# 背景介绍

## 背景

Hive on Spark是由Cloudera发起，由Intel、MapR等公司共同参与的开源项目，其目的是把Spark作为Hive的一个计算引擎，将Hive的查询作为Spark的任务提交到Spark集群上进行计算。通过该项目，可以提高Hive查询的性能，同时为已经部署了Hive或者Spark的用户提供了更加灵活的选择，从而进一步提高Hive和Spark的普及率。

## 简介

Hive on Spark是从Hive on MapReduce演进而来，Hive的整体解决方案很不错，但是从查询提交到结果返回需要相当长的时间，查询耗时太长，这个主要原因就是由于Hive原生是基于MapReduce的，那么如果我们不生成MapReduce Job，而是生成Spark Job，就可以充分利用Spark的快速执行能力来缩短HiveQL的响应时间。Hive on Spark现在是Hive组件(从Hive1.1 release之后)的一部分。

## 与SparkSQL的区别

SparkSQL和Hive On Spark都是在Spark上实现SQL的解决方案。Spark早先有Shark项目用来实现SQL层，不过后来推翻重做了，就变成了SparkSQL。这是Spark官方Databricks的项目，Spark项目本身主推的SQL实现。Hive On Spark比SparkSQL稍晚。Hive原本是没有很好支持MapReduce之外的引擎的，而Hive On Tez项目让Hive得以支持和Spark近似的Planning结构（非MapReduce的DAG）。所以在此基础上，Cloudera主导启动了Hive On Spark。这个项目得到了IBM，Intel和MapR的支持（但是没有Databricks）。

# 环境配置

CDH 5.4以后引入了Hive on Spark，但是在CDH5.5.x中是不被推荐使用的。最好是在CDH5.5以后的版本中使用。推荐使用HiveServer2的Beeline，当然使用Hive CLI也是可以的。

## 注意事项

为了让Hive工作在spark上，你必须在HiveServer2所在机器上部署spark gateway角色。另外，hive on spark不能读取spark的配置，也不能提交spark作业。

在使用过程中，需要手动设置如下命令，以便让之后的查询都能使用spark引擎。

|  |
| --- |
| set hive.execution.engine=spark; |

## 启用hive on spark

默认hive on spark是禁用的，需要在Cloudera Manager中启用。

1.登录CM界面，打开hive服务。

2.单击配置标签，查找enable.hive.on.spark或者hive.execution.engine属性（版本不一样而属性不一样）。



3.勾选Enbale Hive on Spark(Unsupported)，并保存更改。

4.查找Spark on YARN或者YARN 服务上的 Spark，并勾选保存。



5.保存后，重启hive服务使其生效。

## 配置hive属性

为了改善性能，Cloudera推荐配置到HiveServer2服务中

是否从metstore获取行数统计：

|  |
| --- |
| hive.stats.fetch.column.stats=true |

自动使用索引，默认是不开启，设置为false：

|  |
| --- |
| hive.optimize.index.filter=true |

经过上述配置，就可以在Hive CLI或用HiveServer2直接使用hive on spark了，使用和原来Hive on MapReduce没什么区别，只是在使用前执行下set hive.execution.engine=spark就可以使用spark引擎来运行hive了。

# 优化

## 配置Yarn

Yarn需要配置两个参数：

yarn.nodemanager.resource.cpu-vcores和yarn.nodemanager.resource.memory-mb。

yarn.nodemanager.resource.cpu-vcores代表可以为container分配的CPU 内核的数量。yarn.nodemanager.resource.memory-mb代表可分配给容器的物理内存大小。

### 1、配置cpu core

为每个服务分配一个core，为操作系统预留2个core，剩余的可用的core分配给yarn。比如集群共有16个core，可以留出4个，剩余的12个core分配给yarn。

### 2、配置内存

　　设置Yarn内存为36G

## 配置Spark

给Yarn分配完资源后，需要配置一些Spark的参数，设置Spark可使用的资源。包括executor和Driver的内存，分配executor和设置并行度。

### 1、配置executor内存

在配置executor的内存大小的时候，需要考虑以下因素:

增加executor的内存可以优化map join。但是会增加GC的时间。

在某些情况下，HDFS客户端没有并行处理多个写请求，在有多个请求竞争资源的时候会出现一个executor使用过多的core。

尽可能的减少空闲的core的个数，cloudera推荐设置spark.executor.cores为4，5，6，这取决于给yarn分配的资源。

比如说，我们有12个core可用，我们可以设置为4，这样12/4余数为0，设置为5的话会剩余两个空闲。设置4个可使得空闲的core尽可能的少。

这样配置之后我们可以同时运行三个executor，每个executor最多可以运行4个任务（每个core一个）。

还有一点是要求spark.executor.memoryOverhead和spark.executor.memory的和不能超过yarn.scheduler.maximum-allocation-mb设置的值。我的scheduler请求最大内存分配的是12G。

### 2、配置Driver内存

Spark Driver端的配置如下：

spark.driver.memory：当hive运行在spark上时，driver端可用的最大Java堆内存。

spark.yarn.driver.memoryOverhead：每个driver可以额外从yarn请求的堆内存大小。这个参数加上spark.driver.memory就是yarn为driver端的JVM分配的总内存。

Spark在Driver端的内存不会直接影响性能，但是在没有足够内存的情况下在driver端强制运行Spark任务需要调整。

### 3、设置executor个数

集群的executor个数设置由集群中每个节点的executor个数和集群的worker个数决定，如果集群中有3个worker，则Hive On Spark可以使用的executor最大个数是12个（3 \* 4）。

Hive的性能受可用的executor的个数影响很明显，一般情况下，性能和executor的个数成正比，4个executor的性能大约是2个executor性能的一倍，但是性能在executor设置为一定数量的时候会达到极值，达到这个极值之后再增加executor的个数不会增加性能，反而有可能会为集群增加负担。

### 4、动态分配executor

设置spark.executor.instances到最大值可以使得Spark集群发挥最大性能。但是这样有个问题是当集群有多个用户运行Hive查询时会有问题，应避免为每个用户的会话分配固定数量的executor，因为executor分配后不能为其他用户的查询使用，如果有空闲的executor，在生产环境中，计划分配好executor可以更充分的利用Spark集群资源。

Spark允许动态的给Spark作业分配集群资源，cloudera推荐开启动态分配。

### 5、设置并行度

为了更加充分的利用executor，必须同时允许足够多的并行任务。在大多数情况下，hive会自动决定并行度，但是有时候我们可能会手动地调整并行度。

在输入端，map task的个数等于输入端按照一定格式切分的生成的数目，Hive On Spark的输入格式是CombineHiveInputFormat，可以根据需要切分底层输入格式。调整hive.exec.reducers.bytes.per.reducer控制每个reducer处理多少数据。但是实际情况下，Spark相比于MapReduce，对于指定的hive.exec.reducers.bytes.per.reducer不敏感。我们需要足够的任务让可用的executor保持工作不空闲，当Hive能够生成足够多的任务，尽可能的利用空闲的executor。

## 配置Hive

Hive on Spark的配置大部分即使不使用Hive，也可以对这些参数调优。但是hive.auto.convert.join.noconditionaltask.size这个参数是将普通的join转化成map join的阈值，这个参数调优对于性能有很大影响。MapReduce和Spark都可以通过这个参数进行调优，但是这个参数在Hive On MR上的含义不同于Hive On Spark。

数据的大小由两个统计量标识：

·totalSize 磁盘上数据的大小

·rawDataSize 内存中数据的大小

Hive On MapReduce使用的是totalSize，Spark使用rawDataSize。数据由于经过一系列压缩、序列化等操作，即使是相同的数据集，也会有很大的不同，对于Hive On Spark，需要设置　　　hive.auto.convert.join.noconditionaltask.size，将普通的join操作转化成map join来提升性能，集群资源充足的情况下可以把这个参数的值适当调大，来更多的触发map join。但是设置太高的话，小表的数据会占用过多的内存导致整个任务因为内存耗尽而失败，所有这个参数需要根据集群的资源来进行调整。

Cloudera推荐配置两个额外的配置项：

hive.stats.fetch.column.stats=true

　 hive.optimize.index.filter=true

以下还整理了一些配置项用于hive调优：

hive.optimize.reducededuplication.min.reducer=4

hive.optimize.reducededuplication=true

hive.merge.mapfiles=true

hive.merge.mapredfiles=false

hive.merge.smallfiles.avgsize=16000000

hive.merge.size.per.task=256000000

hive.merge.sparkfiles=true

hive.auto.convert.join=true

hive.auto.convert.join.noconditionaltask=true

hive.auto.convert.join.noconditionaltask.size=20M(might need to increase for Spark, 200M)

hive.optimize.bucketmapjoin.sortedmerge=false

hive.map.aggr.hash.percentmemory=0.5

hive.map.aggr=true

hive.optimize.sort.dynamic.partition=false

hive.stats.autogather=true

hive.stats.fetch.column.stats=true

hive.compute.query.using.stats=true

hive.limit.pushdown.memory.usage=0.4 (MR and Spark)

hive.optimize.index.filter=true

hive.exec.reducers.bytes.per.reducer=67108864

hive.smbjoin.cache.rows=10000

hive.fetch.task.conversion=more

hive.fetch.task.conversion.threshold=1073741824

hive.optimize.ppd=true

### 设置Pre-warming Yarn Container

我们使用Hive On Spark的时候，提交第一个查询时，看到查询结果可能会有比较长的延迟，但是再次运行相同的SQL查询，完成速度要比第一个查询快得多。

当Spark使用yarn管理资源调度时，Spark executor需要额外的时间来启动和初始化，在程序运行之前，Spark不会等待所有的executor准备好之后运行，所以在任务提交到集群之后，仍有一些executor处于非启动状态。在Spark上运行的作业运行速度与executor个数相关，当可用的executor的个数没有达到最大值的时候，作业达不到最大的并行性，所以Hive上提交的第一个SQL查询会慢。

如果是在长时间会话这个问题影响很小。因为只有执行第一个SQL的时候会慢，问题不大。但是很多时候我们写的Hive脚本，需要用一些调度框架去启动（如Oozie）。这时候我们需要考虑进行优化。

为了减少启动时间，我们可以开启container pre-warming机制，开启后只有当任务请求的所有executor准备就绪，作业才会开始运行。这样会提升Spark作业的并行度。