目 录

12 性能调优

12.1 配置原则

如何发挥集群最佳性能

原则1：CPU核数分配原则

- 数据节点：建议预留2～4个核给OS和其他进程(数据库，HBase等)外，其他的核分配给YARN。
- 控制节点：由于运行的进程较多，建议预留6～8个核。

原则2：内存分配

除了分配给OS、其他服务的内存外，剩余的资源应尽量分配给YARN。

原则3：虚拟CPU个数分配

节点上YARN可使用的虚拟CPU个数建议配置为逻辑核数的1.5～2倍之间。如果上层计算应用对CPU的计算能力要求不高，可以配置为2倍的逻辑CPU。

原则4：提高磁盘IO吞吐率

尽可能挂载较多的盘，以提高磁盘IO吞吐率。

影响性能的因素

因素1：文件服务器磁盘I/O

一般磁盘顺序读写的速度为百兆级别，如第二代SATA盘顺序读的理论速度为300Mbps，只从一个盘里读，若想达到1Gbps每秒的导入速度是不可能的。并且若从一个磁盘读，单纯依靠增加map数来提高导入速率也不一定可以。因为随着map数变多，对于一个磁盘里的文件读，相当由顺序读变成了随机读，map数越多，磁盘读取文件的随机性越强，读取性能反而越差。如随机读最差可变成800Kbps。 因此需要想办法增大文件服务器的磁盘IO读效率，可以使用专业的文件服务器，如NAS系统，或者使用更简单的方法，把多个磁盘进行Raid0或者Raid5。

因素2：文件服务器网络带宽

单个文件服务器的网络带宽越大越好，建议在10000Mb/s以上。

因素3：集群节点硬件配置

集群节点硬件配置越高，如CPU核数和内存都很多，可以增大同时运行的map或reduce个数，如果单个节点硬件配置难以提升，可以增加集群节点数。

因素4：SFTP参数配置

不使用压缩、加密算法优先选择aes128-cbc，完整性校验算法优先选择umac-64@openssh.com

因素5：集群参数配置

因素6：Linux文件预读值

设置磁盘文件预读值大小为16384，使用linux命令：

echo 16384 > /sys/block/sda/queue/read_ahead_kb

说明：

sda表示当前磁盘的磁盘名。

12.2 Manager

12.2.1 提升Manager配置服务参数的效率

操作场景

在安装集群或者扩容节点以后，集群中可能添加了较多数量的节点。此时如果系统管理员在FusionInsight Manager上修改服务参数、保存新配置并重启服务时，Manager的Controller进程可能占用大量内存，增加了CPU工作负荷，用户需要等待一段时间才能完成参数修改。系统管理员可以根据实际业务使用情况，手动增加Controller的JVM启动参数中内存参数，提升配置服务参数的效率。

对系统的影响

该操作需要在主管理节点重新启动Controller，重启期间会造成FusionInsight Manager暂时中断。备管理节点Controller无需重启。

前提条件

已确认主备管理节点IP。

操作步骤

1. 使用PuTTY，以**omm**用户登录主管理节点。
2. 执行以下命令，切换目录。

   cd ${BIGDATA_HOME}/om-server/om/sbin
3. 执行以下命令修改Controller启动参数文件“controller.sh”，并保存退出。

   vi controller.sh

   修改配置项“JAVA_HEAP_MAX”的参数值。例如，集群中包含了400个以上的节点，建议修改如下，表示Controller最大可使用8GB内存：

   JAVA_HEAP_MAX=-Xmx8192m
4. 执行以下命令，重新启动Controller。

   sh ${BIGDATA_HOME}/om-server/om/sbin/restart-controller.sh

   提示以下信息表示命令执行成功：

   End into start-controller.sh

   执行**sh ${BIGDATA_HOME}/om-server/om/sbin/status-oms.sh**，查看Controller的“ResHAStatus”是否为“Normal”，并可以重新登录FusionInsight Manager表示重启成功。
5. 使用PuTTY，以**omm**用户登录备管理节点，并重复步骤 2～步骤 3。

12.2.2 根据集群节点数优化Manager配置

操作场景

FusionInsight集群规模不同时，Manager相关参数差异较大。在集群容量调整前或者安装集群时，用户可以手动指定Manager集群节点数，系统将自动调整相关进程参数。

说明：

在安装集群时，可以通过Manager安装配置文件中的“cluster_nodes_scale”参数指定集群节点数。

操作步骤

1. 使用PuTTY，以**omm**用户登录主管理节点。
2. 执行以下命令，切换目录。

   cd ${BIGDATA_HOME}/om-server/om/sbin
3. 执行以下命令查看当前集群Manager相关配置。

   sh oms_config_info.sh -q
4. 执行以下命令指定当前集群的节点数。

   命令格式：**sh oms_config_info.sh -s** *节点数*

   例如：

   sh oms_config_info.sh -s 10

   根据界面提示，输入“y”：

   ```
   The following configurations will be modified:
     Module          Parameter                 Current              Target
     Controller      controller.Xmx            4096m             =>  8192m
     Controller      controller.Xms            1024m             =>  2048m
     ...
   Do you really want to do this operation? (y/n):
   ```

   界面提示以下信息表示配置更新成功：

   ```
   ...
   Operation has been completed. Now restarting OMS server.                [done]
   Restarted oms server successfully.
   ```

   说明：

- 配置更新过程中，OMS会自动重启。
- 相近数量的节点规模对应的Manager相关配置是通用的，例如100节点变为101节点，并没有新的配置项需要刷新。

12.3 HBase

12.3.1 提升BulkLoad效率

操作场景

批量加载功能采用了MapReduce jobs直接生成符合HBase内部数据格式的文件，然后把生成的StoreFiles文件加载到正在运行的集群。使用批量加载相比直接使用HBase的API会节约更多的CPU和网络资源。

ImportTSV是一个HBase的表数据加载工具。

前提条件

在执行批量加载时需要通过“Dimporttsv.bulk.output”参数指定文件的输出路径。

操作步骤

参数入口：执行批量加载任务时，在BulkLoad命令行中加入如下参数。

表12-1 增强BulkLoad效率的配置项

参数	描述	配置的值
–Dimporttsv.mapper.class	用户自定义mapper通过把键值对的构造从mapper移动到reducer以帮助提高性能。mapper只需要把每一行的原始文本发送给reducer，reducer解析每一行的每一条记录并创建键值对。 **说明：** 当该值配置为“org.apache.hadoop.hbase.mapreduce.TsvImporterByteMapper”时，只在执行没有*HBASE_CELL_VISIBILITY OR HBASE_CELL_TTL*选项的批量加载命令时使用。使用“org.apache.hadoop.hbase.mapreduce.TsvImporterByteMapper”时可以得到更好的性能。	org.apache.hadoop.hbase.mapreduce.TsvImporterByteMapper 和 org.apache.hadoop.hbase.mapreduce.TsvImporterTextMapper

12.3.2 提升连续put场景性能

操作场景

对大批量、连续put的场景，配置下面的两个参数为“false”时能大量提升性能。

- “hbase.regionserver.wal.durable.sync”
- “hbase.regionserver.hfile.durable.sync”

当提升性能时，缺点是对于DataNode（默认是3个）同时故障时，存在小概率数据丢失的现象。对数据可靠性要求高的场景请慎重配置。

操作步骤

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > HBase > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

表12-2 提升连续put场景性能的参数

参数	描述	默认值
hbase.regionserver.wal.durable.sync	每一条wal是否持久化到硬盘。	true
hbase.regionserver.hfile.durable.sync	hfile写是否立即持久化到硬盘。	true

12.3.3 Put和Scan性能综合调优

操作场景

HBase有很多与读写性能相关的配置参数。读写请求负载不同的情况下，配置参数需要进行相应的调整，本章节旨在指导用户通过修改RegionServer配置参数进行读写性能调优。

操作步骤

- JVM GC参数

  RegionServer GC_OPTS参数设置建议：

  - –Xms与–Xmx设置相同的值，需要根据实际情况设置，增大内存可以提高读写性能，可以参考参数“hfile.block.cache.size”（见表12–4）和参数“hbase.regionserver.global.memstore.size”（见表12–3）的介绍进行设置。
  - –XX:NewSize与–XX:MaxNewSize设置相同值，建议低负载场景下设置为“512M”，高负载场景下设置为“2048M”。
  - –XX:CMSInitiatingOccupancyFraction建议设置为“100 * (hfile.block.cache.size + hbase.regionserver.global.memstore.size + 0.05)”，最大值不超过90。

- -XX:MaxDirectMemorySize表示JVM使用的堆外内存，建议低负载情况下设置为“512M”，高负载情况下设置为“2048M”。

- Put相关参数

  RegionServer处理put请求的数据，会将数据写入memstore和hlog，

  - 当memstore大小达到设置的“hbase.hregion.memstore.flush.size”参数值大小时，memstore就会刷新到HDFS生成HFile。
  - 当当前region的列簇的HFile数量达到“hbase.hstore.compaction.min”参数值时会触发compaction。
  - 当当前region的列簇HFile数达到“hbase.hstore.blockingStoreFiles”参数值时会阻塞memstore刷新生成HFile的操作，导致put请求阻塞。

  表12-3 Put相关参数

参数	描述	默认值
hbase.regionserver.wal.durable.sync	每一条wal是否持久化到硬盘。 参考提升连续put场景性能。	true
hbase.regionserver.hfile.durable.sync	hfile写是否立即持久化到硬盘。 参考提升连续put场景性能。	true
hbase.hregion.memstore.flush.size	建议设置为HDFS块大小的整数倍，在内存足够put负载大情况下可以调整增大。单位：字节。	134217728
hbase.regionserver.global.memstore.size	建议设置为“hbase.hregion.memstore.flush.size * 写活跃region数 / RegionServer GC –Xmx”。默认值为“0.4”，表示使用RegionServer GC –Xmx的40%。	0.4
hbase.hstore.flusher.count	memstore的flush线程数，在put高负载场景下可以适当调大。	2
hbase.regionserver.thread.compaction.small	HFile compaction线程数，在put高负载情况下可以适当调大。	10
hbase.hstore.blockingStoreFiles	当列簇的HFile数达到该阈值，阻塞该region的所有操作，直到compcation完成，在put高负载场景下可以适当调大。	15

- Scan相关参数

  表12-4 Scan相关参数

参数	描述	默认值
hbase.client.scanner.timeout.period	客户端和RegionServer端参数，表示scan租约的时间，建议设置为60000ms的整数倍，在读高负载情况下可以适当调大。单位：毫秒。	60000
hfile.block.cache.size	数据缓存所占的RegionServer GC –Xmx百分比，在读高负载情况下可以适当调大以增大缓存命中率以提高性能。默认值为“0.25”，表示使用RegionServer GC –Xmx的25%。	0.25

- Handler相关参数

  表12-5 Handler相关参数

参数	描述	默认值
hbase.regionserver.handler.count	RegionServer上的RPC服务器实例数，建议设置为200 ~ 400之间。	200
hbase.regionserver.metahandler.count	RegionServer中处理优先请求的程序实例的数量，建议设置为200 ~ 400之间。	100

12.3.4 提升实时写数据效率

操作场景

需要把数据实时写入到HBase中或者对于大批量、连续put的场景。

前提条件

调用HBase的put或delete接口，把数据保存到HBase中。

操作步骤

- **写数据服务端调优**

  参数入口：

  在FusionInsight Manager系统中，选择“服务管理 > HBase > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

  表12-6 影响实时写数据配置项

配置参数	描述	默认值

配置参数	描述	默认值
hbase.regionserver.wal.durable.sync	控制HLog文件在写入到HDFS时的同步程度。如果为true，HDFS在把数据写入到硬盘后才返回；如果为false，HDFS在把数据写入OS的缓存后就返回。 把该值设置为false比true在写入性能上会更优。	true
hbase.regionserver.hfile.durable.sync	控制HFile文件在写入到HDFS时的同步程度。如果为true，HDFS在把数据写入到硬盘后才返回；如果为false，HDFS在把数据写入OS的缓存后就返回。 把该值设置为false比true在写入性能上会更优。	true
GC_OPTS	HBase利用内存完成读写操作。提高HBase内存可以有效提高HBase性能。GC_OPTS主要需要调整HeapSize的大小和NewSize的大小。调整HeapSize大小的时候，建议将Xms和Xmx设置成相同的值，这样可以避免JVM动态调整HeapSize大小的时候影响性能。调整NewSize大小的时候，建议把其设置为HeapSize大小的1/9。 • HMaster：当HBase集群规模越大、Region数量越多时，可以适当调大HMaster的GC_OPTS参数。 • RegionServer：RegionServer需要的内存一般比HMaster要大。在内存充足的情况下，HeapSize可以相对设置大一些。 **说明：** 主HMaster的HeapSize为4G的时候，HBase集群可以支持100000Region数的规模。根据经验值，单个RegionServer的HeapSize不建议超过20GB。	• HMaster： –Xms2G –Xmx2G –XX:NewSize=256M –XX:MaxNewSize=256M • RegionServer： –Xms4G –Xmx4G –XX:NewSize=512M –XX:MaxNewSize=512M
hbase.regionserver.handler.count	表示RegionServer在同一时刻能够并发处理多少请求。如果设置过高会导致激烈线程竞争，如果设置过小，请求将会在RegionServer长时间等待，降低处理能力。根据资源情况，适当增加处理线程数。 建议根据CPU的使用情况，可以选择设置为100至300之间的值。	200
hbase.hregion.max.filesize	表示HBase中Region的文件总大小的最大值。当Region中的文件大于该参数时，将会导致Region分裂。 该参数设置过小时，可能会导致Split操作过于频繁。当设置过大时，可能导致Compact需要处理的文件大小增加，影响Compact执行效率。	10737418240（单位：字节）
hbase.hregion.memstore.flush.size	在RegionServer中，当写操作内存中存在超过memstore.flush.size大小的memstore，则MemStoreFlusher就启动flush操作将该memstore以hfile的形式写入对应的store中。 如果RegionServer的内存充足，而且活跃Region数量也不是很多的时候，可以适当增大该值，可以减少compaction的次数，有助于提升系统性能。 同时，这种flush产生的时候，并不是紧急的flush，flush操作可能会有一定延迟，在延迟期间，写操作还可以进行，Memstore还会继续增大，最大值为“memstore.flush.size” * “hbase.hregion.memstore.block.multiplier”。当超过最大值时，将会阻塞操作。适当增大“hbase.hregion.memstore.block.multiplier”可以减少阻塞，减少性能波动。	134217728（单位：字节）
hbase.regionserver.global.memstore.size	RegionServer中，负责flush操作的是MemStoreFlusher线程。该线程定期检查写操作内存，当写操作占用内存总量达到阈值，MemStoreFlusher将启动flush操作，按照从大到小的顺序，flush若干相对较大的memstore，直到所占用内存小于阈值。 阈值 = “hbase.regionserver.global.memstore.size” * “hbase.regionserver.global.memstore.size.lower.limit” * “HBase_HEAPSIZE” **说明：** 该配置与“hfile.block.cache.size”的和不能超过0.8，也就是写和读操作的内存不能超过HeapSize的80%，这样可以保证除读和写外其它操作的正常运行。	0.4
hbase.hstore.blockingStoreFiles	在region flush前首先判断file文件个数，是否大于hbase.hstore.blockingStoreFiles。 如果大于需要先compaction并且让flush延时90s（这个值可以通过hbase.hstore.blockingWaitTime进行配置），在延时过程中，将会继续写从而使得Memstore还会继续增大超过最大值“memstore.flush.size” * “hbase.hregion.memstore.block.multiplier”，导致写操作阻塞。当完成compaction后，可能就会产生大量写入。这样就导致性能激烈震荡。 增加hbase.hstore.blockingStoreFiles，可以减低BLOCK几率。	15
hbase.regionserver.thread.compaction.throttle	控制一次Minor Compaction时，进行compaction的文件总大小的阈值。Compaction时的文件总大小会影响这一次compaction的执行时间，如果太大，可能会阻塞其它的compaction或flush操作。	1610612736（单位：字节）

配置参数	描述	默认值
hbase.hstore.compaction.min	当一个Store中文件超过该值时，会进行compact，适当增大该值，可以减少文件被重复执行compaction。但是如果过大，会导致Store中文件数过多而影响读取的性能。	6
hbase.hstore.compaction.max	控制一次compaction操作时的文件数量的最大值。 与“hbase.hstore.compaction.max.size”的作用基本相同，主要是控制一次compaction操作的时间不要太长。	10
hbase.hstore.compaction.max.size	如果一个HFile文件的大小大于该值，那么在Minor Compaction操作中不会选择这个文件进行compaction操作，除非进行Major Compaction操作。 这个值可以防止较大的HFile参与compaction操作。在禁止Major Compaction后，一个Store中可能存在几个HFile，而不会合并成为一个HFile，这样不会对数据读取造成太大的性能影响。	9223372036854775807（单位：字节）
hbase.hregion.majorcompaction	设置Major Compaction的执行周期。默认值为604800000毫秒。由于执行Major Compaction会占用较多的系统资源，如果正在处于系统繁忙时期，会影响系统的性能。 如果业务没有较多的更新、删除、回收过期数据空间时，可以把该值设置为0，以禁止Major Compaction。 如果必须要执行Major Compaction，以回收更多的空间，可以适当增加该值，同时配置参数“hbase.offpeak.end.hour”和“hbase.offpeak.start.hour”以控制Major Compaction发生在业务空闲的时期。	604800000（单位：毫秒）
• hbase.regionserver.maxlogs • hbase.regionserver.hlog.blocksize	• 表示一个RegionServer上未进行Flush的Hlog的文件数量的阈值，如果大于该值，RegionServer会强制进行flush操作。 • 表示每个HLog文件的最大大小。如果HLog文件大小大于该值，就会滚动出一个新的HLog文件，旧的将被禁用并归档。 这两个参数共同决定了RegionServer中可以存在的未进行Flush的hlog数量。当这个数据量小于MemStore的总大小的时候，会出现由于HLog文件过多而触发的强制flush操作。这个时候可以适当调整这两个参数的大小，以避免出现这种强制flush的情况。	• 32 • 134217728（单位：字节）

- **写数据客户端调优**

  写数据时，在场景允许的情况下，最好使用Put List的方式，可以极大的提升写性能。每一次Put的List的长度，需要结合单条Put的大小，以及实际环境的一些参数进行设定。建议在选定之前先做一些基础的测试。

- **写数据表设计调优**

  表12-7 影响实时写数据相关参数

配置参数	描述	默认值
COMPRESSION	配置数据的压缩算法，这里的压缩是HFile中block级别的压缩。对于可以压缩的数据，配置压缩算法可以有效减少磁盘的IO，从而达到提高性能的目的。 **说明：** 并非所有数据都可以进行有效压缩。例如一张图片的数据，因为图片一般已经是压缩后的数据，所以压缩效果有限。 常用的压缩算法是SNAPPY，因为它有较好的Encoding/Decoding速度和可以接受的压缩率。	NONE
BLOCKSIZE	配置HFile中block块的大小，不同的block块大小，可以影响HBase读写数据的效率。越大的block块，配合压缩算法，压缩的效率就越好；但是由于HBase的读取数据是以block块为单位的，所以越大的block块，对于随机读的情况，性能可能会比较差。 如果要提升写入的性能，一般扩大到128KB或者256KB，可以提升写数据的效率，也不会影响太大的随机读性能。	65536（单位：字节）
IN_MEMORY	配置这个表的数据优先缓存在内存中，这样可以有效提升读取的性能。对于一些小表，而且需要频繁进行读取操作的，可以设置此配置项。	false

12.3.5 提升实时读数据效率

操作场景

需要读取HBase数据场景。

前提条件

调用HBase的get或scan接口，从HBase中实时读取数据。

操作步骤

- **读数据服务端调优**

  参数入口：

  在FusionInsight Manager系统中，选择“服务管理 > HBase > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

  表12-8 影响实时写数据配置项

配置参数	描述	默认值
GC_OPTS	HBase利用内存完成读写操作。提高HBase内存可以有效提高HBase性能。 GC_OPTS主要需要调整HeapSize的大小和NewSize的大小。调整HeapSize大小的时候，建议将Xms和Xmx设置成相同的值，这样可以避免JVM动态调整HeapSize大小的时候影响性能。调整NewSize大小的时候，建议把其设置为HeapSize大小的1/9。 • HMaster：当HBase集群规模越大、Region数量越多时，可以适当调大HMaster的GC_OPTS参数。 • RegionServer：RegionServer需要的内存一般比HMaster要大。在内存充足的情况下，HeapSize可以相对设置大一些。 **说明：** 主HMaster的HeapSize为4G的时候，HBase集群可以支持100000Region数的规模。根据经验值，单个RegionServer的HeapSize不建议超过20GB。	• HMaster： –Xms2G –Xmx2G –XX:NewSize=256M –XX:MaxNewSize=256M • RegionServer： –Xms4G –Xmx4G –XX:NewSize=512M –XX:MaxNewSize=512M
hbase.regionserver.handler.count	表示RegionServer在同一时刻能够并发处理多少请求。如果设置过高会导致激烈线程竞争，如果设置过小，请求将会在RegionServer长时间等待，降低处理能力。根据资源情况，适当增加处理线程数。 建议根据CPU的使用情况，可以选择设置为100至300之间的值。	200
hfile.block.cache.size	HBase缓存区大小，主要影响查询性能。根据查询模式以及查询记录分布情况来决定缓存区的大小。如果采用随机查询使得缓存区的命中率较低，可以适当降低缓存区大小。	0.25

  说明：

  如果同时存在读和写的操作，这两种操作的性能会互相影响。如果写入导致的flush和Compaction操作频繁发生，会占用大量的磁盘IO操作，从而影响读取的性能。如果写入导致阻塞较多的Compaction操作，就会出现Region中存在多个HFile的情况，从而影响读取的性能。所以如果读取的性能不理想的时候，也要考虑写入的配置是否合理。

- **读数据客户端调优**

  Scan数据时需要设置caching（一次从服务端读取的记录条数，默认是1），若使用默认值读性能会降到极低。

  当不需要读一条数据所有的列时，需要指定读取的列，以减少网络IO。

  只读取RowKey时，可以为Scan添加一个只读取RowKey的filter（FirstKeyOnlyFilter或KeyOnlyFilter）。

- **读数据表设计调优**

  表12-9 影响实时读数据相关参数

配置参数	描述	默认值
COMPRESSION	配置数据的压缩算法，这里的压缩是HFile中block级别的压缩。对于可以压缩的数据，配置压缩算法可以有效减少磁盘的IO，从而达到提高性能的目的。 **说明：** 并非所有数据都可以进行有效压缩。例如一张图片的数据，因为图片一般已经是压缩后的数据，所以压缩效果有限。 常用的压缩算法是SNAPPY，因为它有较好的Encoding/Decoding速度和可以接受的压缩率。	NONE
BLOCKSIZE	配置HFile中block块的大小，不同的block块大小，可以影响HBase读写数据的效率。越大的block块，配合压缩算法，压缩的效率就越好；但是由于HBase的读取数据是以block块为单位的，所以越大的block块，对于随机读的情况，性能可能会比较差。 如果要提升写入的性能，一般扩大到128KB或者256KB，可以提升写数据的效率，也不会影响太大的随机读性能。	65536（单位：字节）
DATA_BLOCK_ENCODING	配置HFile中block块的编码方法。当一行数据中存在多列时，一般可以配置为“FAST_DIFF”，可以有效的节省数据存储的空间，从而提供性能。	NONE

12.3.6 JVM参数优化

操作场景

当集群数据量达到一定规模后，JVM的默认配置将无法满足集群的业务需求，轻则集群变慢，重则集群服务不可用。所以需要根据实际的业务情况进行合理的JVM参数配置，提高集群性能。

操作步骤

参数入口：

HBase角色相关的JVM参数需要配置在“${HBASE_HOME}/conf”目录下的“hbase-env.sh”文件中。

每个角色都有各自的JVM参数配置变量，如表12-10。

表12-10 HBase相关JVM参数配置变量

变量名	变量影响的角色
HBASE_OPTS	该变量中设置的参数，将影响HBase的所有角色。
SERVER_GC_OPTS	该变量中设置的参数，将影响HBase Server端的所有角色，例如：Master、RegionServer等。
CLIENT_GC_OPTS	该变量中设置的参数，将影响HBase的Client进程。
HBASE_MASTER_OPTS	该变量中设置的参数，将影响HBase的Master。
HBASE_REGIONSERVER_OPTS	该变量中设置的参数，将影响HBase的RegionServer。
HBASE_THRIFT_OPTS	该变量中设置的参数，将影响HBase的Thrift。

配置方式举例：

export HADOOP_NAMENODE_OPTS="-Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,N

12.4 HDFS

12.4.1 提升写性能

操作场景

在HDFS中，通过调整属性的值，使得HDFS集群更适应自身的业务情况，从而提升HDFS的写性能。

操作步骤

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > HDFS > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

表12-11 HDFS写性能优化配置

参数	描述	默认值
dfs.datanode.drop.cache.behind.reads	设置为true表示丢弃缓存的数据（需要在DataNode中配置）。 当同一份数据，重复读取的次数较少时，建议设置为true，使得缓存能够被其他操作使用。重复读取的次数较多时，设置为false能够提升重复读取的速度。	true
dfs.client-write-packet-size	当HDFS Client往DataNode写数据时，将数据生成一个包。然后将这个包在网络上传出。此参数指定传输数据包的大小，可以通过各Job来指定。单位：字节。 在万兆网部署下，可适当增大该参数值，来提升传输的吞吐量。	262144

12.4.2 JVM参数优化

操作场景

当集群数据量达到一定规模后，JVM的默认配置将无法满足集群的业务需求，轻则集群变慢，重则集群服务不可用。所以需要根据实际的业务情况进行合理的JVM参数配置，提高集群性能。

操作步骤

参数入口：

HDFS角色相关的JVM参数需要配置在“${HADOOP_HOME}/etc/hadoop”目录下的“hadoop-env.sh”文件中。

JVM各参数的含义请参见其官网：http://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html

每个角色都有各自的JVM参数配置变量，如表12-12。

表12-12 HDFS相关JVM参数配置变量

变量名	变量影响的角色
HADOOP_OPTS	该变量中设置的参数，将影响HDFS的所有角色。
HADOOP_NAMENODE_OPTS	该变量中设置的参数，将影响HDFS的NameNode。
HADOOP_DATANODE_OPTS	该变量中设置的参数，将影响HDFS的DataNode。
HADOOP_JOURNALNODE_OPTS	该变量中设置的参数，将影响HDFS的JournalNode。

变量名	变量影响的角色
HADOOP_ZKFC_OPTS	该变量中设置的参数，将影响HDFS的ZKFC。
HADOOP_SECONDARYNAMENODE_OPTS	该变量中设置的参数，将影响HDFS的SecondaryNameNode。
HADOOP_CLIENT_OPTS	该变量中设置的参数，将影响HDFS的Client进程。
HADOOP_BALANCER_OPTS	该变量中设置的参数，将影响HDFS的Balancer进程。
HADOOP_MOVER_OPTS	该变量中设置的参数，将影响HDFS的Mover进程。

配置方式举例：

export HADOOP_NAMENODE_OPTS="-Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,N

12.4.3 使用客户端元数据缓存提高读取性能

操作场景

通过使用客户端缓存元数据块的位置来提高HDFS读取性能。

说明：

此功能仅用于读取不经常修改的文件。因为在服务器端由某些其他客户端完成的数据修改，对于高速缓存的客户端将是不可见的，这可能导致从缓存中拿到的元数据是过期的。

操作步骤

设置参数的路径：

在FusionInsight Manager页面中，选择“服务管理 > HDFS > 服务配置”，将“参数类别”设置为“全部配置”，并在搜索框中输入参数名称。

表12-13 参数配置

参数	描述	默认值
dfs.client.metadata.cache.enabled	启用/禁用块位置元数据的客户端缓存。将此参数设置为“true”，搭配“dfs.client.metadata.cache.pattern”参数以启用缓存。	false
dfs.client.metadata.cache.pattern	需要缓存的文件路径的正则表达式模式。只有这些文件的块位置元数据被缓存，直到这些元数据过期。此配置仅在参数“dfs.client.metadata.cache.enabled”设置为“true”时有效。 示例：“/test.*”表示读取其路径是以“/test”开头的所有文件。 **说明：** • 为确保一致性，配置特定模式以仅缓存其他客户端不经常修改的文件。 • 正则表达式模式将仅验证URI的path部分，而不验证在Fully Qualified路径情况下的schema和authority。	<empty>
dfs.client.metadata.cache.expiry.sec	缓存元数据的持续时间。缓存条目在该持续时间过期后失效。即使在缓存过程中经常使用的元数据也会发生失效。 配置值可采用时间后缀s/m/h表示，分别表示秒，分钟和小时。 **说明：** 若将该参数配置为“0s”，将禁用缓存功能。	60s
dfs.client.metadata.cache.max.entries	缓存一次最多可保存的非过期数据条目。	65536

说明：

要在过期前完全清除客户端缓存，可调用*DFSClient#clearLocatedBlockCache()*。

用法如下所示。

```
FileSystem fs = FileSystem.get(conf);
DistributedFileSystem dfs = (DistributedFileSystem) fs;
DFSClient dfsClient = dfs.getClient();
dfsClient.clearLocatedBlockCache();
```

12.4.4 使用当前活动缓存提升客户端与NameNode的连接性能

操作场景

HDFS部署在具有多个NameNode实例的HA（High Availability）模式中，HDFS客户端需要依次连接到每个NameNode，以确定当前活动的NameNode是什么，并将其用于客户端操作。

一旦识别出来，当前活动的NameNode的详细信息就可以被缓存并共享给在客户端机器中运行的所有客户端。这样，每个新客户端可以首先尝试从缓存加载活动的Name Node的详细信息，并将RPC调用保存到备用的NameNode。在异常情况下有很多优势，例如当备用的NameNode连接长时间不响应时。

当发生故障，将另一个NameNode切换为活动状态时，缓存的详细信息将被更新为当前活动的NameNode的信息。

操作步骤

设置参数的路径如下：

在FusionInsight Manager页面中，选择“服务管理 > HDFS > 服务配置”，将“参数类别”设置为“全部配置”，并在搜索框中输入参数名称。

表12-14 配置参数

参数	描述	默认值
dfs.client.failover.proxy.provider.[nameservice ID]	配置客户端Failover proxy provider类，该类使用传递的协议创建NameNode proxy。该参数可以被配置为“org.apache.hadoop.hdfs.server.namenode.ha.BlackListingFailoverProxyProvider”或者“org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider”。	org.apache.hadoop.hd
dfs.client.failover.activeinfo.share.flag	启用缓存并将当前活动的NameNode的详细信息共享给其他客户端。若要启用缓存，需将其设置为“true”。	false
dfs.client.failover.activeinfo.share.path	指定将在机器中的所有客户端创建的共享文件的本地目录。如果要为不同用户共享缓存，该文件夹应具有必需的权限（如在给定目录中创建，读写缓存文件）。	/tmp
dfs.client.failover.activeinfo.share.io.timeout.sec	控制超时的可选配置。用于在读取或写入缓存文件时获取锁定。如果在该时间内无法获取缓存文件上的锁定，则放弃尝试读取或更新缓存。单位为秒。	5

说明：

由HDFS客户端创建的缓存文件必须由其他客户端重新使用。因此，这些文件永远不会从本地系统中删除。若禁用该功能，可能需要进行手动清理。

12.5 Hive

12.5.1 建立表分区

操作场景

Hive在做Select查询时，一般会扫描整个表内容，会消耗较多时间去扫描不关注的数据。此时，可根据业务需求及其查询维度，建立合理的表分区，从而提高查询效率。

操作步骤

1. 使用PuTTY工具，以root用户登录已安装Hive客户端的节点。
2. 执行以下命令，进入客户端安装目录，例如“/opt/client”。

   cd /opt/client
3. 执行***source bigdata_env***命令，配置客户端环境变量。
4. 在客户端中执行如下命令，执行登录操作。

   kinit 用户名
5. 执行以下命令登录客户端工具。

   beeline
6. 指定静态分区或者动态分区。
   - 静态分区：

     静态分区是手动输入分区名称，在创建表时使用关键字***PARTITIONED BY***指定分区列名及数据类型。应用开发时，使用***ALTER TABLE ADD PARTITION***语句增加分区，以及使用***LOAD DATA INTO PARTITON***语句将数据加载到分区时，只能静态分区。
   - 动态分区：通过查询命令，将结果插入到某个表的分区时，可以使用动态分区。

     动态分区通过在客户端工具执行如下命令来开启：

     set hive.exec.dynamic.partition=true

     动态分区默认模式是strict，也就是必须至少指定一列为静态分区，在静态分区下建立动态子分区，可以通过如下设置来开启完全的动态分区：

     set hive.exec.dynamic.partition.mode=nonstrict

说明：

1. 动态分区可能导致一个DML语句创建大量的分区，对应的创建大量新文件夹，对系统性能可能带来影响。
2. 在文件数量大的情况下，执行一个SQL语句启动时间较长，可以在执行SQL语句之前执行“set mapreduce.input.fileinputformat.list-status.num-threads = 100;”语句来缩短启动时间。“mapreduce.input.fileinputformat.list-status.num-threads”参数需要先添加到Hive的白名单才可设置。

12.5.2 Join优化

操作场景

使用Join语句时，如果数据量大，可能造成命令执行速度和查询速度慢，此时可进行Join优化。

Join优化可分为以下方式：

- Map Join
- Sort Merge Bucket Map Join
- Join顺序优化

Map Join

Hive的Map Join适用于能够在内存中存放下的小表（指表大小小于25MB），通过“hive.mapjoin.smalltable.filesize”定义小表的大小，默认为25MB。

Map Join的方法有两种：

- 使用/*+ MAPJOIN(join_table) */。
- 执行语句前设置如下参数，当前版本中该值默认为true。

```
set hive.auto.convert.join=true
```

使用Map Join时没有Reduce任务，而是在Map任务前起了一个MapReduce Local Task，这个Task通过TableScan读取小表内容到本机，在本机以HashTable的形式保存并写入硬盘上传到DFS，并在distributed cache中保存，在Map Task中从本地磁盘或者distributed cache中读取小表内容直接与大表join得到结果并输出。

使用Map Join时需要注意小表不能过大，如果小表将内存基本用尽，会使整个系统性能下降甚至出现内存溢出的异常。

Sort Merge Bucket Map Join

使用Sort Merge Bucket Map Join必须满足以下2个条件：

1. join的两张表都很大，内存中无法存放。
2. 两张表都按照join key进行分桶(clustered by (column))和排序(sorted by(column))，且两张表的分桶数正好是倍数关系。

通过如下设置，启用Sort Merge Bucket Map Join：

```
set hive.optimize.bucketmapjoin=true

 set hive.optimize.bucketmapjoin.sortedmerge=true
```

这种Map Join也没有Reduce任务，是在Map任务前启动MapReduce Local Task，将小表内容按桶读取到本地，在本机保存多个桶的HashTable备份并写入HDFS，并保存在Distributed Cache中，在Map Task中从本地磁盘或者Distributed Cache中按桶一个一个读取小表内容，然后与大表做匹配直接得到结果并输出。

Join顺序优化

当有3张及以上的表进行Join时，选择不同的Join顺序，执行时间存在较大差异。使用恰当的Join顺序可以有效缩短任务执行时间。

Join顺序原则：

- Join出来结果较小的组合，例如表数据量小或两张表Join后产生结果较少，优先执行。
- Join出来结果大的组合，例如表数据量大或两张表Join后产生结果较多，在后面执行。

例如，customer表的数据量最多，orders表和lineitem表优先Join可获得较少的中间结果。

原有的Join语句如下：

```
select
  l_orderkey,
  sum(l_extendedprice * (1 – l_discount)) as revenue,
  o_orderdate,
  o_shippriority
from
  customer,
  orders,
  lineitem
where
  c_mktsegment = 'BUILDING'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < '1995-03-22'
  and l_shipdate > '1995-03-22'
limit 10;
```

Join顺序优化后如下：

```
select
  l_orderkey,
  sum(l_extendedprice * (1 – l_discount)) as revenue,
  o_orderdate,
  o_shippriority
from
  orders,
  lineitem,
  customer
where
  c_mktsegment = 'BUILDING'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < '1995-03-22'
  and l_shipdate > '1995-03-22'
limit 10;
```

注意事项

Join数据倾斜问题

执行任务的时候，任务进度长时间维持在99%，这种现象叫数据倾斜。

数据倾斜是经常存在的，因为有少量的Reduce任务分配到的数据量和其他Reduce差异过大，导致大部分Reduce都已完成任务，但少量Reduce任务还没完成的情况。

解决数据倾斜的问题，可通过设置***set hive.optimize.skewjoin=true***并调整hive.skewjoin.key的大小。hive.skewjoin.key是指Reduce端接收到多少个key即认为数据是倾斜的，并自动分发到多个Reduce。

12.5.3 Group By优化

操作场景

优化Group by语句，可提升命令执行速度和查询速度。

Group by的时候， Map端会先进行分组， 分组完后分发到Reduce端， Reduce端再进行分组。可采用Map端聚合的方式来进行Group by优化，开启Map端初步聚合，减少Map的输出数据量。

操作步骤

在Hive客户端进行如下设置：

```
set hive.map.aggr=true
```

注意事项

Group By数据倾斜

Group By也同样存在数据倾斜的问题，设置hive.groupby.skewindata为true，生成的查询计划会有两个MapReduce Job，第一个Job的Map输出结果会随机的分布到Reduce中，每个Reduce做聚合操作，并输出结果，这样的处理会使相同的Group By Key可能被分发到不同的Reduce中，从而达到负载均衡，第二个Job再根据预处理的结果按照Group By Key分发到Reduce中完成最终的聚合操作。

Count Distinct聚合问题

当使用聚合函数count distinct完成去重计数时，处理值为空的情况会使Reduce产生很严重的数据倾斜，可以将空值单独处理，如果是计算count distinct，可以通过where字句将该值排除掉，并在最后的count distinct结果中加1。如果还有其他计算，可以先将值为空的记录单独处理，再和其他计算结果合并。

12.5.4 数据存储优化

操作场景

“ORC”是一种高效的列存储格式，在压缩比和读取效率上优于其他文件格式。

建议使用“ORC”作为Hive表默认的存储格式。

前提条件

已登录Hive客户端，具体操作请参见《管理员指南》的“使用Hive客户端”。

操作步骤

- 推荐：使用“SNAPPY”压缩，适用于压缩比和读取效率要求均衡场景。

```
Create table xx stored as orc tblproperties ("orc.compress"="SNAPPY")
```

- 可用：使用“ZLIB”压缩，适用于压缩比要求较高场景。

```
Create table xx stored as orc tblproperties ("orc.compress"="ZLIB")
```

说明：

xx为具体使用的Hive表名。

12.5.5 SQL优化

操作场景

在Hive上执行SQL语句查询时，如果语句中存在“(a&b) or (a&c)”逻辑时，建议将逻辑改为“a & (b or c)”。

样例

假设条件a为“p_partkey = l_partkey”，优化前样例如下所示：

```
select
    sum(l_extendedprice* (1 – l_discount)) as revenue
from
    lineitem,
    part
where
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#32'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 7 and l_quantity <= 7 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (   p_partkey = l_partkey
        and p_brand = 'Brand#35'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 15 and l_quantity <= 15 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (   p_partkey = l_partkey
```

```
            and p_brand = 'Brand#24'
            and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
            and l_quantity >= 26 and l_quantity <= 26 + 10
            and p_size between 1 and 15
            and l_shipmode in ('AIR', 'AIR REG')
            and l_shipinstruct = 'DELIVER IN PERSON'
    )
```

优化后样例如下所示：

```
select
    sum(l_extendedprice* (1 – l_discount)) as revenue
from
    lineitem,
    part
where   p_partkey = l_partkey and
    ((
        p_brand = 'Brand#32'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 7 and l_quantity <= 7 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_brand = 'Brand#35'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 15 and l_quantity <= 15 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_brand = 'Brand#24'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 26 and l_quantity <= 26 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    ))
```

12.5.6 使用Hive CBO优化查询

操作场景

在Hive中执行多表Join时，Hive支持开启CBO（Cost Based Optimization），系统会自动根据表的统计信息，例如数据量、文件数等，选出最优计划提高多表Join的效率。Hive需要先收集表的统计信息后才能使CBO正确的优化。

说明：

- CBO优化器会基于统计信息和查询条件，尽可能地使join顺序达到最优。但是也可能存在特殊情况导致join顺序调整不准确。例如数据存在倾斜，以及查询条件值在表中不存在等场景，可能调整出非优化的join顺序。
- 开启列统计信息自动收集时，需要在reduce侧做聚合统计。对于没有reduce阶段的insert任务，将会多出reduce阶段，用于收集统计信息。

前提条件

已登录Hive客户端，具体操作请参见《管理员指南》的“使用Hive客户端”。

操作步骤

1. 在Manager界面Hive组件的服务配置中搜索“hive.cbo.enable”参数，选中“true”永久开启功能或者通过以下命令临时开启功能：

   set hive.cbo.enable=true;

2. 手动收集Hive表已有数据的统计信息。

   执行以下命令，可以手动收集统计信息。仅支持统计一张表，如果需要统计不同的表需重复执行。

   ```
   ANALYZE TABLE [db_name.]tablename [PARTITION(partcol1[=val1], partcol2[=val2], ...)]
   COMPUTE STATISTICS
   [FOR COLUMNS]
   [NOSCAN];
   ```

   说明：

   - 指定FOR COLUMNS时，收集列级别的统计信息。
   - 指定NOSCAN时，将只统计文件大小和个数，不扫描具体文件。

   例如：

   analyze table table_name compute statistics;

   analyze table table_name compute statistics for columns;

3. 配置Hive自动收集统计信息。开启配置后，执行**insert overwrite/into**命令插入数据时才自动统计新数据的信息。
   - 在Hive客户端执行以下命令临时开启收集：

     set hive.stats.autogather = true; 开启表/分区级别的统计信息自动收集。

     set hive.stats.column.autogather = true; 开启列级别的统计信息自动收集。

     说明：
     - 列级别统计信息的收集不支持复杂的数据类型，例如Map，Struct等。
     - 表级别统计信息的自动收集不支持Hive on HBase表。

   - 在Manager界面Hive的服务配置中，搜索参数“hive.stats.autogather”和“hive.stats.column.autogather”，选中“true”永久开启收集功能。
4. 执行以下命令可以查看统计信息。

   DESCRIBE FORMATTED table_name[.column_name] PARTITION partition_spec;

   例如：

   desc formatted table_name;

   desc formatted table_name.id;

   desc formatted table_name.id partition(time='2016-05-27');

   说明：

   分区表仅支持分区级别的统计信息收集，因此分区表需要指定分区来查询统计信息。

12.6 Kafka

12.6.1 Kafka性能调优

操作场景

通过调整Kafka服务端参数，可以提升特定业务场景下Kafka的处理能力。

参数入口：在FusionInsight Manager系统中，选择“服务管理 > Kafka > 服务配置”，“参数类别”设置为“全部配置”。在搜索框中输入参数名称。

参数调优

表12-15 调优参数

配置参数	缺省值	调优场景
num.recovery.threads.per.data.dir	10	在Kafka启动过程中，数据量较大情况下，可调大此参数，可以提升启动速度。
background.threads	10	Broker后台任务处理的线程数目。数据量较大的情况下，可适当调大此参数，以提升Broker处理能力。
num.replica.fetchers	1	副本向Leader请求同步数据的线程数，增大这个数值会增加副本的I/O并发度。
num.io.threads	8	Broker用来处理磁盘I/O的线程数目，这个线程数目建议至少等于硬盘的个数。
KAFKA_HEAP_OPTS	-Xmx6G	Kafka JVM堆内存设置。当Broker上数据量较大时，应适当调整堆内存大小。

12.7 MapReduce

12.7.1 多CPU内核下的调优配置

操作场景

当CPU内核数很多时，如CPU内核为磁盘数的3倍时的调优配置。

操作步骤

以下参数有如下两个配置入口：

- 服务器端配置

  在FusionInsight Manager系统中，选择“服务管理 > YARN > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。
- 客户端配置

  直接在客户端中修改相应的配置文件。

  说明：
  - HDFS客户端配置文件路径：客户端安装目录/HDFS/hadoop/etc/hadoop/hdfs-site.xml。
  - Yarn客户端配置文件路径：客户端安装目录/HDFS/hadoop/etc/hadoop/yarn-site.xml。

- MapReduce客户端配置文件路径：客户端安装目录/HDFS/hadoop/etc/hadoop/mapred-site.xml。

表12-16 多CPU内核设置

配置	描述	参数	默认值	Server/Client
节点容器槽位数	如下配置组合决定了每节点任务(map、reduce)的并发数。 • “yarn.nodemanager.resource.memory-mb” • “mapreduce.map.memory.mb” • “mapreduce.reduce.memory.mb”	yarn.nodemanager.resource.memory-mb **说明：** 需要在FusionInsight Manager系统进行配置。	8192	Server
		mapreduce.map.memory.mb **说明：** 需要在客户端进行配置，配置文件路径：*客户端安装目录*/HDFS/hadoop/etc/hadoop/mapred-site.xml。	4096	Client
		mapreduce.reduce.memory.mb **说明：** 需要在客户端进行配置，配置文件路径：*客户端安装目录*/HDFS/hadoop/etc/hadoop/mapred-site.xml。	4096	Client
Map输出与压缩	Map任务所产生的输出可以在写入磁盘之前被压缩，这样可以节约磁盘空间并得到更快的写盘速度，同时可以减少至Reducer的数据传输量。需要在客户端进行配置。 • mapreduce.map.output.compress指定了Map任务输出结果可以在网络传输前被压缩。这是一个per-job的配置。 • mapreduce.map.output.compress.codec指定用于压缩的编解码器。	mapreduce.map.output.compress **说明：** 需要在客户端进行配置，配置文件路径：*客户端安装目录*/HDFS/hadoop/etc/hadoop/mapred-site.xml。	true	Client
		mapreduce.map.output.compress.codec **说明：** 需要在客户端进行配置，配置文件路径：*客户端安装目录*/HDFS/hadoop/etc/hadoop/mapred-site.xml。	org.apache.hadoop.io.compress.SnappyCodec	Client
Spills	mapreduce.map.sort.spill.percent	mapreduce.map.sort.spill.percent **说明：** 需要在客户端进行配置，配置文件路径：*客户端安装目录*/HDFS/hadoop/etc/hadoop/mapred-site.xml。	0.8	Client
数据包大小	当HDFS客户端写数据至数据节点时，数据会被累积，直到形成一个包。然后这个数据包会通过网络传输。dfs.client-write-packet-size配置项可以指定该数据包的大小。这个可以通过每个job进行指定。	dfs.client-write-packet-size **说明：** 需要在客户端进行配置，配置文件路径：*客户端安装目录*/HDFS/hadoop/etc/hadoop/hdfs-site.xml。	262144	Client

12.7.2 确定Job基线

操作场景

确定Job基线是调优的基础，一切调优项效果的检查，都是通过和基线数据做对比来获得。

Job基线的确定有如下三个原则：

- 充分利用集群资源
- reduce阶段尽量放在一轮
- 每个task的执行时间要合理

操作步骤

- **原则一：充分利用集群资源。**

Job运行时，会让所有的节点都有任务处理，且处于繁忙状态，这样才能保证资源充分利用，任务的并发度达到最大。可以通过调整处理的数据量大小，以及调整map和reduce个数来实现。

Reduce个数的控制使用“mapreduce.job.reduces”。

Map个数取决于使用了哪种InputFormat，以及待处理的数据文件是否可分割。默认的TextFileInputFormat将根据block的个数来分配map数(一个block一个map)。通过如下配置参数进行调整。

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > YARN > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

参数	描述	默认值
mapreduce.input.fileinputformat.split.maxsize	可以设置数据分片的数据最大值。 由用户定义的分片大小的设置及每个文件block大小的设置，可以计算分片的大小。计算公式如下： splitSize = Math.max(minSize, Math.min(maxSize, blockSize)) 如果maxSize设置大于blockSize，那么每个block就是一个分片，否则就会将一个block文件分隔为多个分片，如果block中剩下的一小段数据量小于splitSize，还是认为它是独立的分片。	-
mapreduce.input.fileinputformat.split.minsize	可以设置数据分片的数据最小值。	0

- **原则二：控制reduce阶段在一轮中完成。**

  避免以下两种场景：

  - 大部分的reduce在第一轮运行完后，剩下唯一一个reduce继续运行。这种情况下，这个reduce的执行时间将极大影响这个job的运行时间。因此需要将reduce个数减少。
  - 所有的map运行完后，只有个别节点有reduce在运行。这时候集群资源没有得到充分利用，需要增加reduce的个数以便每个节点都有任务处理。

- **原则三：每个task的执行时间要合理。**

  如果一个job，每个map或reduce的执行时间只有几秒钟，就意味着这个job的大部分时间都消耗在task的调度和进程启停上了，因此需要增加每个task处理的数据大小。建议一个task处理时间为1分钟。

  控制单个task处理时间的大小，可以通过如下配置来调整。

  参数入口：

  在FusionInsight Manager系统中，选择“服务管理 > YARN > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

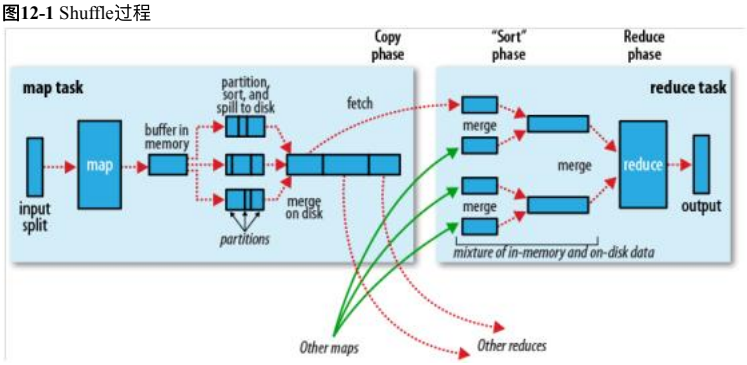
参数	描述	默认值
mapreduce.input.fileinputformat.split.maxsize	可以设置数据分片的数据最大值。 由用户定义的分片大小的设置及每个文件block大小的设置，可以计算分片的大小。计算公式如下： splitSize = Math.max(minSize, Math.min(maxSize, blockSize)) 如果maxSize设置大于blockSize，那么每个block就是一个分片，否则就会将一个block文件分隔为多个分片，如果block中剩下的一小段数据量小于splitSize，还是认为它是独立的分片。	-
mapreduce.input.fileinputformat.split.minsize	可以设置数据分片的数据最小值。	0

12.7.3 Shuffle调优

操作场景

Shuffle阶段是MapReduce性能的关键部分，包括了从Map task将中间数据写到磁盘一直到Reduce task拷贝数据并最终放到reduce函数的全部过程。这一块Hadoop提供了大量的调优参数。

图12-1 Shuffle过程



操作步骤

1. **Map阶段的调优**

   - 判断Map使用的内存大小

     判断Map分配的内存是否足够，一个简单的办法是查看运行完成的job的Counters中，对应的task是否发生过多次GC，以及GC时间占总task运行时间之比。通常，GC时间不应超过task运行时间的10%，即GC time elapsed (ms)/CPU time spent (ms)<10%。

     主要通过如下参数进行调整。

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > Yarn > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

表12-17 参数说明

参数	描述	默认值
mapreduce.map.memory.mb	设置Map调度内存。	4096
mapreduce.map.java.opts	设置Map进程JVM参数。	-Xmx2048M -Djava.net.preferIPv4Stack=true

建议：配置“mapreduce.map.java.opts”参数中“-Xmx”值为“mapreduce.map.memory.mb”参数值的0.8倍。

- 使用Combiner

  在Map阶段，有一个可选过程，将同一个key值的中间结果合并，叫做combiner。一般将reduce类设置为combiner即可。通过combine，一般情况下可以显著减少Map输出的中间结果，从而减少shuffle过程的网络带宽占用。可通过如下接口为一个任务设置Combiner类。

  表12-18 Combiner设置接口

类名	接口名	描述
org.apache.hadoop.mapreduce.Job	public void setCombinerClass(Class<? extends Reducer> cls)	为Job设置一个combiner类。

2. **Copy阶段的调优**

   - 数据是否压缩

     对Map的中间结果进行压缩，当数据量大时，会显著减少网络传输的数据量，但是也因为多了压缩和解压，带来了更多的CPU消耗。因此需要做好权衡。当任务属于网络瓶颈类型时，压缩Map中间结果效果明显。针对bulkload调优，压缩中间结果后性能提升60%左右。

     配置方法：将“mapreduce.map.output.compress”参数值设置为“true”，将“mapreduce.map.output.compress.codec”参数值设置为“org.apache.hadoop.io.compress.SnappyCodec”。

3. **Merge阶段的调优**

   通过调整如下参数减少reduce写磁盘的次数。

   参数入口：

   在FusionInsight Manager系统中，选择“服务管理 > YARN > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

   表12-19 参数说明

参数	描述	默认值
mapreduce.reduce.merge.inmem.threshold	允许多少个文件同时存在reduce内存里。当达到这个阈值时，reduce就会触发mergeAndSpill，将数据写到硬盘上。	1000
mapreduce.reduce.shuffle.merge.percent	当reduce中存放map中间结果的buffer使用达到多少百分比时，会触发merge操作。	0.66
mapreduce.reduce.shuffle.input.buffer.percent	允许map中间结果占用reduce堆大小的百分比。	0.70
mapreduce.reduce.input.buffer.percent	当开始执行reduce函数时，允许map文件占reduce堆大小的百分比。 当map文件比较小时，可以将这个值设置成1.0，这样可以避免reduce将拷贝过来的map中间结果写磁盘。	0

12.7.4 大任务的AM调优

操作场景

任务场景：运行的一个大任务（map总数达到了10万的规模），但是一直没有跑成功。经过查询，发现是ApplicationMaster（以下简称AM）反应缓慢，最终超时失败。

此任务的问题是，task数量变多时，AM管理的对象也线性增长，因此就需要更多的内存来管理。AM默认分配的内存堆大小是1GB。

操作步骤

通过调大如下的参数来进行AM调优。

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > Yarn > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

参数	描述	默认值
yarn.app.mapreduce.am.resource.mb	该参数值必须大于下面参数的堆大小。单位：MB	1536
yarn.app.mapreduce.am.command-opts	传递到MapReduce ApplicationMaster的JVM启动参数。	-Xmx1024m -XX:CMSFullGCsBeforeCompaction=1 -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -verbose:gc

12.7.5 推测执行

操作场景

当集群规模很大时（如几百上千台节点的集群），个别机器出现软硬件故障的概率就变大了，并且会因此延长整个任务的执行时间（跑完的任务都在等出问题的机器跑结束）。推测执行通过将一个task分给多台机器跑，取先运行完的那个，会很好的解决这个问题。对于小集群，可以将这个功能关闭。

操作步骤

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > YARN > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

参数	描述	默认值
mapreduce.map.speculative	是否开启map的推测执行。true表示开启。	false
mapreduce.reduce.speculative	是否开启reduce的推测执行。true表示开启。	false

12.7.6 通过“Slow Start”调优

操作场景

Slow Start特性指定Map任务完成度为多少时Reduce任务可以启动，过早启动Reduce任务会导致资源占用，影响任务运行效率，但适当的提早启动Reduce任务会提高Shuffle阶段的资源利用率，提高任务运行效率。例如：某集群可启动10个Map任务，MapReduce作业共15个Map任务，那么在一轮Map任务执行完成后只剩5个Map任务，集群还有剩余资源，在这种场景下，配置Slow Start参数值小于1，比如0.8，则Reduce就可以利用集群剩余资源。

操作步骤

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > MapReduce > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

参数	描述	默认值
mapreduce.job.reduce.slowstart.completedmaps	当多少占比的Map执行完后开始执行Reduce。默认100%的Map跑完后开始起Reduce。	1

12.7.7 MR job commit阶段优化

操作场景

默认情况下，如果一个MR任务会产生大量的输出结果文件，那么该job在最后的commit阶段会耗费较长的时间将每个task的临时输出结果commit到最终的结果输出目录。特别是在大集群中，大Job的commit过程会严重影响任务的性能表现。

针对以上情况，可以通过将以下参数“mapreduce.fileoutputcommitter.algorithm.version”配置为“2”，来提升MR Job commit阶段的性能。

操作步骤

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > Yarn > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

表12-20 参数说明

参数	描述	默认值
mapreduce.fileoutputcommitter.algorithm.version	用于指定Job的最终输出文件提交的算法版本，取值为“1”或“2”。 **说明：** 版本2为建议的优化算法版本。该算法通过让任务直接将每个task的输出结果提交到最终的结果输出目录，从而减少大作业的输出提交时间。	2

12.8 Solr

12.8.1 索引集分片划分建议

操作场景

该操作指导系统管理员通过调整Solr的实例数和索引集（Collection）的分片数（Shard）来提高Solr的索引性能。

操作步骤

- 调整Solr的实例数

  如有8个数据节点，建议按照如下方案进行规划：

  当索引数据存放在本地磁盘时：

  - SolrServerAdmin角色占用2个节点，其中每个节点上再部署SolrServerN(N=1～5)，共计12个实例；
  - 其余6个节点，每个节点分别部署SolrServerN(N=1～5)，共计30个实例。
  - 共计42个实例

  当索引数据存放在HDFS时：

- SolrServerAdmin角色占用2个节点，其中每个节点上再部署SolrServerN(N=1～2)，共计3个实例；
- 其余6个节点，每个节点分别部署SolrServerN(N=1～3)，共计18个实例。
- 共计24个实例

设置每个实例占用系统最大内存为4GB。如果内存资源足够，有利于索引性能提升。

调整内存占用的步骤如下：

1. 登录FusionInsight Manager系统。
2. 单击“服务管理 > Solr > 服务配置”。
3. 调节每个实例的“SOLR_GC_OPTS”参数，“-Xms2G -Xmx4G”部分表示占用内存大小为2GB～4GB。将此参数值修改为“-Xms4G -Xmx4G”。
4. 单击“保存配置”，在弹出的对话框中选择“重新启动受影响的服务或实例。”重启服务后生效。

- 调整Solr索引集的分片数

  在创建HDFS数据和HBase数据的全文检索索引集时（具体操作可参见“业务操作指南 > Solr > 业务常见操作”），建议按照以下方案进行规划：

  设置每个实例对应1个Shard，每个shard包含两个replica（即副本因子为2），当要求副本数为2时，创建包含21个shard的collection，每个shard两个副本分布于两个不同节点实例，这样42个实例分别包含一个replica。

  根据以上方案，每个索引集包含42个Shard（以部署42个实例为例）。Shard数量越大，有助于查询或索引的性能提升，但同时也会增加服务的交互操作，消耗更多系统资源。

  当索引数据存放在本地磁盘时，同样参考以上规则进行规划，也就是每个索引集包含24个shard(以部署24个实例为例)。

12.8.2 Solr公共读写调优建议

操作场景

该操作指导系统管理员对Solr可进行的公共读写性能调优。

前提步骤

已安装Solr服务的客户端。

操作步骤

当Solr索引数据存放在本地磁盘或者HDFS时，可以从以下几个方面进行调优配置：

Solr作为全文检索服务器时，从Solr自身相关配置（JVM、Schema、Solrconfig）方面进行调优。

1. Schema的配置优化
   - uniqueKey定义为long类型

     说明：

     - long类型的查询性能优于string类型，如果需要定义为string类型，可以在业务层建立long到string的映射。
     - 建议unqiueKey字段配置required="true"。
     - 建议uniqueKey字段配置docValues="true"。
     - 为了获得更好的查询性能，建议查询时显式地指定返回字段为uniqueKey字段。

   - 需要排序、统计的字段配置为docValues="true"，可以有效节省内存使用

     说明：

     配置docValues="true"后，不需要再配置stored="true"

2. 优化索引方案

   可以通过修改“solrconfig.xml”文件内容来实现以下调优效果。

调优配置项	修改结果
提高索引速度，加大索引线程数	<maxIndexingThreads>${solr.maxIndexingThreads:16}</maxIndexingThreads>
增大文档索引缓存	<ramBufferSizeMB>1024</ramBufferSizeMB>
增大索引段合并因子	<mergeFactor>20</mergeFactor>
加大索引自动硬提交时间	<maxTime>${solr.autoCommit.maxTime:30000}</maxTime>
增大索引的自动软提交时间	<maxTime>${solr.autoSoftCommit.maxTime:60000}</maxTime>
基于docValues获取uniqueKey的值	<useDocValueGetField>true</useDocValueGetField>
对docId进行排序，优先顺序读取磁盘	<sortDocIdBeforeGetDoc>true</sortDocIdBeforeGetDoc>
缓存docId，避免二次读取磁盘	<useQuickFirstMatch>true</useQuickFirstMatch>

   说明：

   useDocValueGetField的使用场景：

- 返回字段（fl）为uniqueKey
- uniqueKey为Numberic类型（long/int/float/double）
- uniqueKey配置docValues=true

useQuickFirstMatch的使用场景：

- 索引入库后不再更改（删除/合并操作等）

---

3. 优化查询方案

缓存在Solr中充当了一个非常重要的角色，Solr中主要包括以下3种缓存：

- Filter cache（过滤器缓存），用于保存过滤器（fq参数）和层面搜索的结果；
- Document cache（文档缓存），用于保存lucene文档存储的字段；
- Query result（查询缓存），用于保存查询的结果。

> **说明：**
> Solr中还包含lucene内部缓存，该缓存用户无法调控。

通过调整这3种缓存，可以对Solr的搜索实例进行调优。在调整参数前，需要事先得到Solr实例中的以下信息：

- 索引中文档的数量：单击“服务管理 > Solr”，在“Solr WebUI”下单击任意“SolrServerAdmin(XX)”进入Solr Admin页面。在“Core Selector”中选择索引数据集Collection下的任意一个core，单击“Query > Execute Query”，“numFound”项为该参数值；
- 过滤器的数量：用户期望值（如：200）；
- 一次查询返回最大的文档数量：用户期望值（如：100）；
- 不同查询和不同排序的个数：用户期望值（如：500）；
- 每次查询字段数：用户期望值（如：3）；
- 实例查询的并发数：用户期望值（如：10）。

可以通过修改“solrconfig.xml”文件内容（修改方法参考3）配置缓存：

缓存类型	**修改方案**
过滤器缓存	<filterCache class="solr.FastLRUCache" size="200" initialSize="200" autowarmCount="50"/> • “size”和“initialSize”值为缓存document id的数量。 • “autowarmCount”为“initialSiz”值的1/4。 • 根据实际场景合理设置，过大会占用大量内存。
查询结果缓存	<queryResultCache class="solr.FastLRUCache" size="3000" initialSize="3000" autowarmCount="750"/> • “size”和“initialSize”的值 = *不同查询和不同排序的个数*×*每次查询字段数*×2。 • “autowarmCount”值为“initialSize”值的1/4。
文档缓存	<documentCache class="solr.FastLRUCache" size="1000" initialSize="1000"/> “size”和“initialSize”的值 = *一次查询返回最大的文档数量*×*实例查询的并发数*。

12.8.3 Solr over HBase调优建议

操作场景

该操作指导系统管理员在使用Solr over HBase相关功能时，对环境配置进行调优。

前提条件

已成功安装HDFS、Solr、Yarn、HBase服务。

操作步骤

使用Solr over HBase相关功能时，可以从以下几个方面进行调优配置：

- 操作系统优化

  如果Solr索引在HDFS上，参考《Solr over HDFS调优建议》章节操作步骤中操作系统优化小节进行配置。

- 实时索引相关调优建议

1. 修改collection配置集的“solrconfig.xml”配置文件中，**<autoSoftCommit>**配置项，根据使用场景尽量设置大一些，设置越大索引效率越高。
2. 修改HBase服务配置，然后重启HBase服务：

   “replication.source.nb.capacity”：5000（HBase集群每次向HBaseIndexer发送的entry最大的个数，推荐5000。可根据集群规模做出适当调整，根据HBaseIndexer部署情况适当增大）。

   “replication.source.size.capacity”：4194304（HBase每次向HBaseIndexer发送的entry包的最大值大小，不推荐过大）。
3. 修改HDFS服务配置，然后重启HDFS服务：

“hadoop.rpc.protection”：authentication（关闭数据传输加密，默认为privacy）。

“ipc.server.handler.queue.size”：队列中允许的每个处理程序可处理的调用数，根据集群环境适当调整。

“dfs.namenode.handler.count”：NameNode的服务器线程数，根据集群环境适当调整。

“dfs.namenode.service.handler.count”：NameNode的服务器线程数，根据集群环境适当调整。

“dfs.datanode.handler.count”：DataNode的服务线程数，根据集群环境适当调整。

4. 修改HBaseIndexer服务配置，然后重启HBaseIndexer实例：

“hbaseindexer.indexer.threads”：50（默认值为20，HBaseIndexer实例进行索引操作时启动的并发线程数量）。

调整“GC_OPTS”至4G，内存空间充足，可以考虑适当增加。

- 批量索引、增量索引相关调优建议

1. SolrServer的GC参数配置（如果内存充足可以考虑增大）：**-Xmx8G -Xms8G**
2. Yarn的配置，修改后重启Yarn服务：

“mapreduce.reduce.memory.mb”：8192（根据节点配置进行修改）

“yarn.resourcemanager.scheduler.class”：org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler

3. 修改HBaseIndexer的配置文件，进入“Solr客户端的安装目录/hbase-indexer/”，执行一下命令修改配置文件：

vi /opt/client/Solr/hbase-indexer/conf/hbase-indexer-site.xml

设置“hbase-indexer-site.xml”部分以下几个参数的参数值（如某些参数项不存在，可手动添加）：

表12-21 “hbase-indexer-site.xml”配置文件修改

参数（name）	参数值（value）
solr.record.writer.batch.size	500
solr.record.writer.max.queues.size	300
solr.record.writer.num.threads	5
solr.record.writer.maxSegments	5

vi /opt/client/Solr/hbase-indexer/conf/yarn-site.xml

设置“yarn-site.xml”部分以下几个参数的参数值（如某些参数项不存在， 可手动添加）：

表12-22 “yarn-site.xml”配置文件修改

参数（name）	参数值（value）
mapreduce.map.speculative	false
mapreduce.reduce.speculative	false

12.8.4 Solr over HDFS调优建议

操作场景

该操作指导系统管理员在使用Solr over HDFS相关功能时，对环境配置进行调优。

前提条件

已成功安装HDFS、Solr、Yarn服务，且FusionInsight Manager页面内Solr服务的服务配置参数“INDEX_STORED_ON_HDFS”为“TRUE”。

完成使用Solr over HDFS的准备工作。

操作步骤

使用Solr over HDFS时，可以选择从以下几个方面进行调优配置：

- 磁盘和网络规划

1. 磁盘划分的时候要注意，ZooKeeper单独占用一个磁盘或一个磁盘分区，否则当数据量过大，如果和HDFS共磁盘，频繁的数据访问和配置集访问，会导致ZooKeeper停止响应。
2. HDFS采用多块单盘或者多个RAID0分别挂载多个数据目录，MR任务处理数据量很大时磁盘IO会成为瓶颈。
3. 网络组网模式，当前环境为多网卡绑定模式bond0，网络组网模式可根据实际情况而定，不一定要求为多网卡绑定模式。

- 操作系统优化

每台主机上都要执行，由于Solr在执行读取HDFS时，会生成很多个临时端口（连接本地DataNode。登录FusionInsight Manager，单击“服务管理 > HDFS > 服务配置”，查看“dfs.datanode.port”，其值为“DataNode Port”。执行**netstat -anp | grep** *DataNode Port* **| wc -l**命令，结果大于4096时。），出现TIME_WAIT，最终导致任务失败，为避免此种情况，需要如下设置

使用PuTTY客户端，登录每个节点，进入Solr客户端所在目录，执行以下命令：

vi /etc/sysctl.conf

添加以下内容：

```
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 30
```

net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_tw_recycle = 1

保存退出后，执行

sysctl –p

- Solr实例部署，索引存放到HDFS上时，确保所有Solr实例同DataNode部署在相同的节点上。
- 对Solr提交的MapReduce任务的性能优化，请参考节点配置调优和JVM参数优化。

  可参考如下建议，设置Yarn服务的配置参数，然后重启Yarn服务：

  “mapreduce.task.timeout”：1800000（原始值为60000，处理大数据量时可以适当调大）。

  “yarn.nodemanager.resource.cpu-vcores”：24（原始值为8，该值在处理大数量时可设置为当前节点总的CPU个数的1~2倍）。

  “yarn.nodemanager.resource.memory-mb”：该参数最好对每个nodemanaegr分别配置，看一下主机界面每个节点的内存使用率，每个nodemanager该参数配置为：空闲的内存数减去8G，或者为总内存的75%。
- Solr使用“HDFSDirectoryFactory”索引方式读写HDFS上的索引文件时，从Solr作为HDFS的Client方面的配置进行调优。

  从Solr作为HDFS的Client方面的配置进行调优时，如果Solr和其他组件要部署在相同的节点上，建议每个节点上只部署一个Solr实例。
- 调整HDFS缓存提高索引性能。Solr中HDFS相关缓存，通常分配系统上可用内存量的10-20%。例如，在有128GB内存的主机上运行HDFS和Solr时，通常分配12.8GB～25.6GB的内存作为HDFS相关缓存。随着索引大小的增加，需要调整此参数以保持最佳性能。该参数可通过修改“solrconfig.xml”文件进行配置。

  请按照以下步骤，调整分配的缓存大小：

  1. 使用PuTTY工具，以root用户登录Solr客户端所在节点，进入Solr客户端所在目录，执行以下命令：

     source bigdata_env

     kinit solr
  2. 执行以下命令获取配置文件集，并打开“solrconfig.xml”文件：

     solrctl confset --get confWithSchema /home/solr/

     vi /home/solr/conf/solrconfig.xml
  3. 修改参数“solr.hdfs.blockcache.slab.count”的值“<int name="solr.hdfs.blockcache.slab.count">${solr.hdfs.blockcache.slab.count:1}</int>”。

     其中每个slab的大小为128MB ，18个slab，大约占用2.3GB内存，这是每个分片Shard的配置，如果一个台主机6个shard，那么共计占用13.8GB内存。

     由此可知，在此例中可将改参数值修改为“<int name="solr.hdfs.blockcache.slab.count">${solr.hdfs.blockcache.slab.count:18}</int>”。
  4. 执行以下命令，上传修改过的配置文件集：

     solrctl confset --update confWithSchema /home/solr/

12.9 Spark

12.9.1 Spark Core调优

12.9.1.1 数据序列化

操作场景

Spark支持两种方式的序列化：

- Java原生序列化JavaSerializer
- Kryo序列化KryoSerializer

序列化对于Spark应用的性能来说，具有很大的影响。在特定的数据格式的情况下，KryoSerializer的性能可以达到JavaSerializer的10倍以上，而对于一些Int之类的基本类型数据，性能的提升就几乎可以忽略。

KryoSerializer依赖Twitter的Chill库来实现，相对于JavaSerializer，主要的问题在于不是所有的Java Serializable对象都能支持，兼容性不好，所以需要手动注册类。

序列化功能用在两个地方：序列化任务和序列化数据。Spark任务序列化只支持JavaSerializer，数据序列化支持JavaSerializer和KryoSerializer。

操作步骤

Spark程序运行时，在shuffle和RDD Cache等过程中，会有大量的数据需要序列化，默认使用JavaSerializer，通过配置让KryoSerializer作为数据序列化器来提升序列化性能。

在开发应用程序时，添加如下代码来使用KryoSerializer作为数据序列化器。

- 实现类注册器并手动注册类。

```
package com.etl.common;

import com.esotericsoftware.kryo.Kryo;
import org.apache.spark.serializer.KryoRegistrator;

public class DemoRegistrator implements KryoRegistrator
{
   @Override
   public void registerClasses(Kryo kryo)
   {
      //以下为示例类，请注册自定义的类
      kryo.register(AggrateKey.class);
      kryo.register(AggrateValue.class);
   }
}
```

您可以在Spark客户端对spark.kryo.registrationRequired参数进行配置，设置是否需要Kryo注册序列化。

当参数设置为true时，如果工程中存在未被序列化的类，则会抛出异常。如果设置为false（默认值），Kryo会自动将未注册的类名写到对应的对象中。此操作会对系统性能造成影响。设置为true时，用户需手动注册类，针对未序列化的类，系统不会自动写入类名，而是抛出异常，相对比false，其性能较好。

- 配置KryoSerializer作为数据序列化器和类注册器。

  ```
  val conf = new SparkConf()
  conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
  .set("spark.kryo.registrator", "com.etl.common.DemoRegistrator")
  ```

12.9.1.2 配置内存

操作场景

Spark是内存计算框架，计算过程中内存不够对Spark的执行效率影响很大。可以通过监控GC（Garbage Collection），评估内存中RDD的大小来判断内存是否变成性能瓶颈，并根据情况优化。

监控节点进程的GC情况（在客户端的conf/spark-default.conf配置文件中，在spark.driver.extraJavaOptions和spark.executor.extraJavaOptions配置项中添加参数："-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps"

），如果频繁出现Full GC，需要优化GC。把RDD做Cache操作，通过日志查看RDD在内存中的大小，如果数据太大，需要改变RDD的存储级别来优化。

操作步骤

- 优化GC，调整老年代和新生代的大小和比例。在客户端的conf/spark-default.conf配置文件中，在spark.driver.extraJavaOptions和spark.executor.extraJavaOptions配置项中添加参数：-XX:NewRatio。如，" -XX:NewRatio=2"，则新生代占整个堆空间的1/3，老年代占2/3。
- 开发Spark应用程序时，优化RDD的数据结构。
  - 使用原始类型数组替代集合类，如可使用fastutil库。
  - 避免嵌套结构。
  - Key尽量不要使用String。
- 开发Spark应用程序时，建议序列化RDD。

  RDD做cache时默认是不序列化数据的，可以通过设置存储级别来序列化RDD减小内存。例如：

  ```
  testRDD.persist(StorageLevel.MEMORY_ONLY_SER)
  ```

12.9.1.3 设置并行度

操作场景

并行度控制任务的数量，影响shuffle操作后数据被切分成的块数。调整并行度让任务的数量和每个任务处理的数据与机器的处理能力达到最优。

查看CPU使用情况和内存占用情况，当任务和数据不是平均分布在各节点，而是集中在个别节点时，可以增大并行度使任务和数据更均匀的分布在各个节点。增加任务的并行度，充分利用集群机器的计算能力，一般并行度设置为集群CPU总和的2-3倍。

操作步骤

并行度可以通过如下三种方式来设置，用户可以根据实际的内存、CPU、数据以及应用程序逻辑的情况调整并行度参数。

- 在会产生shuffle的操作函数内设置并行度参数，优先级最高。

  ```
  testRDD.groupByKey(24)
  ```

- 在代码中配置“spark.default.parallelism”设置并行度，优先级次之。

  ```
  val conf = new SparkConf()
  conf.set("spark.default.parallelism", 24)
  ```

- 在“$SPARK_HOME/conf/spark-defaults.conf”文件中配置“spark.default.parallelism”的值，优先级最低。

  ```
  spark.default.parallelism    24
  ```

12.9.1.4 使用广播变量

操作场景

Broadcast（广播）可以把数据集合分发到每一个节点上，Spark任务在执行过程中要使用这个数据集合时，就会在本地查找Broadcast过来的数据集合。如果不使用Broadcast，每次任务需要数据集合时，都会把数据序列化到任务里面，不但耗时，还使任务变得很大。

1. 每个任务分片在执行中都需要同一份数据集合时，就可以把公共数据集Broadcast到每个节点，让每个节点在本地都保存一份。
2. 大表和小表做join操作时可以把小表Broadcast到各个节点，从而就可以把join操作转变成普通的操作，减少了shuffle操作。

操作步骤

在开发应用程序时，添加如下代码，将“testArr”数据广播到各个节点。

```
def main(args: Array[String]) {
  ...
  val testArr: Array[Long] = new Array[Long](200)
  val testBroadcast: Broadcast[Array[Long]] = sc.broadcast(testArr)
  val resultRdd: RDD[Long] = inpputRdd.map(input => handleData(testBroadcast, input))
  ...
}

def handleData(broadcast: Broadcast[Array[Long]], input: String) {
  val value = broadcast.value
  ...
}
```

12.9.1.5 使用External Shuffle Service提升性能

操作场景

Spark系统在运行含shuffle过程的应用时，Executor进程除了运行task，还要负责写shuffle数据以及给其他Executor提供shuffle数据。当Executor进程任务过重，导致触发GC（Garbage Collection）而不能为其他Executor提供shuffle数据时，会影响任务运行。

External shuffle Service是长期存在于NodeManager进程中的一个辅助服务。通过该服务来抓取shuffle数据，减少了Executor的压力，在Executor GC的时候也不会影响其他Executor的任务运行。

操作步骤

1. 登录FusionInsight Manager系统。
2. 单击“服务管理 > Spark > 服务配置”。在“参数类别”中选择“全部配置”。
3. 选择“SparkResource > 默认”，修改以下参数：

   表12-23 参数列表

参数	默认值	修改结果
spark.shuffle.service.enabled	false	true

4. 重启Spark服务，配置生效。

> **说明：**
>
> 如果需要在Spark客户端用External Shuffle Service功能，需要重新下载并安装Spark客户端，具体操作请参见《安装客户端》章节。

12.9.1.6 Yarn模式下动态资源调度

操作场景

对于Spark应用来说，资源是影响Spark应用执行效率的一个重要因素。当一个长期运行的服务（比如JDBCServer），若分配给它多个Executor，可是却没有任何任务分配给它，而此时有其他的应用却资源紧张，这就造成了很大的资源浪费和资源不合理的调度。

动态资源调度就是为了解决这种场景，根据当前应用任务的负载情况，实时的增减Executor个数，从而实现动态分配资源，使整个Spark系统更加健康。

操作步骤

1. 需要先配置External shuffle service。
2. 登录FusionInsight Manager，将“spark.dynamicAllocation.enabled”参数的值设置为“true”，表示开启动态资源调度功能。默认情况下关闭此功能。
3. 下面是一些可选配置，如表12-24所示。

   表12-24 动态资源调度参数

配置项	说明	默认值
spark.dynamicAllocation.minExecutors	最小Executor个数。	0
spark.dynamicAllocation.initialExecutors	初始Executor个数。	spark.dynamicAllocation.minExecutors
spark.dynamicAllocation.maxExecutors	最大Executor个数。	2048
spark.dynamicAllocation.schedulerBacklogTimeout	调度第一次超时时间。	1(s)
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout	调度第二次及之后超时时间。	spark.dynamicAllocation.schedulerBacklogTimeout
spark.dynamicAllocation.executorIdleTimeout	普通Executor空闲超时时间。	60(s)
spark.dynamicAllocation.cachedExecutorIdleTimeout	含有cached blocks的Executor空闲超时时间。	Integer.MAX_VALUE

> **说明：**
>
> 使用动态资源调度功能，必须配置External Shuffle Service。

12.9.1.7 配置进程参数

操作场景

Spark on YARN模式下，有Driver、ApplicationMaster、Executor三种进程。在任务调度和运行的过程中，Driver和Executor承担了很大的责任，而ApplicationMaster主要负责container的启停。

因而Driver和Executor的参数配置对spark应用的执行有着很大的影响意义。用户可通过如下操作对Spark集群性能做优化。

操作步骤

1. 配置Driver内存。

   Driver负责任务的调度，和Executor、AM之间的消息通信。当任务数变多，任务平行度增大时，Driver内存都需要相应增大。

   您可以根据实际任务数量的多少，为Driver设置一个合适的内存。

   - 将“spark-defaults.conf”中的“spark.driver.memory”配置项或者“spark-env.sh”中的“SPARK_DRIVER_MEMORY”配置项设置为合适大小。

- 在使用spark-submit命令时，添加“--driver-memory MEM”参数设置内存。

2. 配置Executor个数。

   每个Executor每个核同时能跑一个task，所以增加了Executor的个数相当于增大了任务的并发度。在资源充足的情况下，可以相应增加Executor的个数，以提高运行效率。

   - 将“spark-defaults.conf”中的“spark.executor.instance”配置项或者“spark-env.sh”中的“SPARK_EXECUTOR_INSTANCES”配置项设置为合适大小。您还可以设置动态资源调度功能进行优化，详情请参见“Yarn模式下动态资源调度”章节。
   - 在使用spark-submit命令时，添加“--num-executors NUM”参数设置Executor个数。

3. 配置Executor核数。

   每个Executor多个核同时能跑多个task，相当于增大了任务的并发度。但是由于所有核共用Executor的内存，所以要在内存和核数之间做好平衡。

   - 将“spark-defaults.conf”中的“spark.executor.cores”配置项或者“spark-env.sh”中的“SPARK_EXECUTOR_CORES”配置项设置为合适大小。
   - 在使用spark-submit命令时，添加“--executor-cores NUM”参数设置核数。

4. 配置Executor内存。

   Executor的内存主要用于任务执行、通信等。当一个任务很大的时候，可能需要较多资源，因而内存也可以做相应的增加；当一个任务较小运行较快时，就可以增大并发度减少内存。

   - 将“spark-defaults.conf”中的“spark.executor.memory”配置项或者“spark-env.sh”中的“SPARK_EXECUTOR_MEMORY”配置项设置为合适大小。
   - 在使用spark-submit命令时，添加“--executor-memory MEM”参数设置内存。

示例

- 在执行spark wordcount计算中。1.6T数据，250个executor。

  在默认参数下执行失败，出现Futures timed out和OOM错误。

  因为数据量大，task数多，而wordcount每个task都比较小，完成速度快。当task数多时driver端相应的一些对象就变大了，而且每个task完成时executor和driver都要通信，这就会导致由于内存不足，进程之间通信断连等问题。

  当把Driver的内存设置到4g时，应用成功跑完。

- 使用JDBCServer执行TPC-DS测试套，默认参数配置下也报了很多错误：Executor Lost等。而当配置Driver内存为30g，executor核数为2，executor个数为125，executor内存为6g时，所有任务才执行成功。

12.9.1.8 设计DAG

操作场景

合理的设计程序结构，可以优化执行效率。在程序编写过程中要尽量减少shuffle操作，合并窄依赖操作。

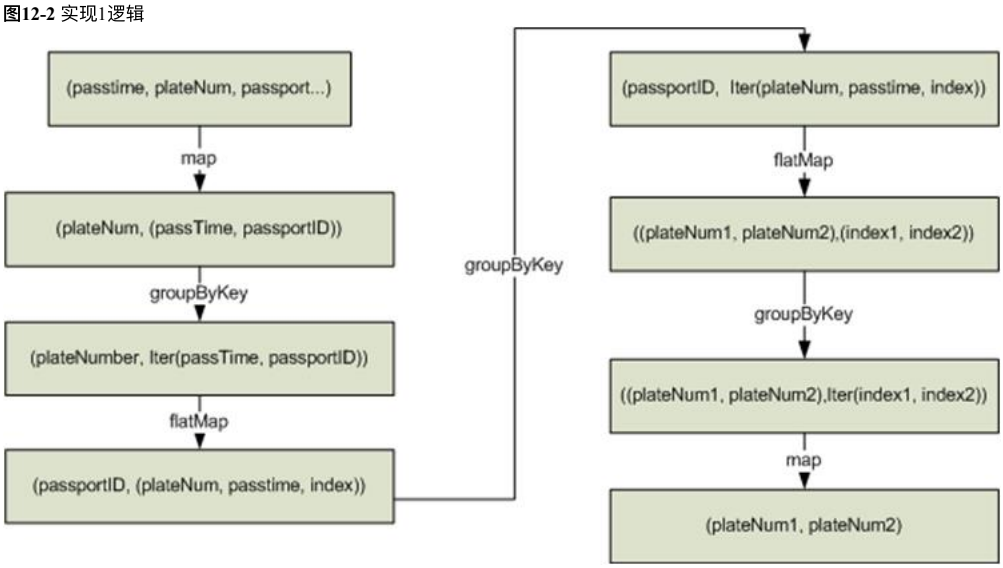
操作步骤

以“同行车判断”例子讲解DAG设计的思路。

- **数据格式：**通过收费站时间、车牌号、收费站编号.....
- **逻辑：**以下两种情况下判定这两辆车是同行车：
  - 如果两辆车都通过相同序列的收费站，
  - 通过同一收费站之间的时间差小于一个特定的值。

该例子有两种实现模式，其中实现1的逻辑如图12-2所示，实现2的逻辑如图12-3所示。

图12-2 实现1逻辑



实现1的逻辑说明：

1. 根据车牌号聚合该车通过的所有收费站并排序，处理后数据如下：

   车牌号1，[（通过时间，收费站3），（通过时间，收费站2），（通过时间，收费站4），（通过时间，收费站5）]

2. 标识该收费站是这辆车通过的第几个收费站。

（收费站3，（车牌号1，通过时间，通过的第1个收费站））

（收费站2，（车牌号1，通过时间，通过的第2个收费站））

（收费站4，（车牌号1，通过时间，通过的第3个收费站））

（收费站5，（车牌号1，通过时间，通过的第4个收费站））

3. 根据收费站聚合数据。

   收费站1，[（车牌号1，通过时间，通过的第1个收费站），（车牌号2，通过时间，通过的第5个收费站），（车牌号3，通过时间，通过的第2个收费站）]

4. 判断两辆车通过该收费站的时间差是否满足同行车的要求，如果满足则取出这两辆车。

   （车牌号1，车牌号2），（通过的第1个收费站，通过的第5个收费站）

   （车牌号1，车牌号3），（通过的第1个收费站，通过的第2个收费站）

5. 根据通过相同收费站的两辆车的车牌号聚合数据，如下：

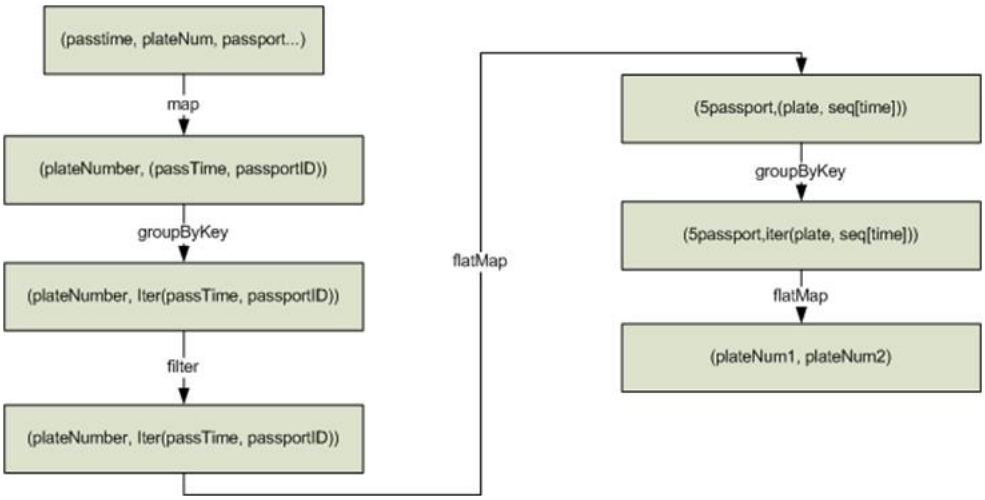
   （车牌号1，车牌号2），[（通过的第1个收费站，通过的第5个收费站），（通过的第2个收费站，通过的第6个收费站），（通过的第1个收费站，通过的第7个收费站），（通过的第3个收费站，通过的第8个收费站）]

6. 如果车牌号1和车牌号2通过相同收费站是顺序排列的（比如收费站3、4、5是车牌1通过的第1、2、3个收费站，是车牌2通过的第6、7、8个收费站）且数量大于同行车要求的数量则这两辆车是同行车。

实现1逻辑的缺点：

- 逻辑复杂
- 实现过程中shuffle操作过多，对性能影响较大。

图12-3 实现2逻辑



实现2的逻辑说明：

1. 根据车牌号聚合该车通过的所有收费站并排序，处理后数据如下：

   车牌号1，[（通过时间，收费站3），（通过时间，收费站2），（通过时间，收费站4），（通过时间，收费站5）]

2. 根据同行车要通过的收费站数量（例子里为3）分段该车通过的收费站序列，如上面的数据被分解成：

   收费站3->收费站2->收费站4，（车牌号1，[收费站3时间，收费站2时间，收费站4时间]）

   收费站2->收费站4->收费站5，（车牌号1，[收费站2时间，收费站4时间，收费站5时间]）

3. 把通过相同收费站序列的车辆聚合，如下：

   收费站3->收费站2->收费站4，[（车牌号1，[收费站3时间，收费站2时间，收费站4时间]），（车牌号2，[收费站3时间，收费站2时间，收费站4时间]），（车牌号3，[收费站3时间，收费站2时间，收费站4时间]）]

4. 判断通过相同序列收费站的车辆通过相同收费站的时间差是不是满足同行车的要求，如果满足则说明是同行车。

实现2的优点如下：

- 简化了实现逻辑。
- 减少了一个groupByKey，也就减少了一次shuffle操作，提升了性能。

12.9.1.9 经验总结

使用mapPartitions，按每个分区计算结果

如果每条记录的开销太大，例：

```
rdd.map{x=>conn=getDBConn;conn.write(x.toString);conn.close}
```

则可以使用MapPartitions，按每个分区计算结果，如

```
rdd.mapPartitions(records => conn.getDBConn;for(item <- records)
write(item.toString); conn.close)
```

使用mapPartitions可以更灵活地操作数据，例如对一个很大的数据求TopN，当N不是很大时，可以先使用mapPartitions对每个partition求TopN，collect结果到本地之后再做排序取TopN。这样相比直接对全量数据做排序取TopN效率要高很多。

使用coalesce调整分片的数量

coalesce可以调整分片的数量。coalesce函数有两个参数：

coalesce(numPartitions: Int, shuffle: Boolean = false)

当shuffle为true的时候，函数作用与repartition(numPartitions: Int)相同，会将数据通过Shuffle的方式重新分区；当shuffle为false的时候，则只是简单的将父RDD的多个partition合并到同一个task进行计算，shuffle为false时，如果numPartitions大于父RDD的切片数，那么分区不会重新调整。

遇到下列场景，可选择使用coalesce算子：

- 当之前的操作有很多filter时，使用coalesce减少空运行的任务数量。此时使用coalesce(numPartitions, false)，numPartitions小于父RDD切片数。
- 当输入切片个数太大，导致程序无法正常运行时使用。
- 当任务数过大时候Shuffle压力太大导致程序挂住不动，或者出现linux资源受限的问题。此时需要对数据重新进行分区，使用coalesce(numPartitions, true)。

localDir配置

Spark的Shuffle过程需要写本地磁盘，Shuffle是Spark性能的瓶颈，I/O是Shuffle的瓶颈。配置多个磁盘则可以并行的把数据写入磁盘。如果节点中挂载多个磁盘，则在每个磁盘配置一个Spark的localDir，这将有效分散Shuffle文件的存放，提高磁盘I/O的效率。如果只有一个磁盘，配置了多个目录，性能提升效果不明显。

Collect小数据

大数据量不适用collect操作。

collect操作会将Executor的数据发送到Driver端，因此使用collect前需要确保Driver端内存足够，以免Driver进程发生OutOfMemory异常。当不确定数据量大小时，可使用saveAsTextFile等操作把数据写入HDFS中。只有在能够大致确定数据大小且driver内存充足的时候，才能使用collect。

使用reduceByKey

reduceByKey会在Map端做本地聚合，使得Shuffle过程更加平缓，而groupByKey等Shuffle操作不会在Map端做聚合。因此能使用reduceByKey的地方尽量使用该算子，避免出现groupByKey().map(x=>(x._1,x._2.size))这类实现方式。

广播map代替数组

当每条记录需要查表，如果是Driver端用广播方式传递的数据，数据结构优先采用set/map而不是Iterator，因为Set/Map的查询速率接近O(1)，而Iterator是O(n)。

数据倾斜

当数据发生倾斜（某一部分数据量特别大），虽然没有GC（Gabage Collection，垃圾回收），但是task执行时间严重不一致。

- 需要重新设计key，以更小粒度的key使得task大小合理化。
- 修改并行度。

优化数据结构

- 把数据按列存放，读取数据时就可以只扫描需要的列。
- 使用Hash Shuffle时，通过设置spark.shuffle.consolidateFiles为true，来合并shuffle中间文件，减少shuffle文件的数量，减少文件IO操作以提升性能。最终文件数为reduce tasks数目。

12.9.1.10 优化返回查询结果返回大量数据的场景

操作场景

使用命令提交任务时，如果任务中包含了查询结果返回大量数据的操作，由于此类操作会在driver端占用大量内存进行数据存放，则存在导致内存溢出的风险，因此需要进行优化来支持该场景。

操作步骤

在客户端的“spark-defaults.conf”配置文件中调整如下参数。

表12-25 参数说明

参数	描述	默认值
spark.sql.bigdata.thriftServer.useHdfsCollect	driver端读取数据时的方式，当配置为false时，读取的数据会存放在内存中，适合数据量不大（数据量小于Driver端设置的内存大小）的情况。 如果返回的数据量大（数据量大于Driver端设置的内存大小），需要把此值配置成true，由于先保存成HDFS文件，再分批读取，性能比配置成false慢。	false

12.9.2 SQL和DataFrame调优

12.9.2.1 Spark SQL join优化

操作场景

Spark SQL中，当对两个表进行join操作时，利用Broadcast特性（见“使用广播变量”章节），将小表BroadCast到各个节点上，从而转变成非shuffle操作，提高任务执行性能。

说明：

这里join操作，只指inner join。

操作步骤

在Spark SQL中进行Join操作时，可以按照以下步骤进行优化。为了方便说明，设表A和表B，且A、B表都有个名为name的列。对A、B表进行join操作。

1. 估计表的大小。

根据每次加载数据的大小，来估计表大小。

也可以在Hive的数据库存储路径下直接查看表的大小。首先在Spark的配置文件hive-site.xml中，查看Hive的数据库路径的配置，默认为“/user/hive/warehouse”。Spark服务多实例默认数据库路径为“/user/*hive*/warehouse”，例如“/user/hive1/warehouse”。

```
<property>
   <name>hive.metastore.warehouse.dir</name>
  <value>${test.warehouse.dir}</value>
  <description></description>
</property>
```

然后通过hadoop命令查看对应表的大小。如查看表A的大小命令为：

```
hadoop fs –du –s –h ${test.warehouse.dir}/a
```

说明：

进行广播操作，需要至少有一个表不是空表。

2. 配置自动广播的阈值。

   Spark中，判断表是否广播的阈值为67108864（即64M）。如果两个表的大小至少有一个小于64M时，可以跳过该步骤。

   自动广播阈值的配置参数介绍，见表12-26。

   表12-26 参数介绍

参数	默认值	描述
spark.sql.autoBroadcastJoinThreshold	67108864	当进行join操作时，配置广播的最大值；当表的字节数小于该值时便进行广播。 当配置为-1时，将不进行广播。 参见https://spark.apache.org/docs/latest/sql-programming-guide.html

   配置自动广播阈值的方法：

   - 在Spark的配置文件“spark-defaults.conf”中，设置“spark.sql.autoBroadcastJoinThreshold”的值。其中，<size>根据场景而定，但要求该值至少比其中一个表大。

     spark.sql.autoBroadcastJoinThreshold = <size>

   - 利用Hive CLI命令，设置阈值。在运行Join操作时，提前运行下面语句

     SET spark.sql.autoBroadcastJoinThreshold=<size>

     其中，<size>根据场景而定，但要求该值至少比其中一个表大。

3. （可选）如下两种场景，需要执行Analyze命令（***ANALYZE TABLE tableName COMPUTE STATISTICS noscan;***）更新表元数据后进行广播。

   - 需要广播的表是分区表，新建表且文件类型为非Parquet文件类型。
   - 需要广播的表是分区表，更新表数据后。

4. 进行join操作。

   这时join的两个table，至少有个表是小于阈值的。

   如果A表和B表都小于阈值，且A表的字节数小于B表时，则运行B join A，如

   ```
   SELECT A.name FROM B JOIN A ON A.name = B.name;
   ```

   否则运行A join B。

   ```
   SELECT A.name FROM A JOIN B ON A.name = B.name;
   ```

5. 使用Executor广播减少Driver内存压力。

   默认的BroadCastJoin会将小表的内容，全部收集到Driver中，因此需要适当的调大Driver的内存。内存增加的计算公式为：“spark.sql.autoBroadcastJoinThreshold * the number of broadcast table * 2”。当广播任务比较频繁的时候，Driver有可能因为OOM而异常退出。

   此时，可以开启Executor广播，在客户端“spark-defaults.conf”文件中配置Executor广播参数“spark.sql.bigdata.useExecutorBroadcast”为“true”，减少Driver内存压力。

   表12-27 参数介绍

参数	描述	默认值
spark.sql.bigdata.useExecutorBroadcast	设置为true时，使用Executor广播，将表数据缓存在Executor中，而不是放在Driver之中，减少Spark Driver内存的压力。	true

参考信息

小表执行超时，导致任务结束。

默认情况下，BroadCastJoin只允许小表计算5分钟，超过5分钟该任务会出现超时异常，而这个时候小表的broadcast任务依然在执行，造成资源浪费。

这种情况下，有两种方式处理：

- 调整“spark.sql.broadcastTimeout”的数值，加大超时的时间限制。
- 降低“spark.sql.autoBroadcastJoinThreshold”的数值，不使用BroadCastJoin的优化。

12.9.2.2 优化数据倾斜场景下的Spark SQL性能

配置场景

在Spark SQL多表Join的场景下，会存在关联键严重倾斜的情况，导致Hash分桶后，部分桶中的数据远高于其它分桶。最终导致部分Task过重，跑得很慢；其它Task过轻，跑得很快。一方面，数据量大Task运行慢，使得计算性能低；另一方面，数据量少的Task在运行完成后，导致很多CPU空闲，造成CPU资源浪费。

通过如下配置项可将部分数据采用Broadcast方式分发，以便均衡Task，提高CPU资源的利用率，从而提高性能。

说明：

未产生倾斜的数据，将采用原有方式进行分桶并运行。

使用约束：

1. 两表中倾斜的关联键不同。（例如：A join B on A.name = B.name，如果A中name = "zhangsan"倾斜，那么B中name = "zhangsan"不能倾斜，其他的如name = "lisi"可以倾斜。）
2. 只支持两表间的Join。
3. TS（TableScan）到Join之间的操作只支持TS（TableScan）、FIL（Filter）、SEL（Select）。

配置描述

在客户端的“spark-defaults.conf”配置文件中调整如下参数。

表12-28 参数说明

参数	描述	默认值
spark.sql.planner.skewJoin	设置是否开启数据倾斜优化。“true”表示开启。开启后，会基于“spark.sql.planner.skewJoin.threshold”参数识别出倾斜关键，系统会将这部分数据采用Broadcast方式，可以避免数据倾斜，提升CPU利用率，从而提升性能。	false
spark.sql.planner.skewJoin.threshold	用于判断是否存在数据倾斜的阈值。当存在关联键的个数大于该阈值，则存在数据倾斜，该关联键为倾斜关联键。	100000

12.9.2.3 优化小文件场景下的Spark SQL性能

配置场景

Spark SQL的表中，经常会存在很多小文件（大小远小于HDFS块大小），每个小文件默认对应Spark中的一个Partition，也就是一个Task。在很多小文件场景下，Spark会起很多Task。当SQL逻辑中存在Shuffle操作时，会大大增加hash分桶数，严重影响性能。

在小文件场景下，您可以通过如下配置手动指定每个Task的数据量（Split Size），确保不会产生过多的Task，提高性能。

说明：

当SQL逻辑中不包含Shuffle操作时，设置此配置项，不会有明显的性能提升。

配置描述

在客户端的“spark-defaults.conf”配置文件中调整如下参数。

表12-29 参数说明

参数	描述	默认值
spark.sql.small.file.combine	用于设置是否开启小文件优化。“true”表示开启。开启后，可以避免过多的小Task。	false
spark.sql.small.file.split.size	合并小文件后，用于指定单个Task期望的数据量。 单位：Byte	256000000

12.9.2.4 INSERT...SELECT操作调优

操作场景

在以下几种情况下，执行INSERT...SELECT操作可以进行一定的调优操作。

- 查询的数据是大量的小文件。
- 查询的数据是较多的大文件。
- 在beeline/JDBCServer模式下使用非spark用户操作。

操作步骤

可对INSERT...SELECT操作做如下的调优操作。

- 如果建的是Hive表，将存储类型设为Parquet，从而减少执行INSERT...SELECT语句的时间。
- 建议使用spark-sql或者在beeline/JDBCServer模式下使用spark用户来执行INSERT...SELECT操作，避免执行更改文件owner的操作，从而减少执行INSERT...SELECT语句的时间。

  说明：

  在beeline/JDBCServer模式下，executor的用户跟driver是一致的，driver是JDBCServer服务的一部分，是由spark用户启动的，因此其用户也是spark用户，且当前无法实现在运行时将beeline端的用户透传到executor，因此使用非spark用户时需要对文件进行更改owner为beeline端的用户，即实际用户。

- 如果查询的数据是大量的小文件将会产生大量map操作，从而导致输出存在大量的小文件，在执行重命名文件操作时将会耗费较多时间，此时可以通过设置spark.sql.small.file.combine = true来开启小文件合并功能来减少输出文件数，减少执行重命名文件操作的时间，从而减少执行INSERT...SELECT语句的时间。

- 如果查询的数据是较多的大文件，那么执行该操作的输出文件也会很多，当这个量级达到数十万时，此时可以通过设置spark.sql.small.file.combine = true来开启小文件合并功能，同时设置spark.sql.small.file.split.size为一个较合理的值，控制输出文件大小，减小输出文件个数（原则是确保reduce任务能够充分利用集群资源，否则会增加写文件的时间），减少执行重命名文件的时间，从而减少执行INSERT...SELECT语句的时间。

说明：

上述优化操作并不能解决全部的性能问题，对于以下两种场景仍然需要较多时间：

- 对于动态分区表，如果其分区数非常多，那么也需要执行较长的时间。
- 如果查询的数据为大量的大文件，那么即使开启小文件合并功能，其输出文件也依旧很多，那么也需要较长的时间。

12.9.2.5 多并发JDBC客户端连接JDBCServer

操作场景

JDBCServer支持多用户多并发接入，但当并发任务数量较高的时候，默认的JDBCServer配置将无法支持，因此需要进行优化来支持该场景。

操作步骤

1. 设置JDBCServer的公平调度策略。

   Spark默认使用FIFO（First In First Out）的调度策略，但对于多并发的场景，使用FIFO策略容易导致短任务执行失败。因此在多并发的场景下，需要使用公平调度策略，防止任务执行失败。

   a. 在Spark中设置公平调度，具体请参考http://spark.apache.org/docs/latest/job-scheduling.html#scheduling-within-an-application

   b. 在JDBC客户端中设置公平调度。

      i. 在BeeLine命令行客户端或者JDBC自定义代码中，执行以下语句，

      其中PoolName是公平调度的某一个调度池。

      SET spark.sql.thriftserver.scheduler.pool=PoolName;

      ii. 执行相应的SQL命令，Spark任务将会在上面的调度池中运行。

2. 设置BroadCastHashJoin的线程池个数。

   BroadCastHashJoin使用多线程方式广播表，在多并发场景下，会有多个表同时在多线程中，一旦广播表的个数大于线程池个数，任务会出错，因此需要在JDBCServer的“spark-defaults.conf”配置文件中或在命令行中执行***set spark.sql.broadcastHashJoin.maxThreadNum=****value*，调整线程池个数。

   表12-30 参数描述

参数	描述	默认值
spark.sql.broadcastHashJoin.maxThreadNum	用于BroadcastHashJoin的最大的线程池个数，同一时间被广播的表的个数应该小于该参数值。	128

3. 设置BroadCastHashJoin的超时时间。

   BroadCastHashJoin有超时参数，一旦超过预设的时间，该查询任务直接失败，在多并发场景下，由于计算任务抢占资源，可能会导致BroadCastHashJoin的Spark任务无法执行，导致超时出现。因此需要在JDBCServer的“spark-defaults.conf”配置文件中调整超时时间。

   表12-31 参数描述

参数	描述	默认值
spark.sql.broadcastTimeout	BroadcastHashJoin中广播表的超时时间，当任务并发数较高的时候，可以调高该参数值，或者直接配置为负数，负数为无穷大的超时时间。	300（数值类型，实际为五分钟）

4. 设置串行BroadcastHashJoin。

   当并发任务非常重（例如全部完成时间超过2个小时），需要将BroadcastHashJoin设置为串行化，这样就能去除超时时间对并发任务的影响。但是串行化相对于并行化，会降低集群资源的使用率，因此在轻量级任务并发时，不要开启该配置项。

   表12-32 参数描述

参数	描述	默认值
spark.sql.bigdata.useSerialBroadcastHashJoin	是否使用串行方式执行BroadcastHashJoin。串行化BroadcastHashJoin会降低集群资源使用率，但对于高并发的重任务，可以解决超时的困扰。	false

12.9.2.6 Parquet元数据buildScan优化

操作场景

在分区表的场景下，会对每个分区串行执行buildScan操作来构造RDD，在构造RDD时会随着分区数的增加而增加执行时间。因此，提供并行执行buildScan操作来构造RDD，从而提升执行效率。

该优化主要是Driver利用多线程并行执行buildScan操作来提升性能，因此不适合多session场景，因为在多session场景下有可能造成Driver运行过多的线程，从而造成未知错误。

操作步骤

在客户端的“spark-defaults.conf”配置文件中调整如下参数。

表12-33 参数描述

参数	描述	默认值
spark.sql.sources.parallelBuildScan.threshold	并行执行buildScan操作的分区数阈值。	–1
spark.sql.sources.parallelBuildScan.threadNum	并行执行buildScan操作的线程数。	2

12.9.2.7 ParquetRelation InputSplits优化

操作场景

当前读取ParquetRelation类型的数据时每次都会执行getSplits操作，如果要读取的文件较多，则耗时较长。因此在第一次构造Relation时读取全部InputSplits信息并缓存，后续只要缓存没被清除，则每次只需从缓存中读取所需的InputSplits信息，从而提升非第一次查询的性能。

操作步骤

在客户端的“spark-defaults.conf”配置文件调整如下参数。

表12-34 参数描述

参数	描述	默认值
spark.sql.source.inputSplit.useCache	是否开启缓存ParquetRelation的InputSplits信息功能。	false

说明：

- 只有在数据文件数在区间(1000, 100000]之内，且spark.sql.source.inputSplit.useCache=true时才会缓存ParquetRelation的InputSplits信息。
- 缓存ParquetRelation的InputSplits信息功能只会在第一次构造Relation时（例如第一次执行查询操作时）生效。如果在第一次构造Relation之后才开启缓存ParquetRelation的InputSplits信息功能，那么可通过执行REFRESH操作来清除对应的Relation缓存。
- 一旦执行过数据导入，必须执行REFRESH操作来刷新缓存，否则有可能造成执行查询操作时上报文件不存在异常。

12.9.2.8 Limit优化

操作场景

在RDD的Partition数过多时（大于或等于1000，例如spark.sql.shuffle.partitions=2000时），建议使用Limit优化，减少扫描的文件数。

在物理计划中，Limit分为非末端Limit和末端Limit。非末端Limit优化可以通过“spark.sql.optimize.limit”配置项进行开启或关闭。末端Limit优化没有控制开关，默认都进行优化。用户可以通过在Spark SQL客户端中执行***explain <SQL statement>***命令查看物理计划。

举例说明：

如下所示，使用***explain***命令查询物理计划，在查询结果的中间位置显示的就是非末端Limit，即下面蓝色字体标识的部分。

```
spark-sql> explain select count(*) from (select key, sum(value) from src group by key limit 10) t;
== Physical Plan ==
TungstenAggregate(key=[], functions=[(count(1),mode=Final,isDistinct=false)], output=[_c0#24L])
TungstenAggregate(key=[], functions=[(count(1),mode=Partial,isDistinct=false)], output=[currentCount#27L])
Limit 10
ConvertToSafe
TungstenAggregate(key=[key#21], functions=[], output=[])
TungstenExchange hashpartitioning(key#21)
TungstenAggregate(key=[key#21], functions=[], output=[key#21])
HiveTableScan [key#21], (MetastoreRelation default, src, None), Statistics(5812)
Time taken: 0.119 seconds, Fetched 9 row(s)
```

如下所示，使用***explain***命令查询物理计划，在查询结果的最前面显示的就是末端Limit，即下面蓝色字体标识的部分。

```
spark-sql> explain select key, sum(value) from src group by key limit 10;
== Physical Plan ==
Limit 10
ConvertToSafe
TungstenAggregate(key=[key#33], functions=[(sum(cast(value#34 as double)),mode=Final,isDistinct=false)], output=[key#33,_c1#35])
TungstenExchange hashpartitioning(key#33)
TungstenAggregate(key=[key#33], functions=[(sum(cast(value#34 as double)),mode=Partial,isDistinct=false)], output=[key#33,currentSum#38])
HiveTableScan [key#33,value#34], (MetastoreRelation default, src, None), Statistics(5812)
```

操作步骤

在客户端的“spark-defaults.conf”配置文件中调整如下参数。

表12-35 参数描述

参数	描述	默认值
spark.sql.optimize.limit	非末端Limit优化开关。开启本开关，且Limit个数小于等于1000时，优化生效，使非末端Limit也使用末端Limit的分批读取的算法。建议开启。	false
spark.sql.limit.numPartsToTry.first	在Limit场景下第一次读取的Partition数。在RDD的Partition数过多时，建议开启。建议配置区间为[1, min(limit value, 10)]。	–1
spark.sql.limit.numPartsToTry.other	在Limit场景下后续读取的Partition数。在RDD的Partition数过多时，建议开启。建议配置区间为[50, 100]。生效条件是spark.sql.limit.numPartsToTry.first>0，若不配置则默认使用spark.sql.limit.numPartsToTry.first的参数值。	–1

12.9.2.9 LimitScan优化

操作场景

用户场景中，存在快速展示数据的场景，典型的SQL语句如下所示：

select col1, col2 from table limit 100;

通过执行包含Limit子句的Spark SQL语句，来查询表的部分数据，并快速展示，但是在查询parquet大表时，由于获取大表分区信息耗时较长，导致无法达到实时查询的目的。此时用户可以开启LimitScan优化，提高查询性能，快速展示查询结果。

操作步骤

在客户端的“spark-defaults.conf”配置文件中调整如下参数。

表12-36 参数描述

参数	描述	默认值
spark.sql.limitScan.enabled	是否开启LimitScan优化。	true
spark.sql.limitScan.num	Limit个数限制，超过这个数值，则不做LimitScan优化。	5000

使用约束

- 只适用于Hive命令建的表，不能用于DataSource表。
- Select的表字段类型只适用于原生数据类型（Array，Struct，Map），则不做LimitScan优化。
- Select的字段必须为表中真实字段，或者该字段的别名，例如“select col1, col2 as alias”。
- Select字段中如果有带算术运算、函数的字段，则不支持LimitScan优化，例如“select col1*100+100, avg(col2) from table1 limit 100”。
- 不适用于包含WHERE、HAVING、GROUP BY、CLUSTER BY、DISTRIBUTE BY、SORT BY、ORDER BY子句的Select语句。
- 若Spark服务端的“hive-site.xml”文件中“hive.server2.enable.doAs”配置成“true”，则无法使用LimitScan功能。

12.9.2.10 预先Broadcast小表优化

操作场景

开启预先Broadcast小表优化后，同一SQL语句或不同SQL语句内，存在相同的小表或对相同小表做子查询时，只需要将小表广播一次，后续就可以复用缓存在内存中的表数据，避免重复广播，从而提升SQL的性能。

支持预广播的小表是指小于自动广播阈值的表，用户可以通过Spark的配置文件“spark-defaults.conf”中的“spark.sql.autoBroadcastJoinThreshold”参数指定自动广播阈值。

操作步骤

1. 通过如下配置项开启预广播小表的功能。

   在客户端的“spark-defaults.conf”配置文件中调整如下参数。

   表12-37 参数描述

参数	描述	默认值
spark.sql.saveBroadcastTables.enabled	是否开启预先广播小表的优化功能。	true

   在服务端的“spark-defaults.conf”配置文件中调整如下参数。重启服务端后才生效。

   表12-38 参数描述

参数	描述	默认值
spark.sql.broadcastTables.sizeInBytes	Driver中缓存的表大小（表在HDFS上的大小，而不是表在内存中缓存的实际内容的大小）的阈值，超过这个阈值则采用FIFO的方式挤出最先缓存的表。单位为Byte。 **说明：** 这是服务端参数，不能在客户端通过***SET***的方式进行设置。	104857600

2. （可选）相关命令介绍。
   - 如果用户更新分区小表，更新后需要手动执行***Analyze***命令（Text表）或***Refresh***命令（Parquet表或Orc表）来更新元数据，从而重新缓存更新后的分区小表。
     - Analyze命令

       ANALYZE TABLE tableName COMPUTE STATISTICS noscan;
     - Refresh命令

       Refresh table tableName;
   - 显示所有缓存的广播表。

     SHOW BROADCAST TABLES
   - 清除所有缓存的广播表。

     CLEAR BROADCAST

只有授予“ADMIN”权限的用户才能执行该命令。

执行该命令后，再次运行SQL语句时，会再触发广播并将广播表缓存起来。

使用限制

- 不支持Cache在内存中的小表。
- 不支持手动修改表的totalSize和ModifyTime等元数据信息。
- 支持预广播的小表格式为Text，Parquet和Orc。
- 支持的Join类型有：Inner Join、Left Join、Right Join、Full Join。
- spark-sql中缓存的广播表只支持在单个CLI内共享（spark-sql重启后需要重新广播，触发缓存）。

12.9.3 Spark Streaming调优

操作场景

Spark Streaming作为一种mini-batch方式的流式处理框架，它主要的特点是：秒级时延和高吞吐量。因此Spark Streaming调优的目标：在秒级延迟的情景下，提高Spark Streaming的吞吐能力，在单位时间处理尽可能多的数据。

说明：

本章节适用于输入数据源为Kafka的使用场景。

操作步骤

一个简单的流处理系统由以下三部分组件组成：数据源 + 接收器 + 处理器。数据源为Kafka，接受器为Spark Streaming中的Kafka数据源接收器，处理器为Spark Streaming。

对Spark Streaming调优，就必须使该三个部件的性能都最优化。

- **数据源调优**

  在实际的应用场景中，数据源为了保证数据的容错性，会将数据保存在本地磁盘中，而Spark Streaming的计算结果往往全部在内存中完成，数据源很有可能成为流式系统的最大瓶颈点。

  对Kafka的性能调优，有以下几个点：

  - 使用Kafka-0.8.2以后版本，可以使用异步模式的新Producer接口。
  - 配置多个Broker的目录，设置多个IO线程，配置Topic合理的Partition个数。

  详情请参见Kafka开源文档中的“性能调优”部分：http://kafka.apache.org/documentation.html

- **接收器调优**

  Spark Streaming中已有多种数据源的接收器，例如Kafka、Flume、MQTT、ZeroMQ等，其中Kafka的接收器类型最多，也是最成熟一套接收器。

  Kafka包括三种模式的接收器API：

  - KafkaReceiver：直接接收Kafka数据，进程异常后，可能出现数据丢失。
  - ReliableKafkaReceiver：通过ZooKeeper记录接收数据位移。
  - DirectKafka：直接通过RDD读取Kafka每个Partition中的数据，数据高可靠。

  从实现上来看，DirectKafka的性能会是最好的，实际测试上来看，DirectKafka也确实比其他两个API性能好了不少。因此推荐使用DirectKafka的API实现接收器。

  数据接收器作为一个Kafka的消费者，对于它的配置优化，请参见Kafka开源文档：http://kafka.apache.org/documentation.html

- **处理器调优**

  Spark Streaming的底层由Spark执行，因此大部分对于Spark的调优措施，都可以应用在Spark Streaming之中，例如：

  - 数据序列化
  - 配置内存
  - 设置并行度
  - 使用External Shuffle Service提升性能

  说明：

  在做Spark Streaming的性能优化时需注意一点，越追求性能上的优化，Spark Streaming整体的可靠性会越差。例如：

  “spark.streaming.receiver.writeAheadLog.enable”配置为“false”的时候，会明显减少磁盘的操作，提高性能，但由于缺少WAL机制，会出现异常恢复时，数据丢失。

  因此，在调优Spark Streaming的时候，这些保证数据可靠性的配置项，在生产环境中是不能关闭的。

12.9.4 Spark CBO调优

操作场景

SQL语句转化为具体执行计划是由SQL查询编译器决定的，同一个SQL语句可以转化成多种物理执行计划，如何指导编译器选择效率最高的执行计划，这就是优化器的主要作用。传统数据库（例如Oracle）的优化器有两种：基于规则的优化器(Rule-Based Optimization，RBO)和基于代价的优化器(Cost-Based Optimization，CBO)。

- RBO

  RBO使用的规则是根据经验形成的，只要按照这个规则去写SQL语句，无论数据表中的内容怎样、数据分布如何，都不会影响到执行计划。

- CBO

  CBO是根据实际数据分布和组织情况，评估每个计划的执行代价，从而选择代价最小的执行计划。

目前Spark的优化器都是基于RBO的，已经有数十条优化规则，例如谓词下推、常量折叠、投影裁剪等，这些规则是有效的，但是它对数据是不敏感的。导致的一个问题就是数据表中数据分布发生变化时，RBO是不感知的，基于RBO生成的执行计划不能确保是最优的。而CBO的重要作用就是能够根据实际数据分布估算出SQL语句，生成一组可能被使用的执行计划中代价最小的执行计划，从而提升性能。

目前CBO主要的优化点是Join算法选择。举个简单例子，当两个表做Join操作，如果其中一张原本很大的表经过Filter操作之后结果集小于BroadCast的阈值，在没有CBO情况下是无法感知大表过滤后变小的情况，采用的是SortMergeJoin算法，涉及到大量Shuffle操作，很耗费性能；在有CBO的情况下是可以感知到结果集的变化，采用的是BroadcastHashJoin算法，会将过滤后的小表BroadCast到每个节点，转变为非Shuffle操作，从而大大提高性能。

操作步骤

Spark CBO的设计思路是，基于表和列的统计信息，对各个操作算子（Operator）产生的中间结果集大小进行估算，最后根据估算的结果来选择最优的执行计划。

1. 设置配置项。
   - 在“spark-defaults.conf”配置文件中增加配置项“spark.sql.cbo”，将其设置为true，默认为false。
   - 在客户端执行SQL语句***set spark.sql.cbo=true***进行配置。
2. 执行统计信息生成命令，得到统计信息。

   说明：

   此步骤只需在运行所有SQL前执行一次。如果数据集发生了变化（插入、更新或删除），为保证CBO的优化效果，需要对有变化的表或者列再次执行统计信息生成命令重新生成统计信息，以得到最新的数据分布情况。

   - 表：执行***COMPUTE STATS FOR TABLE src***命令计算表的统计信息，统计信息包括记录条数、文件数和物理存储总大小。
   - 列：
     - 执行***COMPUTE STATS FOR TABLE src ON COLUMNS***命令计算所有列的统计信息。
     - 执行***COMPUTE STATS FOR TABLE src ON COLUMNS*** *name,age*命令计算表中name和age两个字段的统计信息。

       当前列的统计信息支持四种类型：数值类型、日期类型、时间类型和字符串类型。对于数值类型、日期类型和时间类型，统计信息包括：Max、Min、不同值个数(Number of Distinct Value，NDV)、空值个数(Number of Null)和Histogram（支持等宽、等高直方图）；对于字符串类型，统计信息包括：Max、Min、Max Length、Average Length、不同值个数(Number of Distinct Value，NDV)、空值个数(Number of Null)和Histogram（支持等宽直方图）。
3. CBO调优
   - 自动优化：用户根据自己的业务场景，输入SQL语句查询，程序会自动去判断输入的SQL语句是否符合优化的场景，从而自动选择Join优化算法。
   - 手动优化：用户可以通过***DESC FORMATTED src***命令查看统计信息，根据统计信息的分布，人工优化SQL语句。

12.9.5 Carbon性能调优

查询性能调优

Carbon可以通过调整各种参数来提高查询性能。大部分参数聚焦于增加并行性处理和更好地使用系统资源。

- 字典（Dictionary）缓存：Carbon采用字典编码，以提升查询性能和数据文件压缩率。在一个表格中，Carbon会对每个字典编码列创建字典文件。这些字典文件通过解码查询执行结果，由查询处理器加载到内存中。一旦完成加载，这些字典文件会被保存在内存中，避免被再次从磁盘读取，以加速查询执行的速度。但是，如果表的数量较多，则不能将所有列字典文件都保存至物理内存中。因此，Carbon对保存在其内存缓存中的列字典文件的数目有限制。该限制值可通过下列属性参数进行配置。设置一个较大的值，可在内存中缓存更多列字典文件数目，进而提升查询性能。单位为MB，默认值为0。例如，可配置为“carbon.max.level.cache.size=[10]”。
- Spark Executor数量：Executor是Spark并行性的基础实体。通过增加Executor数量，集群中的并行数量也会增加。关于如何配置Executor数量，请参考Spark资料。
- Executor核：每个Executor内，并行任务数受Executor核的配置控制。通过增加Executor核数，可增加并行任务数，从而提高性能。关于如何配置Executor核数，请参考Spark资料。
- HDFS block容量：Carbon通过给不同的处理器分配不同的block来分配查询任务。所以一个HDFS block是一个分区单元。另外，Carbon在Spark驱动器中，支持全局block级索引，这有助于减少需要被扫描的查询block的数量。设置较大的block容量，可提高I/O效率，但是会降低全局索引效率；设置较小的block容量，意味着更多的block数量，会降低I/O效率，但是会提高全局索引效率，同时，对于索引查询会要求更多的内存。
- 扫描线程数量：扫描仪（Scanner）线程控制每个任务中并行处理的数据块的数量。通过增加扫描仪线程数，可增加并行处理的数据块的数量，从而提高性能。可使用“carbon.properties”文件中的“carbon.number.of.cores”属性来配置扫描仪线程数。例如，“carbon.number.of.cores = 4”。

Carbon查询流程

当Carbon首次收到对某个表（例如表A）的查询任务时，系统会加载表A的索引数据到内存中，执行查询流程。当Carbon再次收到对表A的查询任务时，系统则不需要再加载其索引数据。

在Carbon中执行查询时，查询任务会被分成几个扫描任务。即，基于Carbon数据存储的HDFS block对扫描任务进行分割。扫描任务由集群中的执行器执行。扫描任务可以并行、部分并行，或顺序处理，具体采用的方式取决于执行器的数量以及配置的执行器核数。

查询任务的某些部分可在独立的任务级上处理，例如select和filter。查询任务的某些部分可在独立的任务级上进行部分处理，例如group-by、count、distinct count等。

某些操作无法在任务级上处理，例如Having Clause（分组后的过滤），sort等。这些无法在任务级上处理，或只能在任务级上部分处理的操作需要在集群内跨执行器来传输数据（部分结果）。这个传送操作被称为shuffle。

任务数量越多，需要shuffle的数据就越多，会对查询性能产生不利影响。

由于任务数量取决于HDFS block的数量，而HDFS block的数量取决于每个block的大小，因此合理选择HDFS block的大小很重要，需要在提高并行性，进行shuffle操作的数据量和聚合表的大小之间达到平衡。

分割和Executors的关系

如果分割数≤Executor数xExecutor核数，那么任务将以并行方式运行。否则，某些任务只有在其他任务完成之后才能开始。因此，要确保Executor数xExecutor核数≥分割数。同时，还要确保有足够的分割数，这样一个查询任务可被分为足够多的子任务，从而确保并行性。

配置扫描仪线程

扫描仪线程属性决定了每个分割的数据被划分的可并行处理的数据块的数量。如果数量过多，会产生很多小数据块，性能会受到影响。如果数量过少，并行性不佳，性能也会受到影响。因此，决定扫描仪线程数时，最好考虑一个分割内的平均数据大小，选择一个使数据块不会很小的值。经验法则是将单个块大小（MB）除以250得到的值作为扫描仪线程数。

增加并行性还需考虑的重要一点是集群中实际可用的CPU核数，确保并行计算数不超过实际CPU核数的75%至80%。

CPU核数约等于：

并行任务数x扫描仪线程数。其中并行任务数为分割数和执行器数x执行器核数两者之间的较小值。

数据加载性能调优

数据加载性能调优与查询性能调优差异很大。跟查询性能一样，数据加载性能也取决于可达到的并行性。在数据加载情况下，工作线程的数量决定并行的单元。因此，更多的执行器就意味着更多的执行器核数，每个执行器都可以提高数据加载性能。

同时，为了得到更好的性能，可在HDFS中配置如下参数。

表12-39 HDFS配置

参数	建议值
dfs.datanode.drop.cache.behind.reads	false
dfs.datanode.drop.cache.behind.writes	false
dfs.datanode.sync.behind.writes	false

压缩调优

Carbon结合少数轻量级压缩算法和重量级压缩算法来压缩数据。虽然这些算法可处理任何类型的数据，但如果数据经过排序，相似值在一起出现时，就会获得更好的压缩率。

Carbon数据加载过程中，数据基于Table中的列顺序进行排序，从而确保相似值在一起出现，以获得更好的压缩率。

由于Carbon按照Table中定义的列顺序将数据进行排序，因此列顺序对于压缩效率起重要作用。如果低基数维度位于左边，那么排序后的数据分区范围较小，压缩效率较高。如果高基数维度位于左边，那么排序后的数据分区范围较大，压缩效率较低。

内存调优

Carbon为内存调优提供了一个机制，其中数据加载会依赖于查询中需要的列。不论何时，接收到一个查询命令，将会获取到该查询中的列，并确保内存中这些列有数据加载。在该操作期间，如果达到内存的阈值，为了给查询需要的列提供内存空间，最少使用加载级别的文件将会被删除。

12.10 Storm

12.10.1 Storm性能调优

操作场景

通过调整Storm参数设置，可以提升特定业务场景下Storm的性能。

Storm参数入口：在FusionInsight Manager系统中，选择“服务管理 > Storm > 服务配置”，“参数类别”设置为“全部配置”。

拓扑调优

当需要提升Storm数据量处理性能时，可以通过拓扑调优的操作提高效率。建议在可靠性要求不高的场景下进行优化。

表12-40 调优参数

配置参数	缺省值	调优场景
topology.acker.executors	null	Acker的执行器数量。当业务应用对可靠性要求较低，允许不处理部分数据，可设置参数值为“null”或“0”，以关闭Acker的执行器，减少流控制，不统计消息时延，提高性能。
topology.max.spout.pending	null	Spout消息缓存数，仅在Acker不为0或者不为null的情况下生效。Spout将发送到下游Bolt的每条消息加入到pending队列，待下游Bolt处理完成并确认后，再从pending队列移除，当pending队列占满时Spout暂停消息发送。增加pending值可提高Spout的每秒消息吞吐量，提高性能，但延时同步增加。
topology.transfer.buffer.size	32	每个worker进程Disruptor消息队列大小，建议在4到32之间，增大消息队列可以提升吞吐量，但延时可能会增加。
RES_CPUSET_PERCENTAGE	80	设置各个节点上的Supervisor角色实例（包含其启动并管理的Worker进程）所使用的物理CPU百分比。根据Supervisor所在节点业务量需求，适当调整参数值，优化CPU使用率。

JVM调优

当应用程序需要处理大量数据从而占用更多的内存时，存在worker内存大于2GB的情况，推荐使用G1垃圾回收算法。

表12-41 调优参数

配置参数	缺省值	调优场景

配置参数	缺省值	调优场景
WORKER_GC_OPTS	-Xms1G -Xmx1G -XX:+UseG1GC -XX:+PrintGCDetails -Xloggc:artifacts/gc.log -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=1M -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=artifacts/heapdump	应用程序内存中需要保存大量数据，worker进程使用的内存大于2G，那么建议使用G1垃圾回收算法，可修改参数值为“-Xms2G -Xmx5G -XX:+UseG1GC”。

12.11 YARN

12.11.1 通过容器可重用性提高任务的完成效率

操作场景

说明：

容器可重用与任务优先级功能不能同时启用。如果同时启用，容器可重用功能可正常使用，任务优先级设置将会失效。

容器可重用性可以提高任务完成的速度。其优势如下所示：

- 对于HBase数据批量加载而言，容器可重用性的特性可作用于一些Map以及Reduce任务，使其能快速完成任务从而缩短HBase数据批量加载的时间。
- 减少容器调度的时间和初始化的时间。

一旦MapReduce作业被提交。它将分发至Map和Reduce任务中。然后应用管理器（以下简称AM）将执行如下操作。

1. AM向资源管理器（RM）申请容器去执行任务。所有容器将一起完成这个请求。
2. RM指定容器，之后AM将会联系节点管理器（NM）去启动容器。
3. 容器启动完毕，NM从AM拉取并执行任务。
4. 任务执行完毕，运行该任务的容器不会被立即终止，而是尝试向AM拉取下一个任务。
    - 若容器获取到新任务，则该容器会自我清空并初始化，以适用于新任务。
    - 若容器不能获取到新任务，则会请求终止自己的运行。

操作步骤

开启容器可重用性配置项。

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > Yarn > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称，修改表12-42参数值，然后重新下载并安装Yarn客户端，参数配置生效。或直接在客户端目录下修改：如“/opt/client/Yarn/config/mapred-site.xml”文件里修改表12-42参数。

表12-42 容器重用配置

参数	描述	默认值
mapreduce.container.reuse.enabled	该配置项设置为“true”则容器可重用，反之容器不可重用。	false
mapreduce.container.reuse.enforce.strict-locality	该配置项指定是否遵循严格数据本地化。如果设置为“true”，则只有本地节点上的任务能被分配到容器。	false

使用约束

- 容器的重用将受限于job中任务的总数。容器不会被不同的job共享。
- 容器不能被失败的任务重用。
- 设计任务时，应保证一个任务完成后堆栈里不存留任何对象。这对于容器重用中的数据一致性以及内存的优化非常重要。
- 如果容器重用是激活状态，job将不会释放容器给资源管理器，除非所有属于该job的任务都已完成，这种情况会影响到公平调度的原则。

12.11.2 抢占任务

操作场景

抢占任务可精简队列中的job运行并提高资源利用率，由ResourceManager的capacity scheduler实现，其简易流程如下：

1. 假设存在两个队列A和B。其中队列A的capacity为25%，队列B的capacity为75%。
2. 初始状态下，任务1发送给队列A，此任务需要75%的集群资源。之后任务2发送到了队列B，此任务需要50%的集群资源。
3. 任务1将会使用队列A提供的25%的集群资源，并从队列B获取的50%的集群资源。队列B保留25%的集群资源。
4. 启用抢占任务特性，则任务1使用的资源将会被抢占。队列B会从队列A中获取25%的集群资源以满足任务2的执行。
5. 当任务2完成后，集群中存在足够的资源时，任务1将重新开始执行。

操作步骤

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > YARN > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

表12-43 Preemption配置

参数	描述
yarn.resourcemanager.scheduler.monitor.enable	根据“yarn.resourcemanager.scheduler.monitor.policies”中的策略，启用新的scheduler监控。设置为“true”表示启用监控，并根据scheduler的信息，启动抢占功能。设置为“false”表示不启用。
yarn.resourcemanager.scheduler.monitor.policies	设置与scheduler配合的“SchedulingEditPolicy”的类的清单。
yarn.resourcemanager.monitor.capacity.preemption.observe_only	• 设置为“true”，则执行策略，但是不对集群资源进程抢占操作。 • 设置为“false”，则执行策略，且根据策略启用集群资源抢占的功能。
yarn.resourcemanager.monitor.capacity.preemption.monitoring_interval	根据策略监控的时间间隔，单位为毫秒。
yarn.resourcemanager.monitor.capacity.preemption.max_wait_before_kill	应用发送抢占需求到停止container（释放资源）的时间间隔，单位为毫秒。 默认情况下，若ApplicationMaster15秒内没有终止container，ResourceManage等待15秒后会强制终止。
yarn.resourcemanager.monitor.capacity.preemption.total_preemption_per_round	在一个周期内能够抢占资源的最大的比例。
yarn.resourcemanager.monitor.capacity.preemption.max_ignored_over_capacity	集群中资源总量乘以此配置项的值加上某个队列（例如队列A）原有的资源量为资抢占盲区。当队列A中的任务实际使用的资源超过该抢占盲区时，超过部分的资源会被抢占。 **说明：** 设置的值越小越有利于资源抢占。
yarn.resourcemanager.monitor.capacity.preemption.natural_termination_factor	设置抢占目标，Container只会抢占所配置比例的资源。 示例，如果设置为0.5，则在5*“yarn.resourcemanager.monitor.capacity.preemption.max_wait_before_kill”时间内，任务会回收所抢占资源的近95%。即接连抢占5次，每次抢占待抢占资源0.5，呈几何收敛，每次的时间间隔为“yarn.resourcemanager.monitor.capacity.preemption.max_wait_before_kill”

12.11.3 任务优先级

操作场景

集群的资源竞争场景如下：

1. 提交两个低优先级的应用Job 1和Job 2。
2. 正在运行中的Job 1和Job 2有部分task处于running状态，但由于集群或队列资源容量有限，仍有部分task未得到资源而处于pending状态。
3. 提交一个较高优先级的应用Job 3，此时会出现如下资源分配情况：当Job 1和Job 2中running状态的task运行结束并释放资源后，Job 3中处于pending状态的task将优先得到这部分新释放的资源。
4. Job 3完成后，资源释放给Job 1、Job 2继续执行。

用户可以在YARN中配置任务的优先级。任务优先级是通过ResourceManager的Capacity Scheduler实现的。

操作步骤

设置参数“yarn.app.priority”或“mapreduce.job.priority”，使用命令行接口或API接口设置任务优先级。若两种接口都使用，则考虑设置参数“yarn.app.priority”。

- 命令行接口。

  发送任务时，添加“-Dyarn.app.priority=<priority>”参数。

  <priority>可以设置为：

  - VERY_HIGH
  - HIGH
  - NORMAL
  - LOW
  - VERY_LOW

- API接口。

  用户也可以使用API配置对象的优先级。

  设置优先级，可通过**Configuration.set("yarn.app.priority", <priority>)**或**Job.setPriority(JobPriority priority)**设置。

  说明：

  <priority> 的可替换值参见• 命令行接口。。

12.11.4 节点配置调优

操作场景

合理配置大数据集群的调度器后，还可通过调节每个节点的可用内存、CPU资源及本地磁盘的配置进行性能调优。

具体包括以下配置项：

- 可用内存
- CPU虚拟核数
- 物理CPU使用百分比
- 内存和CPU资源的协调
- 本地磁盘

操作步骤

参数入口：

在FusionInsight Manager系统中，选择“服务管理 > Yarn > 服务配置”，“参数类别”类型设置为“全部配置”。在搜索框中输入参数名称。

- **可用内存**

  除了分配给操作系统、其他服务的内存外，剩余的资源应尽量分配给YARN。通过如下配置参数进行调整。

  例如，如果一个container默认使用512M，则内存使用的计算公式为：512M*container数。

  默认情况下，Map或Reduce container会使用1个虚拟CPU内核和1024MB内存，ApplicationMaster使用1536MB内存。

参数	描述	默认值
yarn.nodemanager.resource.memory-mb	设置可分配给容器的物理内存数量。建议配置为>24576（24G） 单位：MB	8192

- **CPU虚拟核数**

  建议将此配置设定在逻辑核数的1.5～2倍之间。如果上层计算应用对CPU的计算能力要求不高，可以配置为2倍的逻辑CPU。

参数	描述	默认值
yarn.nodemanager.resource.cpu-vcores	表示该节点上YARN可使用的虚拟CPU个数，默认是8。 目前推荐将该值设值为逻辑CPU核数的1.5～2倍之间。	8

- **物理CPU使用百分比**

  建议预留适量的CPU给操作系统和其他进程（数据库、HBase等）外，剩余的CPU核都分配给YARN。可以通过如下配置参数进行调整。

参数	描述	默认值
yarn.nodemanager.resource.percentage-physical-cpu-limit	表示该节点上YARN可使用的物理CPU百分比。默认是100，即不进行CPU控制，YARN可以使用节点全部CPU。该参数只支持查看，可通过调整YARN的RES_CPUSET_PERCENTAGE参数来修改本参数值。注意，目前推荐将该值设为可供YARN集群使用的CPU百分数。 例如：当前节点除了YARN服务外的其他服务（如HBase、HDFS、Hive等）及系统进程使用CPU为20%左右，则可以供YARN调度的CPU为1-20%=80%，即配置此参数为80。	90

- **本地磁盘**

  由于本地磁盘会提供给MapReduce写job执行的中间结果，数据量大。因此配置的原则是磁盘尽量多，且磁盘空间尽量大，单个达到百GB以上规模最好。简单的做法是配置和data node相同的磁盘，只在最下一级目录上不同即可。

  说明：

  多个磁盘之间使用逗号隔开。

参数	描述	默认值

参数	描述	默认值
yarn.nodemanager.log-dirs	日志存放地址（可配置多个目录）。 容器日志的存储位置。默认值为%{@auto.detect.datapart.nm.logs}。如果有数据分区，基于该数据分区生成一个类似/srv/BigData/hadoop/data1/nm/containerlogs,/srv/BigData/hadoop/data2/nm/containerlogs的路径清单。如果没有数据分区，生成默认路径/srv/BigData/yarn/data1/nm/containerlogs。除了使用表达式以外，还可以输入完整的路径清单，比如/srv/BigData/yarn/data1/nm/containerlogs或/srv/BigData/yarn/data1/nm/containerlogs,/srv/BigData/yarn/data2/nm/containerlogs。这样数据就会存储在所有设置的目录中，一般会是在不同的设备中。为保证磁盘IO负载均衡，最好提供几个路径且每个路径都对应一个单独的磁盘。应用程序的本地化后的日志目录存在于相对路径/application_%{appid}中。单独容器的日志目录，即container_{$contid}，是该路径下的子目录。每个容器目录都含容器生成的stderr、stdin及syslog文件。要新增目录，比如新增/srv/BigData/yarn/data2/nm/containerlogs目录，应首先删除/srv/BigData/yarn/data2/nm/containerlogs下的文件。之后，为/srv/BigData/yarn/data2/nm/containerlogs赋予跟/srv/BigData/yarn/data1/nm/containerlogs一样的读写权限，再将/srv/BigData/yarn/data1/nm/containerlogs修改为/srv/BigData/yarn/data1/nm/containerlogs,/srv/BigData/yarn/data2/nm/containerlogs。可以新增目录，但不要修改或删除现有目录。否则，NodeManager的数据将丢失，且服务将不可用。 【默认值】%{@auto.detect.datapart.nm.logs} 【注意】请谨慎修改该项。如果配置不当，将造成服务不可用。当角色级别的该配置项修改后，所有实例级别的该配置项都将被修改。如果实例级别的配置项修改后，其他实例的该配置项的值保持不变。	%{@auto.detect.datapart.
yarn.nodemanager.local-dirs	本地化后的文件的存储位置。默认值为%{@auto.detect.datapart.nm.localdir}。如果有数据分区，基于该数据分区生成一个类似/srv/BigData/hadoop/data1/nm/localdir,/srv/BigData/hadoop/data2/nm/localdir的路径清单。如果没有数据分区，生成默认路径/srv/BigData/yarn/data1/nm/localdir。除了使用表达式以外，还可以输入完整的路径清单，比如/srv/BigData/yarn/data1/nm/localdir或/srv/BigData/yarn/data1/nm/localdir,/srv/BigData/yarn/data2/nm/localdir。这样数据就会存储在所有设置的目录中，一般会是在不同的设备中。为保证磁盘IO负载均衡，最好提供几个路径且每个路径都对应一个单独的磁盘。应用程序的本地化后的文件目录存在于相对路径/usercache/%{user}/appcache/application_%{appid}中。单独容器的工作目录，即container_%{contid}，是该路径下的子目录。要新增目录，比如新增/srv/BigData/yarn/data2/nm/localdir目录，应首先删除/srv/BigData/yarn/data2/nm/localdir下的文件。之后，为/srv/BigData/hadoop/data2/nm/localdir赋予跟/srv/BigData/hadoop/data1/nm/localdir一样的读写权限，再将/srv/BigData/yarn/data1/nm/localdir修改为/srv/BigData/yarn/data1/nm/localdir,/srv/BigData/yarn/data2/nm/localdir。可以新增目录，但不要修改或删除现有目录。否则，NodeManager的数据将丢失，且服务将不可用。 【默认值】%{@auto.detect.datapart.nm.localdir} 【注意】请谨慎修改该项。如果配置不当，将造成服务不可用。当角色级别的该配置项修改后，所有实例级别的该配置项都将被修改。如果实例级别的配置项修改后，其他实例的该配置项的值保持不变。	%{@auto.detect.datapart.nr

12.11.5 JVM参数优化

操作场景

当集群数据量达到一定规模后，JVM的默认配置将无法满足集群的业务需求，轻则集群变慢，重则集群服务不可用。所以需要根据实际的业务情况进行合理的JVM参数配置，提高集群性能。

操作步骤

参数入口：

Yarn角色相关的JVM参数需要配置在“${HADOOP_HOME}/etc/hadoop”目录下的“yarn-env.sh”文件中。

每个角色都有各自的JVM参数配置变量，如表12-44。

表12-44 Yarn相关JVM参数配置变量

变量名	变量影响的角色
YARN_OPTS	该变量中设置的参数，将影响Yarn的所有角色。
YARN_CLIENT_OPTS	该变量中设置的参数，将影响Yarn的Client进程。
YARN_RESOURCEMANAGER_OPTS	该变量中设置的参数，将影响Yarn的ResourceManager。
YARN_HISTORYSERVER_OPTS	该变量中设置的参数，将影响Yarn的HistoryServer。
YARN_TIMELINESERVER_OPTS	该变量中设置的参数，将影响Yarn的TimelineServer。
YARN_NODEMANAGER_OPTS	该变量中设置的参数，将影响Yarn的NodeManager。
YARN_PROXYSERVER_OPTS	该变量中设置的参数，将影响Yarn的ProxyServer。

配置方式举例：

export HADOOP_NAMENODE_OPTS="-Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,N