Hive 高级应用

目录

1、Hive shell 操作

1.1、Hive 命令行

这是 hive 支持的一些命令：

Command	Description
quit	Use quit or exit to leave the interactive shell.

set key=value Use this to set value of particular configuration variable. One thing to note here is that if you misspell the variable name, cli will not show an error.
set This will print a list of configuration variables that are overridden by user or hive.
set -v This will print all hadoop and hive configuration variables.
add FILE [file] [file]* Adds a file to the list of resources
add jar jarname
list FILE list all the files added to the distributed cache
list FILE [file]* Check if given resources are already added to distributed cache
! [cmd] Executes a shell command from the hive shell
dfs [dfs cmd] Executes a dfs command from the hive shell
[query] Executes a hive query and prints results to standard out
source FILE Used to execute a script file inside the CLI.

语法结构：

hive [-hiveconf x=y]* [<-i filename>]* [<-f filename>|<-e query-string>] [-S]

说明：

1、-i 从文件初始化 HQL

2、-e 从命令行执行指定的 HQL

3、-f 执行 HQL 脚本

4、-v 输出执行的 HQL 语句到控制台

5、-p <port> connect to Hive Server on port number

6、-hiveconf x=y（Use this to set hive/hadoop configuration variables）

7、-S：表示以不打印日志的形式执行命名操作

示例：

1、 运行一个查询

```
15,44
[root@hadoop02 hivedata]# hive -e 'select * from mydb.tablea'

Logging initialized using configuration in jar:file:/root/apps/apach
j.properties
OK
1       huangbo
2       xuzheng
4       wangbaoqiang
6       huangxiaoming
7       fengjie
10      liudehua
Time taken: 2.016 seconds, Fetched: 6 row(s)
[root@hadoop02 hivedata]#
```

2、 运行一个文件

```
[root@hadoop02 hivedata]# cat hive.hql
select * from mydb.tablea;
[root@hadoop02 hivedata]# hive -f hive.hql

Logging initialized using configuration in jar:f
j.properties
OK
1       huangbo
2       xuzheng
4       wangbaoqiang
6       huangxiaoming
7       fengjie
10      liudehua
Time taken: 1.848 seconds, Fetched: 6 row(s)
[root@hadoop02 hivedata]#
```

3、 运行参数文件

从配置文件启动 hive，并加载配置文件当中的配置参数

```
[root@hadoop02 hivedata]# cat inithive.conf
set mapred.reduce.tasks=4;
[root@hadoop02 hivedata]# hive -i inithive.conf

Logging initialized using configuration in jar:file:/
j.properties
hive> set mapred.reduce tasks;
mapred.reduce tasks is undefined
hive> set mapred.reduce.tasks;
mapred.reduce.tasks=4
hive>
```

1.2、Hive 参数配置方式

Hive 参数大全：
https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties

开发 Hive 应用时，不可避免地需要设定 Hive 的参数。设定 Hive 的参数可以调优 HQL 代码的执行效率，或帮助定位问题。然而实践中经常遇到的一个问题是，为什么设定的参数没有起作用？这通常是错误的设定方式导致的

对于一般参数，有以下三种设定方式：

配置文件 （全局有效）

命令行参数（对 hive 启动实例有效）

参数声明 （对 hive 的连接 session 有效）

配置文件：Hive 的配置文件包括

用户自定义配置文件：$HIVE_CONF_DIR/hive-site.xml

默认配置文件：$HIVE_CONF_DIR/hive-default.xml

用户自定义配置会覆盖默认配置。

另外，Hive 也会读入 Hadoop 的配置，因为 Hive 是作为 Hadoop 的客户端启动的，Hive 的配置会覆盖 Hadoop 的配置。

配置文件的设定对本机启动的所有 Hive 进程都有效。

命令行参数：启动 Hive（客户端或 Server 方式）时，可以在命令行添加-hiveconf param=value 来设定参数，例如：
bin/hive -hiveconf hive.root.logger=INFO,console
这一设定对本次启动的 session（对于 server 方式启动，则是所有请求的 session）有效。

参数声明：可以在 HQL 中使用 SET 关键字设定参数，例如：
set mapred.reduce.tasks = 10;
set mapreduce.job.reduces = 10;
这一设定的作用域也是 session 级的。

set hive.exec.reducers.bytes.per.reducer=<number> 每个 reduce task 的平均负载数据量
Hive 会估算总数据量，然后用该值除以上述参数值，就能得出需要运行的 reduceTask 数

set hive.exec.reducers.max=<number> 设置 reduce task 数量的上限

set mapreduce.job.reduces=<number> 指定固定的 reduce task 数量
但是，这个参数在必要时<业务逻辑决定只能用一个 reduce task> hive 会忽略，比如在设置了 set mapreduce.job.reduces = 3，但是 HQL 语句当中使用了 order by 的话，那么就会忽略该参数的设置

上述三种设定方式的优先级依次递增。即**参数声明覆盖命令行参数，命令行参数覆盖配置文件设定**。注意某些系统级的参数，例如 log4j 相关的设定，必须用前两种方式设定，因为那些参数的读取在 session 建立以前已经完成了。

2、数据倾斜

1、什么是数据倾斜？
由于数据分布不均匀，造成数据大量的集中到一点，造成数据热点

2、Hadoop 框架的特性
A、不怕数据大，怕数据倾斜
B、Jobs 数比较多的作业运行效率相对比较低，如子查询比较多
C、 sum,count,max,min 等聚集函数，通常不会有数据倾斜问题

3、容易数据倾斜情况

操作：

关键词	情形	后果
Join	其中一个表较小， 但是 key 集中	分发到某一个或几个 Reduce 上 的数据远高于平均值
	大表与大表，但是分桶的判 断字段 0 值或空值过多	这些空值都由一个 reduce 处理， 非常慢
group by	group by 维度过小， 某值的数量过多	处理某值的 reduce 非常耗时
Count Distinct	某特殊值过多	处理此特殊值的 reduce 耗时

A、group by 不和聚集函数搭配使用的时候

B、count(distinct)，在数据量大的情况下，容易数据倾斜，因为 count(distinct)是按 group by 字段分组，按 distinct 字段排序

C、 小表关联超大表 join

4、产生数据倾斜的原因：

A：key 分布不均匀

B：业务数据本身的特性

C：建表考虑不周全

D：某些 HQL 语句本身就存在数据倾斜

5、主要表现：

任务进度长时间维持在 99%或者 100%的附近，查看任务监控页面，发现只有少量 reduce 子任务未完成，因为其处理的数据量和其他的 reduce 差异过大。

单一 reduce 处理的记录数和平均记录数相差太大，通常达到好几倍之多，最长时间远大于平均时长。

6、业务场景

A：空值产生的数据倾斜

场景说明：在日志中，常会有信息丢失的问题，比如日志中的 user_id，如果取其中的 user_id 和用户表中的 user_id 相关联，就会碰到数据倾斜的问题。

解决方案 1：user_id 为空的不参与关联

```
select * from log a join user b on a.user_id is not null and a.user_id = b.user_id
union all
select * from log c where c.user_id is null;
```

解决方案 2：赋予空值新的 key 值

```
select * from log a left outer join user b on
case when a.user_id is null then concat('hive',rand()) else a.user_id end = b.user_id
```

总结：方法 2 比方法 1 效率更好，不但 IO 少了，而且作业数也少了，方案 1 中，log 表

读了两次，jobs 肯定是 2，而方案 2 是 1。这个优化适合无效 id（比如-99，”，null）产生的数据倾斜，**把空值的 key 变成一个字符串加上一个随机数**，就能把造成数据倾斜的数据分到不同的 reduce 上解决数据倾斜的问题，

改变之处：使本身为 null 的所有记录不会拥挤在同一个 reduceTask 了，会由于有替代的随机字符串值，而分散到了多个 reduceTask 中了，由于 null 值关联不上，处理后并不影响最终结果。

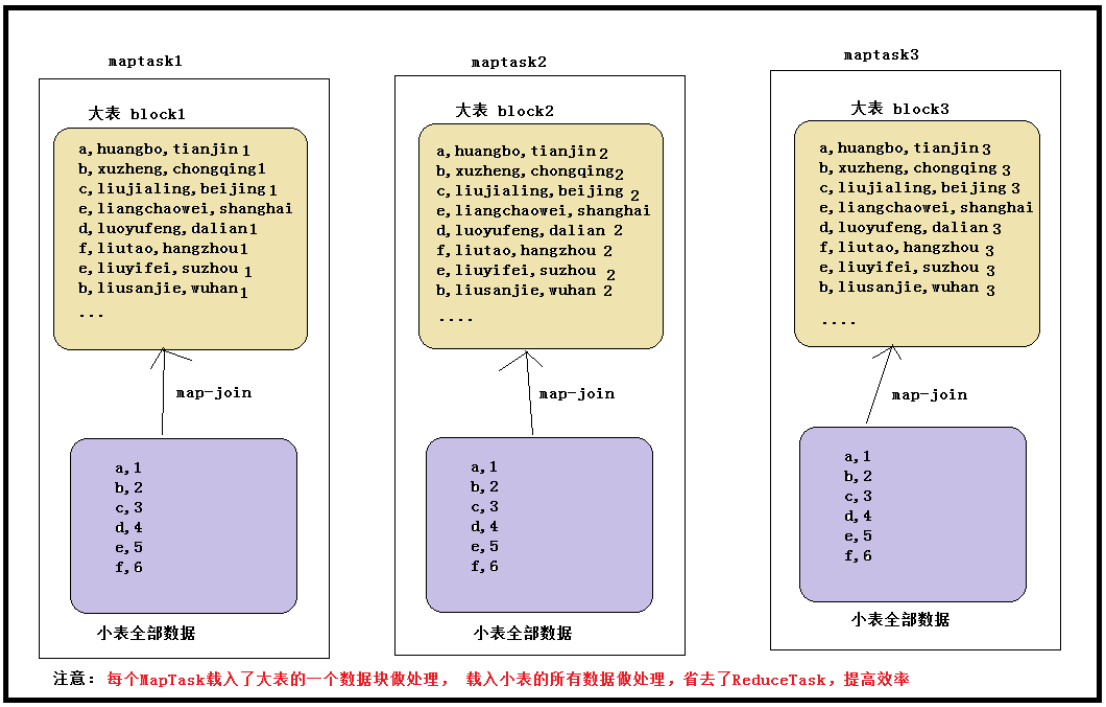
B：不同数据类型关联产生数据倾斜

场景说明：用户表中 user_id 字段为 int，log 表中 user_id 为既有 string 也有 int 的类型，当按照两个表的 user_id 进行 join 操作的时候，默认的 hash 操作会按照 int 类型的 id 进行分配，这样就会导致所有的 string 类型的 id 就被分到同一个 reducer 当中

解决方案：把数字类型 id 转换成 string 类型的 id
select * from user a left outer join log b on b.user_id = cast(a.user_id as string)

C：大小表关联查询

注意：使用 map join 解决小表关联大表造成的数据倾斜问题。这个方法使用的频率很高。
map join 概念：将其中做连接的小表（全量数据）分发到所有 MapTask 端进行 Join，从而避免了 reduceTask，前提要求是内存足以装下该全量数据



以大表 a 和小表 b 为例，所有的 maptask 节点都装载小表 b 的所有数据，然后大表 a 的一个数据块数据比如说是 a1 去跟 b 全量数据做链接，就**省去了 reduce** 做汇总的过程。所以相对来说，在内存允许的条件下使用 map join 比直接使用 MapReduce 效率还高些，当然这只限于做 join 查询的时候。

在 hive 中，直接提供了能够在 HQL 语句指定该次查询使用 map join，map join 的用法是在查询/子查询的 SELECT 关键字后面添加/*+ MAPJOIN(tablelist) */提示优化器转化为 map join（早期的 Hive 版本的优化器是不能自动优化 map join 的）。其中 tablelist 可以是一个表，或以逗号连接的表的列表。tablelist 中的表将会读入内存，通常应该是将小表写在这里。

MapJoin 具体用法：

select /* +mapjoin(a) */ a.id aid, name, age from a join b on a.id = b.id;

select /* +mapjoin(movies) */ a.title, b.rating from movies a join ratings b on a.movieid = b.movieid;

在 hive0.11 版本以后会自动开启 map join 优化，由两个参数控制：

set hive.auto.convert.join=true;

//设置 MapJoin 优化自动开启

set hive.mapjoin.smalltable.filesize=25000000

//设置小表不超过多大时开启 mapjoin 优化

如果是大大表关联呢？那就大事化小，小事化了。**把大表切分成小表，然后分别 map join**

那么如果小表不大不小，那该如何处理呢？？？

使用 map join 解决小表(记录数少)关联大表的数据倾斜问题，这个方法使用的频率非常高，但如果小表很大，大到 map join 会出现 bug 或异常，这时就需要特别的处理

举一例：日志表和用户表做链接

select * from log a left outer join users b on a.user_id = b.user_id;

users 表有 600w+的记录，把 users 分发到所有的 map 上也是个不小的开销，而且 map join 不支持这么大的小表。如果用普通的 join，又会碰到数据倾斜的问题。

改进方案：

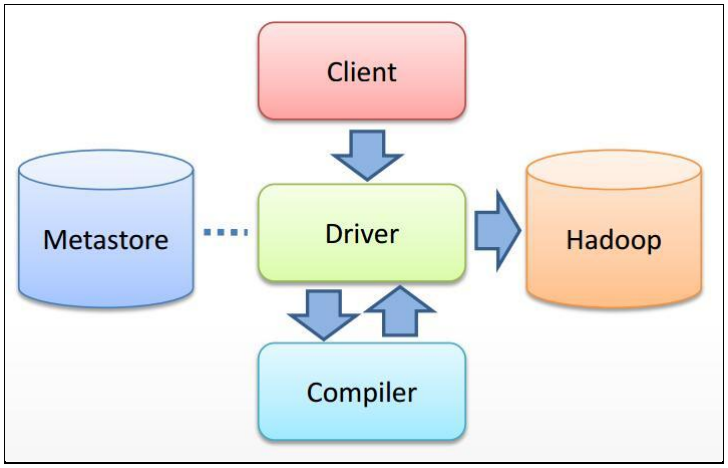
```
select /*+mapjoin(x)*/* from log a
    left outer join (
        select /*+mapjoin(c)*/ d.*
            from ( select distinct user_id from log ) c join users d on c.user_id = d.user_id
    ) x
on a.user_id = b.user_id;
```

假如，log 里 user_id 有上百万个，这就又回到原来 map join 问题。所幸，每日的会员 uv 不会太多，有交易的会员不会太多，有点击的会员不会太多，有佣金的会员不会太多等等。所以这个方法能解决很多场景下的数据倾斜问题

4、Hive 执行过程实例分析

4.1、Hive 执行过程概述

- Hive 将 HQL 转换成一组操作符（Operator），比如 GroupByOperator, JoinOperator 等
- 操作符 Operator 是 Hive 的最小处理单元
- 每个操作符代表一个 HDFS 操作或者 MapReduce 作业
- Hive 通过 ExecMapper 和 ExecReducer 执行 MapReduce 程序，执行模式有本地模式和分布式两种模式



Hive 操作符列表：

操作符	描述
TableScanOperator	扫描hive表数据
ReduceSinkOperator	创建将发送到Reducer端的<Key,Value>对
JoinOperator	Join两份数据
SelectOperator	选择输出列
FileSinkOperator	建立结果数据,输出至文件
FilterOperator	过滤输入数据
GroupByOperator	Group By语句
MapJoinOperator	/*+ mapjoin(t) */
LimitOperator	Limit语句
UnionOperator	Union语句

Hive 编译器的工作职责：

- Parser：将 HQL 语句转换成抽象语法树（AST：Abstract Syntax Tree）
- Semantic Analyzer：将抽象语法树转换成查询块
- Logic Plan Generator：将查询块转换成逻辑查询计划

- Logic Optimizer：重写逻辑查询计划，优化逻辑执行计划
- Physical Plan Gernerator：将逻辑计划转化成物理计划（MapReduce Jobs）
- Physical Optimizer：选择最佳的 Join 策略，优化物理执行计划

优化器类型：

名称	作用
② SimpleFetchOptimizer	优化没有 GroupBy 表达式的聚合查询
② MapJoinProcessor	MapJoin，需要 SQL 中提供 hint，0.11 版本已不用
② BucketMapJoinOptimizer	BucketMapJoin
② GroupByOptimizer	Map 端聚合
① ReduceSinkDeDuplication	合并线性的 OperatorTree 中 partition/sort key 相同的 reduce
① PredicatePushDown	谓词前置
① CorrelationOptimizer	利用查询中的相关性，合并有相关性的 Job，HIVE-2206
ColumnPruner	字段剪枝

上表中带①符号的，优化目的都是尽量将任务合并到一个 Job 中，以减少 Job 数量，带②的优化目的是尽量减少 shuffle 数据量

4.2、Join

对于 join 操作：

```
SELECT pv.pageid, u.age FROM page_view pv JOIN user u ON pv.userid = u.userid;
```

实现过程：

Map：1、以 JOIN ON 条件中的列作为 Key，如果有多个列，则 Key 是这些列的组合

2、以 JOIN 之后所关心的列作为 Value，当有多个列时，Value 是这些列的组合。在 Value 中还会包含表的 Tag 信息，用于标明此 Value 对应于哪个表

3、按照 Key 进行排序

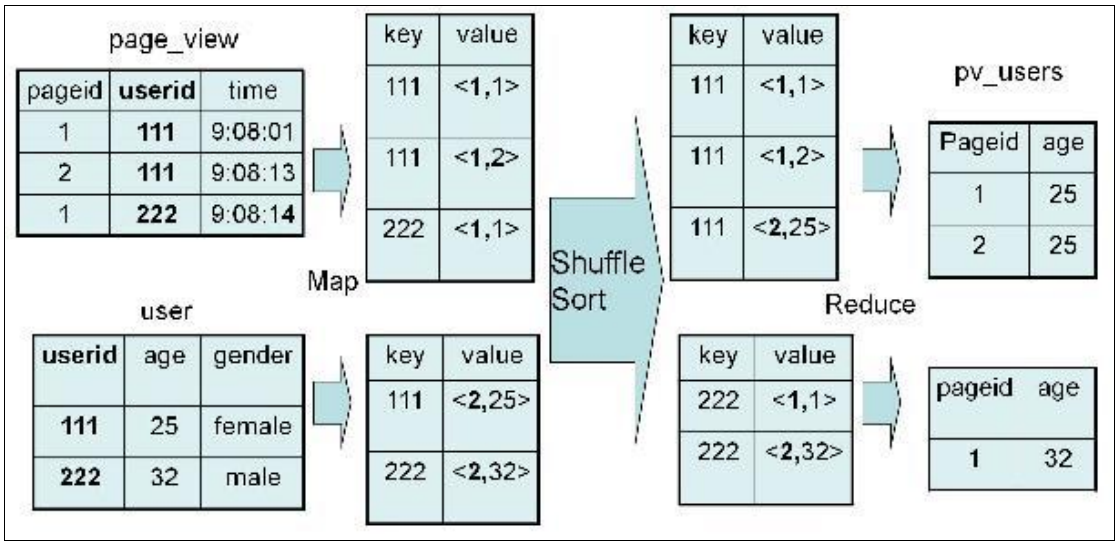
Shuffle：

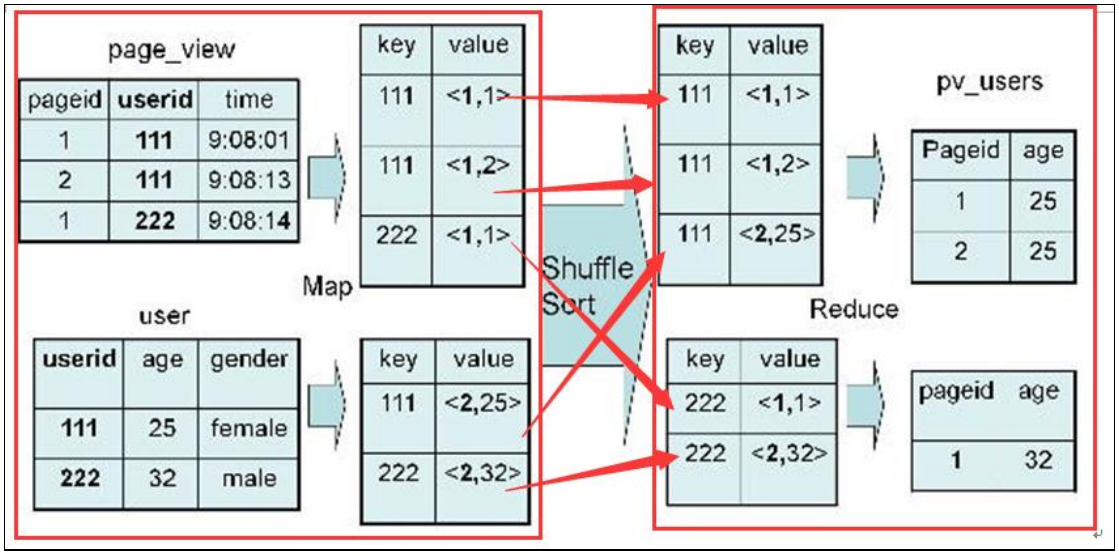
1、根据 Key 的值进行 Hash，并将 Key/Value 对按照 Hash 值推至不同对 Reduce 中

Reduce：

1、 Reducer 根据 Key 值进行 Join 操作，并且通过 Tag 来识别不同的表中的数据

具体实现过程：

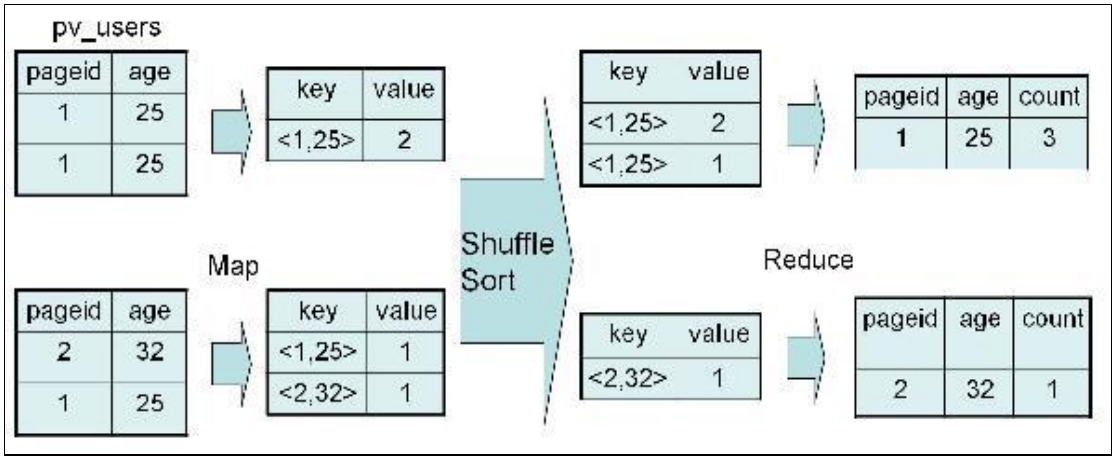


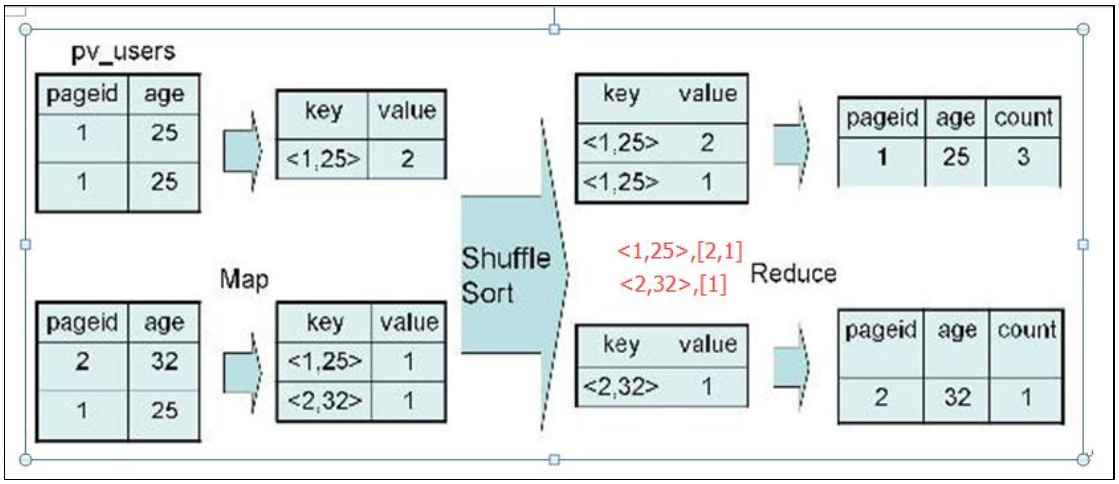


4.3、Group By

对于 group by:

```
SELECT pageid, age, count(1) FROM pv_users GROUP BY pageid, age;
```





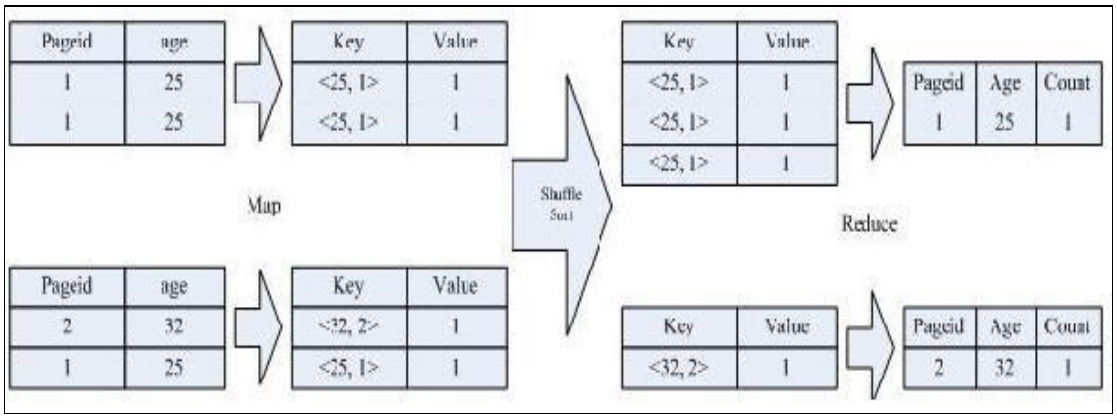
4.4、Distinct

对于 distinct：

```
SELECT age, count(distinct pageid) FROM pv_users GROUP BY age;
```

按照 age 分组，然后统计每个分组里面的不重复的 pageid 有多少个

实现过程：



详细过程解释：该 SQL 语句会按照 age 和 pageid 预先分组，进行 distinct 操作。然后会再按照 age 进行分组，再进行一次 distinct 操作

5、Hive 优化策略

5.1、Hadoop 框架计算特性

1、数据量大不是问题，数据倾斜是个问题

2、jobs 数比较多的作业运行效率相对比较低，比如即使有几百行的表，如果多次关联多次汇总，产生十几个 jobs，耗时很长。原因是 map reduce 作业初始化的时间是比较长的

3、sum,count,max,min 等 UDAF，不怕数据倾斜问题，hadoop 在 map 端的汇总合并优化，使

数据倾斜不成问题

4、count(distinct userid)，在数据量大的情况下，效率较低，如果是多 count(distinct userid,month)效率更低，因为 count(distinct)是按 group by 字段分组，按 distinct 字段排序，一般这种分布方式是很倾斜的，比如 PV 数据，淘宝一天 30 亿的 pv，如果按性别分组，分配 2 个 reduce，每个 reduce 期望处理 15 亿数据，但现实必定是男少女多

5.2、优化常用手段

1、好的模型设计事半功倍
2、解决数据倾斜问题
3、减少 job 数
4、设置合理的 MapReduce 的 task 数，能有效提升性能。(比如，10w+级别的计算，用 160 个 reduce，那是相当的浪费，1 个足够)
5、了解数据分布，自己动手解决数据倾斜问题是个不错的选择。这是通用的算法优化，但算法优化有时不能适应特定业务背景，开发人员了解业务，了解数据，可以通过业务逻辑精确有效的解决数据倾斜问题
6、数据量较大的情况下，慎用 count(distinct)，group by 容易产生倾斜问题
7、对小文件进行合并，是行之有效的提高调度效率的方法，假如所有的作业设置合理的文件数，对云梯的整体调度效率也会产生积极的正向影响
8、优化时把握整体，单个作业最优不如整体最优

5.3、排序选择

cluster by：对同一字段分桶并排序，不能和 sort by 连用
distribute by + sort by：分桶，保证同一字段值只存在一个结果文件当中，结合 sort by 保证每个 reduceTask 结果有序
sort by：单机排序，单个 reduce 结果有序
order by：全局排序，缺陷是只能使用一个 reduce

一定要区分这四种排序的使用方式和适用场景

5.4、怎样做笛卡尔积

当 Hive 设定为严格模式（hive.mapred.mode=strict）时，不允许在 HQL 语句中出现笛卡尔积，这实际说明了 Hive 对笛卡尔积支持较弱。因为找不到 Join key，Hive 只能使用 1 个 reducer 来完成笛卡尔积。

当然也可以用上面说的 limit 的办法来减少某个表参与 join 的数据量，但对于需要笛卡尔积语义的需求来说，经常是一个大表和一个小表的 Join 操作，结果仍然很大（以至于无法用单机处理），这时 MapJoin 才是最好的解决办法。MapJoin，顾名思义，会在 Map 端完成 Join 操作。这需要将 Join 操作的一个或多个表完全读入内存。

PS：MapJoin 在子查询中可能出现未知 BUG。在大表和小表做笛卡尔积时，规避笛卡尔积的方法是，给 Join 添加一个 Join key，**原理很简单：将小表扩充一列 join key，并将小表的条目复制数倍，join key 各不相同；将大表扩充一列 join key 为随机数。**

精髓就在于复制几倍，最后就有几个 reduce 来做，而且大表的数据是前面小表扩张 key 值范围里面随机出来的，所以复制了几倍 n，就相当于这个随机范围就有多大 n，那么相应的，大表的数据就被随机的分为了 n 份。并且最后处理所用的 reduce 数量也是 n，而且也不会出现数据倾斜。

5.5、怎样写 in/exists 语句

虽然经过测验，hive1.2.1 也支持 in/exists 操作，但还是推荐使用 hive 的一个高效替代方案：left semi join

比如说：

select a.id, a.name from a where a.id in (select b.id from b);

select a.id, a.name from a where exists (select id from b where a.id = b.id);

应该转换成：

select a.id, a.name from a left semi join b on a.id = b.id;

5.6、设置合理的 maptask 数量

Map 数过大

Map 阶段输出文件太小，产生大量小文件

初始化和创建 Map 的开销很大

Map 数太小

文件处理或查询并发度小，Job 执行时间过长

大量作业时，容易堵塞集群

在 MapReduce 的编程案例中，我们得知，一个 MR Job 的 MapTask 数量是由输入分片 InputSplit 决定的。而输入分片是由 FileInputFormat.getSplit()决定的。一个输入分片对应一个 MapTask，而输入分片是由三个参数决定的：

dfs.block.size	128M	HDFS 默认数据块大小
mapreduce.input.fileinputformat.split.minsize	1	最小分片大小
mapreduce.input.fileinputformat.split.maxsize	Long.MAX_VALUE	最大分片大小

输入分片大小的计算是这么计算出来的：

long splitSize = Math.max(minSize, Math.min(maxSize, blockSize))

默认情况下，输入分片大小和 HDFS 集群默认数据块大小一致，也就是默认一个数据块，启用一个 MapTask 进行处理，这样做的好处是避免了服务器节点之间的数据传输，提高 job 处理效率

两种经典的控制 MapTask 的个数方案：减少 MapTask 数或者增加 MapTask 数

1、 减少 MapTask 数是通过合并小文件来实现，这一点主要是针对数据源
2、 增加 MapTask 数可以通过控制上一个 job 的 reduceTask 个数

因为 Hive 语句最终要转换为一系列的 MapReduce Job 的，而每一个 MapReduce Job 是由一系列的 MapTask 和 ReduceTask 组成的，默认情况下， MapReduce 中一个 MapTask 或者一个 ReduceTask 就会启动一个 JVM 进程，一个 Task 执行完毕后， JVM 进程就退出。这样如果任务花费时间很短，又要多次启动 JVM 的情况下，JVM 的启动时间会变成一个比较大的消耗，这个时候，就可以通过重用 JVM 来解决：

```
set mapred.job.reuse.jvm.num.tasks=5
```

5.7、小文件合并

文件数目过多，会给 HDFS 带来压力，并且会影响处理效率，可以通过合并 Map 和 Reduce 的结果文件来消除这样的影响：

```
set hive.merge.mapfiles = true              ##在 map only 的任务结束时合并小文件
set hive.merge.mapredfiles = false          ## true 时在 MapReduce 的任务结束时合并小文件
set hive.merge.size.per.task = 256*1000*1000                ##合并文件的大小
set mapred.max.split.size=256000000;                        ##每个 Map 最大分割大小
set mapred.min.split.size.per.node=1;                       ##一个节点上 split 的最少值
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
                                                            ##执行 Map 前进行小文件合并
```

5.8、设置合理的 reduceTask 的数量

Hadoop MapReduce 程序中，reducer 个数的设定极大影响执行效率，这使得 Hive 怎样决定 reducer 个数成为一个关键问题。遗憾的是 Hive 的估计机制很弱，不指定 reducer 个数的情况下，Hive 会猜测确定一个 reducer 个数，基于以下两个设定：

1、hive.exec.reducers.bytes.per.reducer（默认为 256000000）
2、hive.exec.reducers.max（默认为 1009）
3、mapreduce.job.reduces=-1（设置一个常量 reducetask 数量）

计算 reducer 数的公式很简单：
N=min(参数 2，总输入数据量/参数 1)
通常情况下，有必要手动指定 reducer 个数。考虑到 map 阶段的输出数据量通常会比输入有大幅减少，因此即使不设定 reducer 个数，重设参数 2 还是必要的。

依据 Hadoop 的经验，可以将参数 2 设定为 0.95*(集群中 datanode 个数)。

5.9、合并 MapReduce 操作

Multi-group by 是 Hive 的一个非常好的特性，它使得 Hive 中利用中间结果变得非常方便。例如：

```
FROM (SELECT a.status, b.school, b.gender FROM status_updates a JOIN profiles b ON (a.userid =
b.userid and a.ds='2009-03-20' ) ) subq1
INSERT OVERWRITE TABLE gender_summary PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1) GROUP BY subq1.school
```

上述查询语句使用了 multi-group by 特性连续 group by 了 2 次数据，使用不同的 group by key。这一特性可以减少一次 MapReduce 操作

5.10、合理利用分桶：Bucketing 和 Sampling

Bucket 是指将数据以指定列的值为 key 进行 hash，hash 到指定数目的桶中。这样就可以支持高效采样了。如下例就是以 userid 这一列为 bucket 的依据，共设置 32 个 buckets

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                       page_url STRING, referrer_url STRING,
                       ip STRING COMMENT 'IP Address of the User')
    COMMENT 'This is the page view table'
    PARTITIONED BY(dt STRING, country STRING)
    CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
    ROW FORMAT DELIMITED
            FIELDS TERMINATED BY '1'
            COLLECTION ITEMS TERMINATED BY '2'
            MAP KEYS TERMINATED BY '3'
    STORED AS SEQUENCEFILE;
```

通常情况下，Sampling 在全体数据上进行采样，这样效率自然就低，它要去访问所有数据。而如果一个表已经对某一列制作了 bucket，就可以采样所有桶中指定序号的某个桶，这就减少了访问量。

如下例所示就是采样了 page_view 中 32 个桶中的第三个桶的全部数据：

```
SELECT * FROM page_view TABLESAMPLE(BUCKET 3 OUT OF 32);
```

如下例所示就是采样了 page_view 中 32 个桶中的第三个桶的一半数据：

```
SELECT * FROM page_view TABLESAMPLE(BUCKET 3 OUT OF 64);
```

详细说明：http://blog.csdn.net/zhongqi2513/article/details/74612701

5.11、合理利用分区：Partition

Partition 就是分区。分区通过在创建表时启用 partitioned by 实现，用来 partition 的维度并不

是实际数据的某一列，具体分区的标志是由插入内容时给定的。当要查询某一分区的内容时可以采用 where 语句，形似 where tablename.partition_column = a 来实现。

创建含分区的表

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User')
PARTITIONED BY(date STRING, country STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '1'
STORED AS TEXTFILE;
```

载入内容，并指定分区标志

```
load data local inpath '/home/hadoop/pv_2008-06-08_us.txt' into table page_view partition(date='2008-06-08', country='US');
```

查询指定标志的分区内容

```
SELECT page_views.* FROM page_views
    WHERE page_views.date >= '2008-03-01' AND page_views.date <= '2008-03-31' AND page_views.referrer_url like '%xyz.com';
```

5.12、Join 优化

总体原则：

1、 优先过滤后再进行 Join 操作，最大限度的减少参与 join 的数据量

2、 小表 join 大表，最好启动 mapjoin

3、 Join on 的条件相同的话，最好放入同一个 job，并且 join 表的排列顺序从小到大

在使用写有 Join 操作的查询语句时有一条原则：应该将条目少的表/子查询放在 Join 操作符的左边。原因是在 Join 操作的 Reduce 阶段，位于 Join 操作符左边的表的内容会被加载进内存，将条目少的表放在左边，可以有效减少发生 OOM 错误的几率。对于一条语句中有多个 Join 的情况，如果 Join 的条件相同，比如查询

```
INSERT OVERWRITE TABLE pv_users
 SELECT pv.pageid, u.age FROM page_view p
 JOIN user u ON (pv.userid = u.userid)
 JOIN newuser x ON (u.userid = x.userid);
```

如果 Join 的 key 相同，不管有多少个表，都会则会合并为一个 Map-Reduce 任务，而不是”n”个，在做 OUTER JOIN 的时候也是一样

如果 join 的条件不相同，比如：

```
INSERT OVERWRITE TABLE pv_users
    SELECT pv.pageid, u.age FROM page_view p
    JOIN user u ON (pv.userid = u.userid)
    JOIN newuser x on (u.age = x.age);
```

Map-Reduce 的任务数目和 Join 操作的数目是对应的，上述查询和以下查询是等价的

```
// 先 page_view 表和 user 表做链接
INSERT OVERWRITE TABLE tmptable
    SELECT * FROM page_view p JOIN user u ON (pv.userid = u.userid);

// 然后结果表 temptable 和 newuser 表做链接
INSERT OVERWRITE TABLE pv_users
    SELECT x.pageid, x.age FROM tmptable x JOIN newuser y ON (x.age = y.age);
```

在编写 Join 查询语句时，如果确定是由于 join 出现的数据倾斜，那么请做如下设置：

```
set hive.skewjoin.key=100000;        // 这个是 join 的键对应的记录条数超过这个值则会进行分拆，值根据具体数据量设置
set hive.optimize.skewjoin=true;     // 如果是 join 过程出现倾斜应该设置为 true
```

5.13、Group By 优化

Map 端部分聚合：

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。

MapReduce 的 combiner 组件

参数包括：

set hive.map.aggr = true 是否在 Map 端进行聚合，默认为 True

set hive.groupby.mapaggr.checkinterval = 100000 在 Map 端进行聚合操作的条目数目

当使用 Group By 有数据倾斜的时候进行负载均衡：

set hive.groupby.skewindata = true

当 sql 语句使用 groupby 时数据出现倾斜时，如果该变量设置为 true，那么 Hive 会自动进行负载均衡。**策略就是把 MR 任务拆分成两个：第一个先做预汇总，第二个再做最终汇总**

在 MR 的第一个阶段中，Map 的输出结果集合会缓存到 maptaks 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 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；第二个阶段 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。

5.14、合理利用文件存储格式

创建表时，尽量使用 orc、parquet 这些列式存储格式，因为列式存储的表，每一列的数据在物理上是存储在一起的，Hive 查询时会只遍历需要列数据，大大减少处理的数据量。

5.15、本地模式执行 MapReduce

Hive 在集群上查询时，默认是在集群上 N 台机器上运行， 需要多个机器进行协调运行，这

个方式很好地解决了大数据量的查询问题。但是当 Hive 查询处理的数据量比较小时，其实没有必要启动分布式模式去执行，因为以分布式方式执行就涉及到跨网络传输、多节点协调等，并且消耗资源。这个时间可以只使用本地模式来执行 mapreduce job，只在一台机器上执行，速度会很快。启动本地模式涉及到三个参数：

参数名	默认值	备注
hive.exec.mode.local.auto	false	让 hive 决定是否在本地模式自动运行
hive.exec.mode.local.auto.input.files.max	4	不启用本地模式的 task 最大个数
hive.exec.mode.local.auto.inputbytes.max	128M	不启动本地模式的最大输入文件大小

set hive.exec.mode.local.auto=true 是打开 hive 自动判断是否启动本地模式的开关，但是只是打开这个参数并不能保证启动本地模式，要当 map 任务数不超过 hive.exec.mode.local.auto.input.files.max 的个数并且 map 输入文件大小不超过 hive.exec.mode.local.auto.inputbytes.max 所指定的大小时，才能启动本地模式。

5.16、并行化处理

一个 hive sql 语句可能会转为多个 mapreduce Job，每一个 job 就是一个 stage，这些 job 顺序执行，这个在 cli 的运行日志中也可以看到。但是有时候这些任务之间并不是是相互依赖的，如果集群资源允许的话，可以让多个并不相互依赖 stage 并发执行，这样就节约了时间，提高了执行速度，但是如果集群资源匮乏时，启用并行化反倒是会导致各个 job 相互抢占资源而导致整体执行性能的下降。启用并行化：

```
set hive.exec.parallel=true;
set hive.exec.parallel.thread.number=8;       //同一个 sql 允许并行任务的最大线程数
```

5.17、设置压缩存储

1、压缩的原因

Hive 最终是转为 MapReduce 程序来执行的，而 MapReduce 的性能瓶颈在于网络 IO 和磁盘 IO，要解决性能瓶颈，最主要的是减少数据量，对数据进行压缩是个好的方式。压缩虽然是减少了数据量，但是压缩过程要消耗 CPU 的，但是在 Hadoop 中， 往往性能瓶颈不在于 CPU，CPU 压力并不大，所以压缩充分利用了比较空闲的 CPU

2、常用压缩方法对比

压缩格式	是否可拆分	是否自带	压缩率	速度	是否 hadoop 自带
gzip	否	是	很高	比较快	是
lzo	是	是	比较高	很快	否，要安装
snappy	否	是	比较高	很快	否，要安装
bzip2	是	否	最高	慢	是

各个压缩方式所对应的 Class 类：

压缩格式	类
Zlib	org.apache.hadoop.io.compress.DefaultCodec
Gzip	org.apache.hadoop.io.compress.GzipCodec
Bzip2	org.apache.hadoop.io.compress.BZip2Codec
Lzo	org.apache.hadoop.io.compress.lzo.LzoCodec
Lz4	org.apache.hadoop.io.compress.Lz4Codec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

3、压缩方式的选择：

压缩比率

压缩解压缩速度

是否支持 Split

4、压缩使用：

Job 输出文件按照 block 以 GZip 的方式进行压缩：

```
set mapreduce.output.fileoutputformat.compress=true                    // 默认值是 false
set mapreduce.output.fileoutputformat.compress.type=BLOCK              // 默认值是 Record
set
mapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.io.compress.GzipCodec
                                                  // 默认值是 org.apache.hadoop.io.compress.DefaultCodec
```

Map 输出结果也以 Gzip 进行压缩：

```
set mapred.map.output.compress=true
set mapreduce.map.output.compress.codec=org.apache.hadoop.io.compress.GzipCodec
                                                  // 默认值是 org.apache.hadoop.io.compress.DefaultCodec
```

对 Hive 输出结果和中间都进行压缩：

```
set hive.exec.compress.output=true      // 默认值是 false，不压缩
set hive.exec.compress.intermediate=true
                                                  // 默认值是 false，为 true 时 MR 设置的压缩才启用
```