# 第1章 Hive调优

## 1.1 Fetch抓取(不使用MR)

Fetch抓取是指，Hive中对某些情况的查询可以不必使用MapReduce计算。例如：SELECT \* FROM emp;在这种情况下，Hive可以简单地读取emp对应的存储目录下的文件，然后输出查询结果到控制台。

在hive-default.xml.template文件中hive.fetch.task.conversion默认是more，老版本hive默认是minimal，该属性修改为more以后，在全局查找、字段查找、limit查找等都不走mapreduce。

<property>

<name>hive.fetch.task.conversion</name>

<value>more</value>

<description>

Expects one of [none, minimal, more].

Some select queries can be converted to single FETCH task minimizing latency.

Currently the query should be single sourced not having any subquery and should not have any aggregations or distincts (which incurs RS), lateral views and joins.

0. none : disable hive.fetch.task.conversion

1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only

2. more : SELECT, FILTER, LIMIT only (support TABLESAMPLE and virtual columns)

</description>

</property>

|  |
| --- |
|  |

**1）案例实操：**

（1）把hive.fetch.task.conversion设置成none，然后执行查询语句，都会执行mapreduce程序。

hive (default)> set hive.fetch.task.conversion=none;

hive (default)> select \* from emp;

hive (default)> select ename from emp;

hive (default)> select ename from emp limit 3;

（2）把hive.fetch.task.conversion设置成more，然后执行查询语句，如下查询方式都不会执行mapreduce程序。

hive (default)> set hive.fetch.task.conversion=more;

hive (default)> select \* from emp;

hive (default)> select ename from emp;

hive (default)> select ename from emp limit 3;

## 1.2 本地模式(调试使用)

大多数的Hadoop Job是需要Hadoop提供的完整的可扩展性来处理大数据集的。不过，有时Hive的输入数据量是非常小的。在这种情况下，为查询触发执行任务消耗的时间可能会比实际job的执行时间要多的多。对于大多数这种情况，Hive可以通过本地模式在单台机器上处理所有的任务。对于小数据集，执行时间可以明显被缩短。

用户可以通过设置hive.exec.mode.local.auto的值为true，来让Hive在适当的时候自动启动这个优化。

set hive.exec.mode.local.auto=true;  //开启本地mr

//设置local mr的最大输入数据量，当输入数据量小于这个值时采用local  mr的方式，默认为134217728，即128M

set hive.exec.mode.local.auto.inputbytes.max=50000000;

//设置local mr的最大输入文件个数，当输入文件个数小于这个值时采用local mr的方式，默认为4

set hive.exec.mode.local.auto.input.files.max=10;

**1）案例实操：**

（1）开启本地模式，并执行查询语句

hive (default)> set hive.exec.mode.local.auto=true;

hive (default)> select \* from emp cluster by deptno;

Time taken: 1.328 seconds, Fetched: 14 row(s)

（2）关闭本地模式，并执行查询语句

hive (default)> set hive.exec.mode.local.auto=false;

hive (default)> select \* from emp cluster by deptno;

Time taken: 20.09 seconds, Fetched: 14 row(s)

## 1.3 表的优化

### 1.3.1 小表、大表Join

实际测试发现：新版的hive已经对小表JOIN大表和大表JOIN小表进行了优化。小表放在左边和右边已经没有明显区别。

### 1.3.2 hive解决数据倾斜

#### 1.空KEY问题

**1）空KEY过滤**

有时join超时是因为某些key对应的数据太多，而相同key对应的数据都会发送到相同的reducer上，从而导致内存不够。此时我们应该仔细分析这些异常的key，很多情况下，这些key对应的数据是异常数据，我们需要在SQL语句中进行过滤。例如key对应的字段为空，操作如下：

案例实操

（1）配置历史服务器

配置mapred-site.xml

<property>

<name>mapreduce.jobhistory.address</name>

<value>hadoop102:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>hadoop102:19888</value>

</property>

启动历史服务器

sbin/mr-jobhistory-daemon.sh start historyserver

查看jobhistory

<http://hadoop102:19888/jobhistory>

（2）创建空id表

// 创建空id表

create table nullidtable(id bigint, t bigint, uid string, keyword string, url\_rank int, click\_num int, click\_url string) row format delimited fields terminated by '\t';

（3）加载空id数据到空id表中

hive (default)> load data local inpath '/opt/module/hive/datas/nullid' into table nullidtable;

（4）测试不过滤空id

hive (default)> insert overwrite table jointable select n.\* from nullidtable n

left join bigtable o on n.id = o.id;

Time taken: 42.038 seconds

Time taken: 37.284 seconds

（5）测试过滤空id

hive (default)> insert overwrite table jointable select n.\* from (select \* from nullidtable where id is not null ) n left join bigtable o on n.id = o.id;

Time taken: 31.725 seconds

Time taken: 28.876 seconds

**2）空key转换**

有时虽然某个key为空对应的数据很多，但是相应的数据不是异常数据，必须要包含在join的结果中，此时我们可以表a中key为空的字段赋一个随机的值，使得数据随机均匀地分不到不同的reducer上。例如：

案例实操：

不随机分布空null值：

（1）设置5个reduce个数

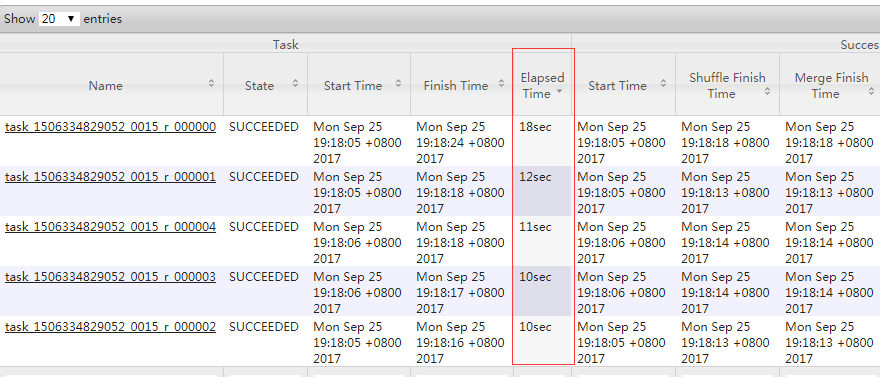
set mapreduce.job.reduces = 5;

（2）JOIN两张表

insert overwrite table jointable

select n.\* from nullidtable n left join bigtable b on n.id = b.id;

结果：如下图所示，可以看出来，出现了数据倾斜，某些reducer的资源消耗远大于其他reducer。



随机分布空null值

（1）设置5个reduce个数

set mapreduce.job.reduces = 5;

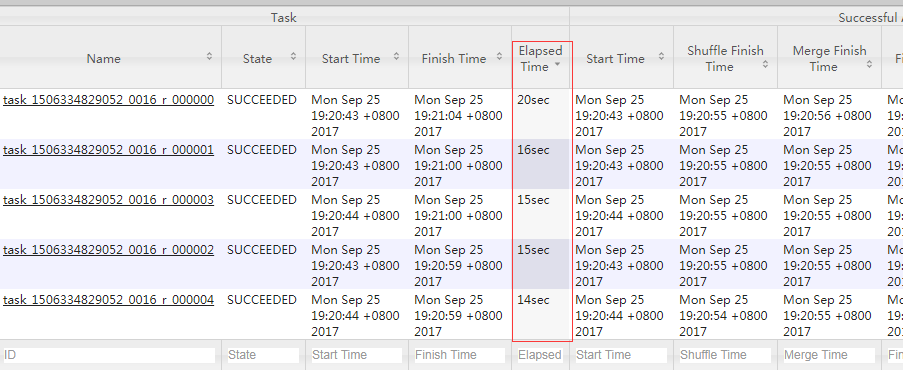
（2）JOIN两张表

insert overwrite table jointable

select n.\* from nullidtable n full join bigtable o on

nvl(n.id,rand()) = o.id;

结果：如下图所示，可以看出来，消除了数据倾斜，负载均衡reducer的资源消耗



#### 2 MapJoin

如果不指定MapJoin或者不符合MapJoin的条件，那么Hive解析器会将Join操作转换成Common Join，即：在Reduce阶段完成join。容易发生数据倾斜。可以用MapJoin把小表全部加载到内存在map端进行join，避免reducer处理。

**1）开启MapJoin参数设置**

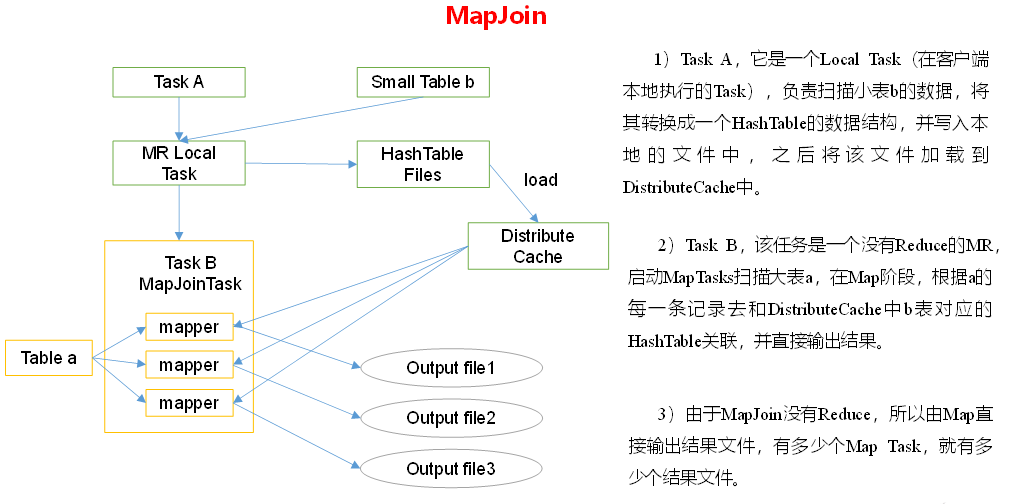
（1）设置自动选择Mapjoin

set hive.auto.convert.join = true; 默认为true

（2）大表小表的阈值设置（默认25M以下认为是小表）：

set hive.mapjoin.smalltable.filesize=25000000;

**2）MapJoin工作机制**



**3）案例实操：**

（1）开启Mapjoin功能

set hive.auto.convert.join = true; 默认为true

（2）执行小表JOIN大表语句

注意：此时小表作为主表，所有数据都要写出去，因此此时会走reduce，mapjoin失效

insert overwrite table jointable

select b.id, b.t, b.uid, b.keyword, b.url\_rank, b.click\_num, b.click\_url

from smalltable s

left join bigtable b

on s.id = b.id;

|  |
| --- |
|  |

Time taken: 24.594 seconds

（3）执行大表JOIN小表语句

insert overwrite table jointable

select b.id, b.t, b.uid, b.keyword, b.url\_rank, b.click\_num, b.click\_url

from bigtable b

left join smalltable s

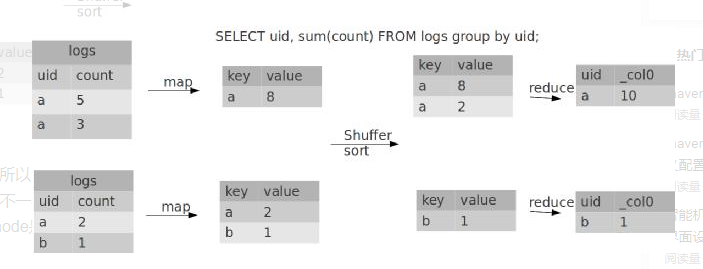
on s.id = b.id;

|  |
| --- |
|  |

Time taken: 24.315 seconds

#### 3 Group By

默认情况下，Map阶段同一Key数据分发给一个reduce，当一个key数据过大时就倾斜了。



并不是所有的聚合操作都需要在Reduce端完成，很多聚合操作都可以先在Map端进行部分聚合，最后在Reduce端得出最终结果。

**1）开启Map端聚合参数设置**

（1）是否在Map端进行聚合，默认为True

set hive.map.aggr = true

（2）在Map端进行聚合操作的条目数目

set hive.groupby.mapaggr.checkinterval = 100000

（3）有数据倾斜的时候进行负载均衡（默认是false）

set hive.groupby.skewindata = true

当选项设定为 true，生成的查询计划会有两个MR Job。第一个MR Job中，Map的输出结果会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果，这样处理的结果是相同的Group By Key有可能被分发到不同的Reduce中，从而达到负载均衡的目的；第二个MR Job再根据预处理的数据结果按照Group By Key分布到Reduce中（这个过程可以保证相同的Group By Key被分布到同一个Reduce中），最后完成最终的聚合操作。

hive (default)> select deptno from emp group by deptno;

Stage-Stage-1: Map: 1 Reduce: 5 Cumulative CPU: 23.68 sec HDFS Read: 19987 HDFS Write: 9 SUCCESS

Total MapReduce CPU Time Spent: 23 seconds 680 msec

OK

deptno

10

20

30

优化以后

hive (default)> set hive.groupby.skewindata = true;

hive (default)> select deptno from emp group by deptno;

Stage-Stage-1: Map: 1 Reduce: 5 Cumulative CPU: 28.53 sec HDFS Read: 18209 HDFS Write: 534 SUCCESS

Stage-Stage-2: Map: 1 Reduce: 5 Cumulative CPU: 38.32 sec HDFS Read: 15014 HDFS Write: 9 SUCCESS

Total MapReduce CPU Time Spent: 1 minutes 6 seconds 850 msec

OK

deptno

10

20

30

### 1.3.3 Count(Distinct) 去重统计

数据量小的时候无所谓，数据量大的情况下，由于COUNT DISTINCT操作需要用一个Reduce Task来完成，这一个Reduce需要处理的数据量太大，就会导致整个Job很难完成，一般COUNT DISTINCT使用先GROUP BY再COUNT的方式替换,但是需要注意group by造成的数据倾斜问题.

虽然会多用一个Job来完成，但在数据量大的情况下，这个绝对是值得的。

### 1.3.4 笛卡尔积

尽量避免笛卡尔积，join的时候不加on条件，或者无效的on条件，Hive只能使用1个reducer来完成笛卡尔积。

### 1.3.5 行列过滤

列处理：在SELECT中，只拿需要的列，如果有，尽量使用分区过滤，少用SELECT \*。

行处理：在分区剪裁中，当使用外关联时，如果将副表的过滤条件写在Where后面，那么就会先全表关联，之后再过滤，比如：

### 1.3.6 分区

#### 1 分区

**引入分区表（需要根据日期对日志进行管理, 通过部门信息模拟）**

dept\_20200401.log

dept\_20200402.log

dept\_20200403.log

**2）创建分区表语法**

hive (default)> create table dept\_partition(

deptno int, dname string, loc string

)

partitioned by (day string)

row format delimited fields terminated by '\t';

|  |
| --- |
| 注意：分区字段不能是表中已经存在的数据，可以将分区字段看作表的伪列。 |

**3）加载数据到分区表中**

1. **数据准备**

dept\_20200401.log

10 ACCOUNTING 1700

20 RESEARCH 1800

dept\_20200402.log

30 SALES 1900

40 OPERATIONS 1700

dept\_20200403.log

50 TEST 2000

60 DEV 1900

1. **加载数据**

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200401.log' into table dept\_partition partition(day='20200401');

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200402.log' into table dept\_partition partition(day='20200402');

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200403.log' into table dept\_partition partition(day='20200403');

注意：分区表加载数据时，必须指定分区

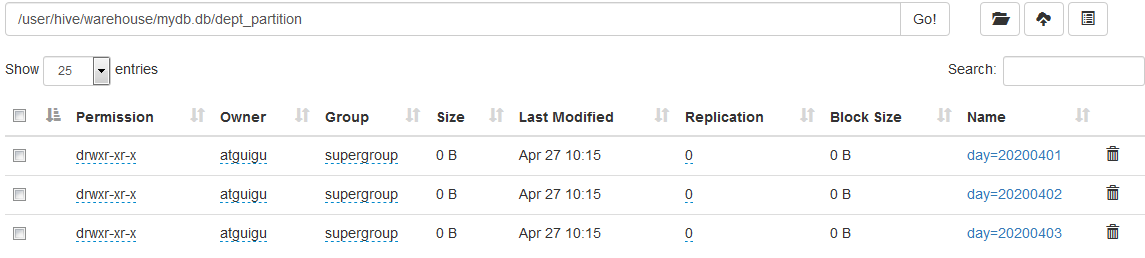


图 分区表

**4）查询分区表中数据**

单分区查询

hive (default)> select \* from dept\_partition where day='20200401';

多分区联合查询

hive (default)> select \* from dept\_partition where day='20200401'

union

select \* from dept\_partition where day='20200402'

union

select \* from dept\_partition where day='20200403';

hive (default)> select \* from dept\_partition where day='20200401' or

day='20200402' or day='20200403' ;

**5）查看分区表有多少分区**

hive> show partitions dept\_partition;

**6）增加分区**

创建单个分区

hive (default)> alter table dept\_partition add partition(day='20200404') ;

同时创建多个分区（分区之间不能有逗号）

hive (default)> alter table dept\_partition add partition(day='20200405') partition(day='20200406');

**7）删除分区**

删除单个分区

hive (default)> alter table dept\_partition drop partition (day='20200406');

同时删除多个分区（分区之间必须有逗号）

hive (default)> alter table dept\_partition drop partition (day='20200404'), partition(day='20200405');

**8）查看分区表结构**

hive> desc formatted dept\_partition;

# Partition Information

# col\_name data\_type comment

month string

#### 2 分区表二级分区

思考: 如何一天的日志数据量也很大，如何再将数据拆分?

**1）创建二级分区表**

|  |
| --- |
| hive (default)> create table dept\_partition2(  deptno int, dname string, loc string  )  partitioned by (day string, hour string)  row format delimited fields terminated by '\t'; |

**2）正常的加载数据**

（1）加载数据到二级分区表中

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200401.log' into table

dept\_partition2 partition(day='20200401', hour='12');

（2）查询分区数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='12';

**3）把数据直接上传到分区目录上，让分区表和数据产生关联的三种方式**

（1）方式一：上传数据后修复

上传数据

hive (default)> dfs -mkdir -p

/user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=13;

hive (default)> dfs -put /opt/module/datas/dept\_20200401.log /user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=13;

查询数据（查询不到刚上传的数据）

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='13';

执行修复命令

hive> msck repair table dept\_partition2;

再次查询数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='13';

（2）方式二：上传数据后添加分区

上传数据

hive (default)> dfs -mkdir -p

/user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=14;

hive (default)> dfs -put /opt/module/hive/datas/dept\_20200401.log /user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=14;

执行添加分区

hive (default)> alter table dept\_partition2 add partition(day='20200401',hour='14');

查询数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='14';

（3）方式三：创建文件夹后load数据到分区

创建目录

hive (default)> dfs -mkdir -p

/user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=15;

上传数据

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200401.log' into table

dept\_partition2 partition(day='20200401',hour='15');

查询数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='15';

#### 3 动态分区调整

关系型数据库中，对分区表Insert数据时候，数据库自动会根据分区字段的值，将数据插入到相应的分区中，Hive中也提供了类似的机制，即动态分区(Dynamic Partition)，只不过，使用Hive的动态分区，需要进行相应的配置。

**1）开启动态分区参数设置**

（1）开启动态分区功能（默认true，开启）

hive.exec.dynamic.partition=true

（2）设置为非严格模式（动态分区的模式，默认strict，表示必须指定至少一个分区为静态分区，nonstrict模式表示允许所有的分区字段都可以使用动态分区。）

hive.exec.dynamic.partition.mode=nonstrict

（3）在所有执行MR的节点上，最大一共可以创建多少个动态分区。默认1000

hive.exec.max.dynamic.partitions=1000

（4）在每个执行MR的节点上，最大可以创建多少个动态分区。该参数需要根据实际的数据来设定。比如：源数据中包含了一年的数据，即day字段有365个值，那么该参数就需要设置成大于365，如果使用默认值100，则会报错。

hive.exec.max.dynamic.partitions.pernode=100

（5）整个MR Job中，最大可以创建多少个HDFS文件。默认100000

hive.exec.max.created.files=100000

（6）当有空分区生成时，是否抛出异常。一般不需要设置。默认false

hive.error.on.empty.partition=false

**2）案例实操**

需求：将dept表中的数据按照地区（loc字段），插入到目标表dept\_partition的相应分区中。

（1）创建目标分区表

hive (default)> create table dept\_partition\_dy(id int, name string) partitioned by (loc int) row format delimited fields terminated by '\t';

|  |
| --- |
|  |

（2）设置动态分区

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

hive (default)> insert into table dept\_partition\_dy partition(loc) select deptno, dname, loc from dept;

（3）查看目标分区表的分区情况

hive (default)> show partitions dept\_partition\_dy;

思考：目标分区表是如何匹配到分区字段的？

#### 4 分桶表

分区提供一个隔离数据和优化查询的便利方式。不过，并非所有的数据集都可形成合理的分区。对于一张表或者分区，Hive 可以进一步组织成桶，也就是更为细粒度的数据范围划分。

分桶是将数据集分解成更容易管理的若干部分的另一个技术。

分区针对的是数据的存储路径；分桶针对的是数据文件。

**1）先创建分桶表，通过直接导入数据文件的方式**

（1）数据准备

1001 ss1

1002 ss2

1003 ss3

1004 ss4

1005 ss5

1006 ss6

1007 ss7

1008 ss8

1009 ss9

1010 ss10

1011 ss11

1012 ss12

1013 ss13

1014 ss14

1015 ss15

1016 ss16

（2）创建分桶表

create table stu\_buck(id int, name string)

clustered by(id)

into 4 buckets

row format delimited fields terminated by '\t';

（3）查看表结构

hive (default)> desc formatted stu\_buck;

Num Buckets: 4

（4）导入数据到分桶表中（hive新版本load数据跑mr，因此要改用hdfs路径导数据）

hive (default)> load data local inpath '/opt/module/hive/datas/student.txt' into table stu\_buck;

（5）查看创建的分桶表中是否分成4个桶

查看hdfs(http://hadoop102:9870)

（6）查询分桶的数据

hive(default)> select \* from stu\_buck;

分桶规则：

根据结果可知：Hive的分桶采用对分桶字段的值进行哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中

## 1.4 合理设置Map及Reduce数

**1）**通常情况下，作业会通过input的目录产生一个或者多个map任务。

主要的决定因素有：input的文件总个数，input的文件大小，集群设置的文件块大小。

**2）**是不是map数越多越好？

答案是否定的。如果一个任务有很多小文件（远远小于块大小128m），则每个小文件也会被当做一个块，用一个map任务来完成，而一个map任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的map数是受限的。

**3）**是不是保证每个map处理接近128m的文件块，就高枕无忧了？

答案也是不一定。比如有一个127m的文件，正常会用一个map去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，如果map处理的逻辑比较复杂，用一个map任务去做，肯定也比较耗时。

针对上面的问题2和3，我们需要采取两种方式来解决：即减少map数和增加map数；

### 1.4.1 复杂文件增加Map数

当input的文件都很大，任务逻辑复杂，map执行非常慢的时候，可以考虑增加Map数，来使得每个map处理的数据量减少，从而提高任务的执行效率。

增加map的方法为：根据

computeSliteSize(Math.max(minSize,Math.min(maxSize,blocksize)))=blocksize=128M公式，调整maxSize最大值。让maxSize最大值低于blocksize就可以增加map的个数。

案例实操：

**1）执行查询**

hive (default)> select count(\*) from emp;

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1

**2）设置最大切片值为100个字节**

hive (default)> set mapreduce.input.fileinputformat.split.maxsize=100;

hive (default)> select count(\*) from emp;

Hadoop job information for Stage-1: number of mappers: 6; number of reducers: 1

### 1.4.2 小文件进行合并

**1）**在map执行前合并小文件，减少map数：CombineHiveInputFormat具有对小文件进行合并的功能（系统默认的格式）。HiveInputFormat没有对小文件合并功能。

set hive.input.format= org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;

**2）**在Map-Reduce的任务结束时合并小文件的设置：

在map-only任务结束时合并小文件，默认true

SET hive.merge.mapfiles = true;

在map-reduce任务结束时合并小文件，默认false

SET hive.merge.mapredfiles = true;

合并文件的大小，默认256M

SET hive.merge.size.per.task = 268435456;

当输出文件的平均大小小于该值时，启动一个独立的map-reduce任务进行文件merge

SET hive.merge.smallfiles.avgsize = 16777216;

### 1.4.3 合理设置Reduce数

**1）调整reduce个数方法一**

（1）每个Reduce处理的数据量默认是256MB

hive.exec.reducers.bytes.per.reducer=256000000

（2）每个任务最大的reduce数，默认为1009

hive.exec.reducers.max=1009

（3）计算reducer数的公式

N=min(参数2，总输入数据量/参数1)

**2）调整reduce个数方法二**

在hadoop的mapred-default.xml文件中修改

设置每个job的Reduce个数

set mapreduce.job.reduces = 15;

**3）reduce个数并不是越多越好**

（1）过多的启动和初始化reduce也会消耗时间和资源；

（2）另外，有多少个reduce，就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；

在设置reduce个数的时候也需要考虑这两个原则：处理大数据量利用合适的reduce数；使单个reduce任务处理数据量大小要合适；

## 1.5 并行执行

Hive会将一个查询转化成一个或者多个阶段。这样的阶段可以是MapReduce阶段、抽样阶段、合并阶段、limit阶段。或者Hive执行过程中可能需要的其他阶段。默认情况下，Hive一次只会执行一个阶段。不过，某个特定的job可能包含众多的阶段，而这些阶段可能并非完全互相依赖的，也就是说有些阶段是可以并行执行的，这样可能使得整个job的执行时间缩短。不过，如果有更多的阶段可以并行执行，那么job可能就越快完成。

通过设置参数hive.exec.parallel值为true，就可以开启并发执行。不过，在共享集群中，需要注意下，如果job中并行阶段增多，那么集群利用率就会增加。

set hive.exec.parallel=true;   //打开任务并行执行，默认为false

set hive.exec.parallel.thread.number=16; //同一个sql允许最大并行度，默认为8。

当然，得是在系统资源比较空闲的时候才有优势，否则，没资源，并行也起不来。

## 1.6 严格模式

Hive可以通过设置防止一些危险操作：

**1）分区表不使用分区过滤**

将hive.strict.checks.no.partition.filter设置为true时，对于分区表，除非where语句中含有分区字段过滤条件来限制范围，否则不允许执行。换句话说，就是用户不允许扫描所有分区。进行这个限制的原因是，通常分区表都拥有非常大的数据集，而且数据增加迅速。没有进行分区限制的查询可能会消耗令人不可接受的巨大资源来处理这个表。  
**2）使用order by没有limit过滤**

将hive.strict.checks.orderby.no.limit设置为true时，对于使用了order by语句的查询，要求必须使用limit语句。因为order by为了执行排序过程会将所有的结果数据分发到同一个Reducer中进行处理，强制要求用户增加这个LIMIT语句可以防止Reducer额外执行很长一段时间。

**3）笛卡尔积**

将hive.strict.checks.cartesian.product设置为true时，会限制笛卡尔积的查询。对关系型数据库非常了解的用户可能期望在 执行JOIN查询的时候不使用ON语句而是使用where语句，这样关系数据库的执行优化器就可以高效地将WHERE语句转化成那个ON语句。不幸的是，Hive并不会执行这种优化，因此，如果表足够大，那么这个查询就会出现不可控的情况。

## 1.7 JVM重用

1. 开启uber模式，实现jvm重用。默认情况下，每个Task任务都需要启动一个jvm来运行，如果Task任务计算的数据量很小，我们可以让同一个Job的多个Task运行在一个Jvm中，不必为每个Task都开启一个Jvm.

开启uber模式，在mapred-site.xml中添加如下配置

<!-- 开启uber模式 -->

<property>

<name>mapreduce.job.ubertask.enable</name>

<value>true</value>

</property>

<!-- uber模式中最大的mapTask数量，可向下修改 -->

<property>

<name>mapreduce.job.ubertask.maxmaps</name>

<value>9</value>

</property>

<!-- uber模式中最大的reduce数量，可向下修改 -->

<property>

<name>mapreduce.job.ubertask.maxreduces</name>

<value>1</value>

</property>

<!-- uber模式中最大的输入数据量，默认使用dfs.blocksize 的值，可向下修改 -->

<property>

<name>mapreduce.job.ubertask.maxbytes</name>

<value></value>

</property>

## 1.8 压缩

要在Hadoop中启用压缩，可以配置如下参数（mapred-site.xml文件中）：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 默认值 | 阶段 | 建议 |
| io.compression.codecs  （在core-site.xml中配置） | org.apache.hadoop.io.compress.DefaultCodec, org.apache.hadoop.io.compress.GzipCodec, org.apache.hadoop.io.compress.BZip2Codec,  org.apache.hadoop.io.compress.Lz4Codec | 输入压缩 | Hadoop使用文件扩展名判断是否支持某种编解码器 |
| mapreduce.map.output.compress | false | mapper输出 | 这个参数设为true启用压缩 |
| mapreduce.map.output.compress.codec | org.apache.hadoop.io.compress.DefaultCodec | mapper输出 | 使用LZO、LZ4或snappy编解码器在此阶段压缩数据 |
| mapreduce.output.fileoutputformat.compress | false | reducer输出 | 这个参数设为true启用压缩 |
| mapreduce.output.fileoutputformat.compress.codec | org.apache.hadoop.io.compress. DefaultCodec | reducer输出 | 使用标准工具或者编解码器，如gzip和bzip2 |
| mapreduce.output.fileoutputformat.compress.type | RECORD | reducer输出 | SequenceFile输出使用的压缩类型：NONE和BLOCK |

## 1.9 执行计划（Explain）

**1）基本语法**

EXPLAIN [EXTENDED | DEPENDENCY | AUTHORIZATION] query

**2）案例实操**

（1）查看下面这条语句的执行计划

hive (default)> explain select \* from emp;

hive (default)> explain select deptno, avg(sal) avg\_sal from emp group by deptno;

（2）查看详细执行计划

hive (default)> explain extended select \* from emp;

hive (default)> explain extended select deptno, avg(sal) avg\_sal from emp group by deptno;