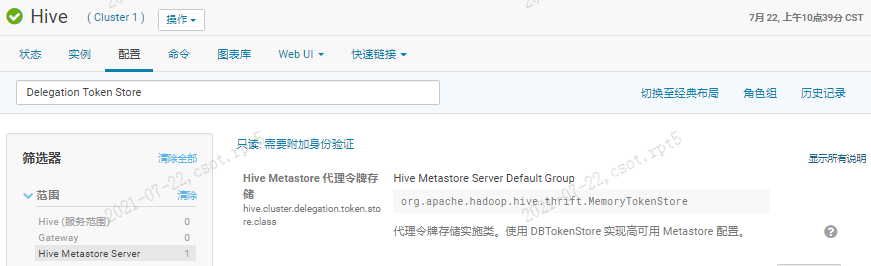
### 3.1.4 Hive Server2的垃圾收集设置



\*当前与HBase采用同样的CMS垃圾回收，可考虑更改为G1收集器。当前最大堆外内存（MaxPermSize）配置为2G，如果Hive Server2堆内存使用未达到上限的情况下，发生内存溢出错误（out of memory），则必须增加最大堆外内存的分配以防止内存溢出错误。

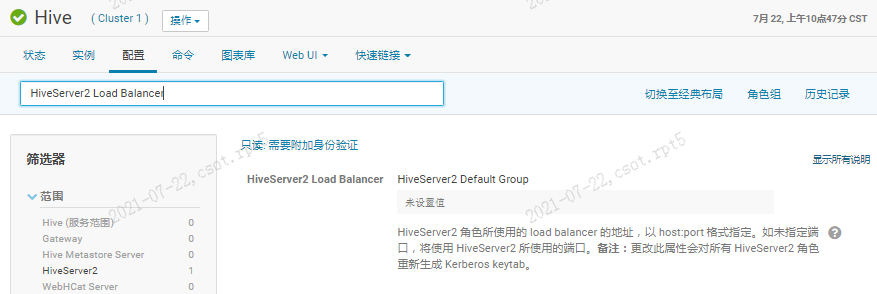
### 3.1.5 Hive Metastore的高可用

Hive Metastore角色开启高可用的前置设置：



### 3.1.6 HiveServer2高可用的负载均衡

HiveServer2角色的负载均衡端口设置，当前默认为HiveServer2端口：



### 3.1.7 hive查询过滤

当前集群环境下为了避免测试误提交超大Hive SQL查询抢夺集群资源，影响到主业务ETL，建议配置如下参数，限制Hive查询的分区量，在一定程度上过滤非正常查询。



加入hive.metastore.limit.partition.request参数，配置值为1000，以限制1000分区以上的hive查询.

执行超过限制的查询会提示以下错误：

MetaException: Number of partitions scanned (=%d) on table '%s' exceeds limit (=%d)

# 4 Impala优化

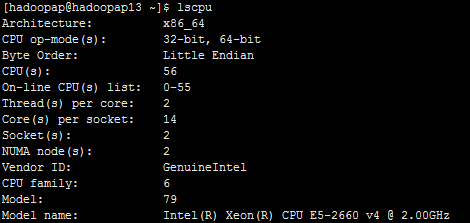
## 4.1 Impala硬件优化

1. CPU

Impala2.2以及更高版本需要支持SSE4.1指令集的CPU，当前Impala版本信息如下：



CPU信息如下：



\*当前CPU与Impala版本适配

1. 内存

256G或者更多，如果在特定节点上查询处理期间的中间结果超过了该节点上Impala可用的内存量，则查询会将临时工作数据写入磁盘，这会导致查询时间过长。下图为Impala查询的内存限制：

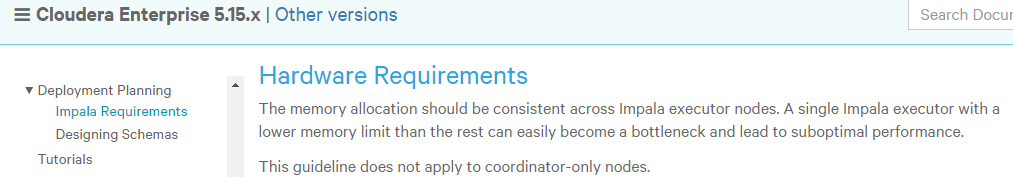


该项有待优化

\*ImpalaD节点之间的内存配置应该是一致的，内存分配低于其他节点的单个ImpalaD很容易成为瓶颈并导致性能欠佳。

\*超过该内存设置限制的查询将被中止，该值受限于机器的物理内存，且不考虑cgroups。

官方说明如下图：



1. 磁盘

DataNode集群应该具有12个或更多磁盘，磁盘I/O速度通常是Impala磁盘性能的限制因素。同时也需要确保有足够的磁盘空间来存储Impala将查询的数据。



\*当前配置符合需求，且根据CM监控得知磁盘I/O与HDFS读写未达瓶颈

## 4.2 Impala用户与相关目录

1. Impala用户

Impala服务会自动创建并使用名为Impala的用户和组，不要删除此帐户或组，也不要修改该帐户或组的权限和权限。确保现有系统不妨碍这些账户和群组的运作。例如，如果您有删除不在白名单中的用户帐户的脚本，请将这些帐户添加到允许的帐户列表中。



\*当前Impala用户正常

为了“DROP TABLE”操作正确删除文件，Impala必须能够将文件移动到HDFS垃圾桶。为此需要在HDFS创建一个/user/impala的目录，Impala用户必须拥有此目录的所有权限，以便可以创建垃圾箱，否则数据文件可能会在“DROP TABLE”陈述。



\*当前HDFS的Impala目录正常

Impala查询不应以root身份执行，使用直接读取可实现最佳Impala性能，但不允许root 使用直接读取。因此，以root身份运行Impala会对性能产生负面影响。

默认情况下，任何用户都可以连接到Impala并访问所有关联的数据库和表。您可以根据连接到Impala服务器的Linux操作系统用户以及该用户的关联组启用授权和身份验证。

## 4.3 Impala重要参数及设置

### 4.3.1 Catalog Server的java堆

官方建议使用4 GB或更多，理想情况下为8 GB或更多，以容纳与Impala使用的表、分区和数据文件。



\*当前该项符合需求，如Impala负载增加导致Impala Catalog角色发生内存交换类状态不良，可考虑适当增加该值。

### 4.3.2 Impala短路径读取

启用短路径读取允许Impala直接从文件系统读取本地数据。这跳过了与DataNode进行通信的需要，从而提高了性能。此设置还最大限度地减少了额外数据副本的数量，在HDFS中配置以下三项参数开启短路径读取。



\*当前集群已启用，正常



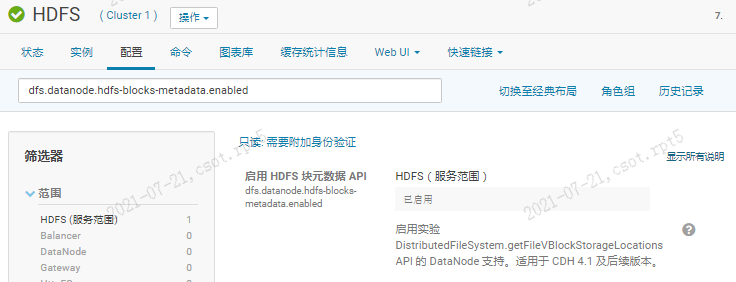
\*当前集群已启用，正常



\*10秒符合官方推荐值10000毫秒，正常

### 4.3.3 HDFS块位置追踪

启用块位置元数据可以让Impala知道哪些磁盘数据块位于哪个磁盘上，从而更好地利用底层磁盘。除非启用此设置，否则Impala不会自动启用该设置。



\*当前集群已开启，正常

以上配置修改后需重启所有DataNode才可生效

## 4.4 Impala 查询准入控制（精细优化项）

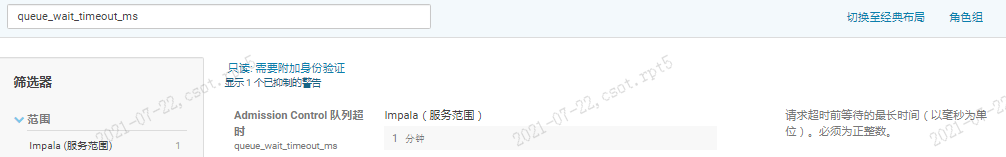
在高压力的CDH集群上，可能大数量并发运行的Impala查询的效果并不是最佳的。例如，当I/O密集型查询充分利用I/O容量时，您可能不会发现运行更多并发查询有任何吞吐量优势。通过允许一些查询全速运行而其他查询等待，而不是让所有查询争用资源并缓慢运行，准入控制可以提高集群的整体吞吐量。

再举一个例子，考虑一个内存绑定的工作负载，例如许多大型连接或聚合查询。每个这样的查询可能会短暂地使用许多GB级的内存来处理中间结果。此时一次运行多个大规模查询可能需要重新运行一些被取消的查询，因为Impala会默认取消超过指定内存限制的查询。在这种情况下，通过准入控制，使集群的整体内存尽可能多的容纳并发查询来提高整体工作负载的可靠性和稳定性。

准入控制功能允许您设置并发Impala查询数量和这些查询使用的内存的上限。任何其他查询都会排队等待，直到较早的查询完成，而不是被取消或运行缓慢并导致争用。当其他查询完成时，允许排队的查询继续进行。

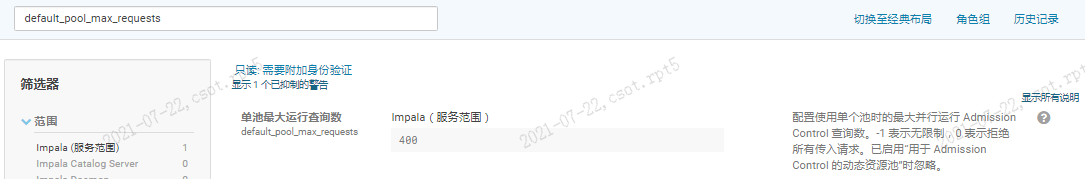
1. queue\_wait\_timeout\_ms :

查询任务超时等待的时长。



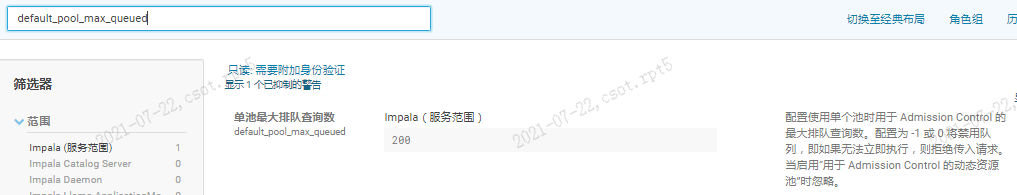
1. default\_pool\_max\_requests ：

没有任务排队时允许运行的最大并发未完成请求数。在高负载期间，并发查询的总数可能会略高，负值表示没有限制。



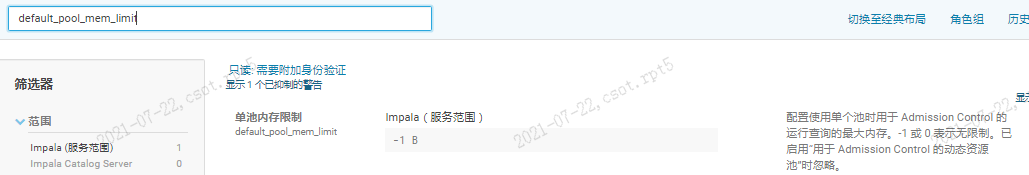
1. default\_pool\_max\_queued ：

任务排队数超过此值时拒绝请求。在高负载期间，并发查询的总数可能会略高，负值表示没有限制。



1. default\_pool\_mem\_limit

没有任务排队时，单池中的未完成请求准许使用的最大内存量（跨整个集群）。以字节、兆字节或千兆字节为单位，由数字后跟后缀指定B，M或者G,不区分大小写。



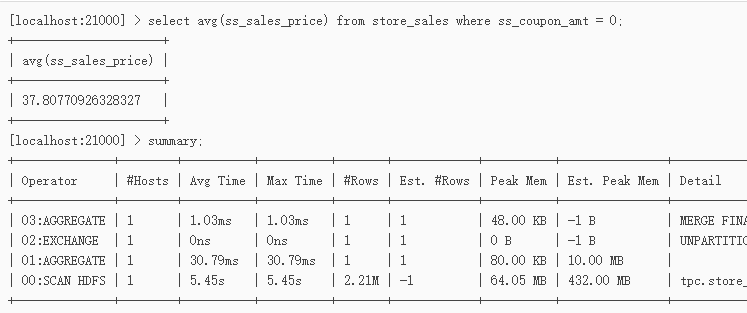
\*减小任务并发或增加单池容量可以使资源消耗量大的任务更快的完成，但不合理的减少任务并发或增加单池最大容量，也可能会发生任务排队、阻塞超时现象。反之不恰当的增加任务并发数量，也可能导致整体吞吐量下降。以上参数修改需要谨慎测试后，再部署实施。

## 4.5 Impala后续调优案例

Impala的EXPLAIN Plans性能查询工具使用

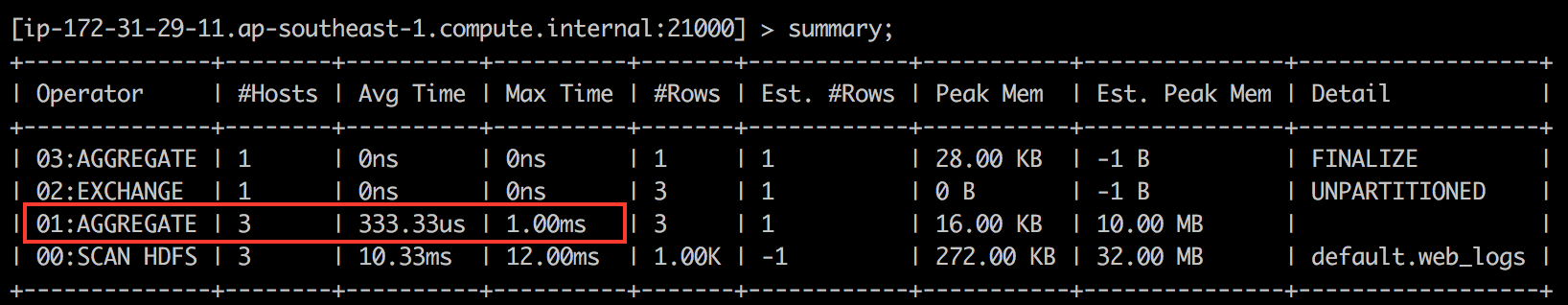
在impala-shell解释器中的SUMMARY命令提供了一个易于理解的查询执行不同阶段的时间概览，用于观测潜在的性能瓶颈。

以下为使用示例：



1. 数据倾斜排查案例

在IMPALA中执⾏完⼀条SELECT语句后，随即输⼊summary指令。此时IMPALA会输出刚刚执⾏完的那条语句的有关信息，如执⾏时间、内存使⽤量等。⽐较平均执⾏时间（Avg Time）和最⼤执⾏时间（Max time），如果两者之间存在的差异过⼤，则可以判断存在数据倾斜问题。



Impala日志轮换

Impala会定期轮换日志，它会自动更新当前日志文件的物理文件并删除不再需要的旧日志文件。



\*提交更改后需要重新启动Impala服务生效。

数值说明：

1）为 0 会保留所有日志文件，在这种情况下，需要使用Linux工具进行手动日志轮换。

2）为 1 仅保留最新的日志文件。

3）默认值为 10。

不建议设置为0，这样可能会使ImpalaD在尝试删除旧日志文件的线程中陷入死循环。

# 5 Kudu优化

## 5.1 Kudu重要参数及设置

### 5.1.1 Kudu Tablet Server 最大内存用量限制

-memory\_limit\_hard\_bytes：tablet Server在批量写入数据时并非实时写入磁盘，而是先Cache在内存中，再flush到磁盘。若该值设置过小时，会造成Kudu数据写入性能显著下降，超过该值是会出现拒绝写入，并诱发写入超时。



### 5.1.2 Kudu Tablet server的并发线程数

- maintenance\_manager\_num\_threads ：这个参数决定了Kudu后台对数据进行维护操作，如写入数据时的并发线程数。并发数越大，吞吐量越高，但对集群计算能力的要求也越高。默认值为1，表示Kudu会采用单线程操作。对于需要大量数据进行快速写入/删除的集群，可以设置更大的值，但不能超过节点挂载的物理磁盘数。该值一般来说，建议设置为4以获取比较均衡的性能，最大不超过8。



### 5.1.3 Kudu Tablet Server块缓存

- block\_cache\_capacity\_mb：分配给Kudu Tablet Server块缓存的最大内存量，适当增加可提高Kudu读性能，一般建议2GB～4GB。



### 5.1.4 Kudu Tablet复制因子

-default\_num\_replicas：默认值为3，表示每个表的数据会在Kudu中存储3份副本。可以根据需要修改这个全局默认值，来设置每个表的副本数。



\*如果Kudu表数量过多，且读取无瓶颈，工作稳定，可考虑减少该值。

# 6 Hbase优化

## 6.1 Hbase的JVM GC

关于Hbase的JVM GC调优

目前使用较多的垃圾收集器简介如下：

ParNew GC：

新生代GC，是SerialGC的多线程版本，最常见的应用场景是配合老年代的CMS GC工作。

CMS（Concurrent Mark Sweep）GC :

优点：基于标记-清除（Mark-Sweep）算法，尽量减少停顿时间。

缺点：存在碎片化问题，在长时间运行的情况下会发生full GC，导致恶劣停顿。会占用更多的CPU资源，和用户争抢线程。在JDK 9中被标记为废弃。

Parrallel GC：

在JDK8等版本中，是server模式JVM的默认GC选择，也被称为吞吐量优先的GC，算法和Serial GC相似，特点是老生代和新生代GC并行进行，更加高效。

G1 GC：

兼顾了吞吐量和停顿时间的GC实现，是Oracle JDK 9后默认的GC可以直观的设值停顿时间，相对于CMS GC，G1未必能做到CMS最好情况下的延时停顿，但比最差情况要好得多

G1仍存在年代的概念，使用了Region棋盘算法，实际上是标记-整理（Mark-Compact）算法，可以避免内存碎片，尤其是堆非常大的时候，G1优势更明显。

G1 吞吐量和停顿表现都非常不错。

GC详细参数释义及建议：

目前集群使用的垃圾回收机制为：

-XX:+UseParNewGC （使用新生代垃圾收集器）

-XX:+UseConcMarkSweepGC （使用CMS垃圾收集器）

-XX:CMSInitiatingOccupancyFraction=70 （当老年代达到70%开始垃圾回收）

-XX:+CMSParallelRemarkEnabled （并行运行最终标记）

\*如上文所言，CMS垃圾收集器具有碎片化，长时间GC时会发生full GC，资源争抢等缺点，此方案已经不是CDH集群JVM垃圾回收方案的最优解。

根据Cloudera官方推荐与英特尔Java性能架构师Eric Kaczmarek测试报告，推荐使用性能更优秀的先进GC机制G1+Parrallel垃圾回收机制（JDK7u21以上均支持）：

G1在HBase中的推荐参数配置（具体值请根据实际调整）：

-XX:+UseG1GC （使用G1垃圾收集器）

-Xms100g -Xmx100g （配置的的堆大小）

-XX:MaxGCPauseMillis=100 （预先配置的可接受GC暂停时间）

--XX:+ParallelRefProcEnabled （多线程发现排队的finalizable对象）

-XX:-ResizePLAB （多CPU下启动动态修改）

-XX:ParallelGCThreads= 8+(40-8)(5/8)=28 （并行GC线程所使用的CPU核心数）

-XX:G1NewSizePercent=1 （用于垃圾回收的堆百分比值）

\*32GB堆：-XX:G1NewSizePercent=3

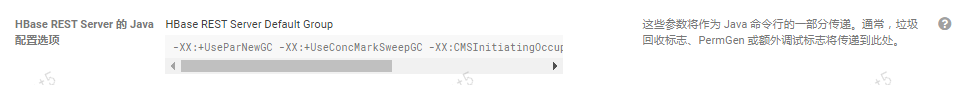
64GB堆：-XX:G1NewSizePercent=2

100GB及以上的堆：-XX:G1NewSizePercent=1

\*ParallelGCThreads值的随算公式：8+（[节点的物理核心数]-8）（5/8）=值

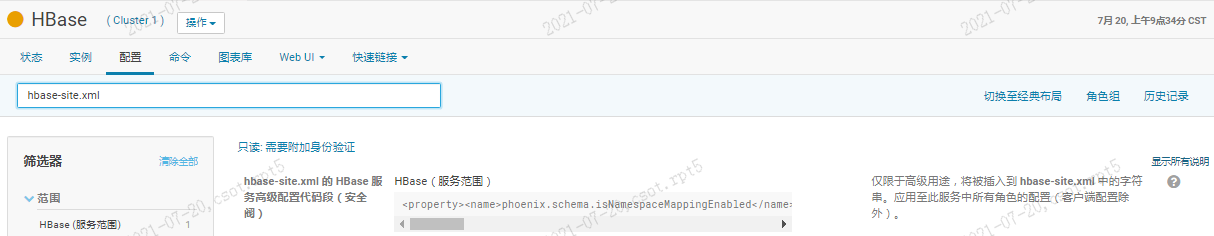
启用G1垃圾收集需要进行的配置更改

1）启用G1 GC收集机制



2）禁用BoundedByteBufferPool

HBase 使用一个BoundedByteBufferPool以避免使堆碎片化，而G1 垃圾收集器具备避免堆碎片化的效用。如果启用G1垃圾收集器，则可以在CDH 5.7及更高版本的HBase中禁用BoundedByteBufferPool。这可以减少JVM需要收集的“老年代”项目的数量，实装至生产集群前建议进行测试。



在上图中的高级代码段中添加以下XML配置：

<property>

<name>hbase.ipc.server.reservoir.enabled</name>

<value>false</value>

</property>

保存更改后，重启Hbase服务生效。

参考文档：

<https://docs.cloudera.com/documentation/enterprise/5-15-x/topics/admin_hbase_garbage_collection.html>

<https://blog.cloudera.com/tuning-java-garbage-collection-for-hbase/>

# 7 OS相关优化项

## 7.1 交换分区临时解决方案

当前集群在业务高峰期经常有多节点报内存交换，这是当前集群遇到内存资源瓶颈的现象之一，交换可能会导致性能下降或稳定性问题。根据CM监控可知有大量内存资源被高速缓存消耗，而不是角色占用。

一般都是黄色告警 swapping suppress

如果集群节点基本不会重启，建议在所有节点使用：

#waspoff -a

临时关闭交换空间。交由OS系统自动释放缓存，避免角色使用内存被交换至磁盘产生意外情况，提升集群组件的稳定性。