1、Hive体系

1.1、Hive是什么?

由Facebook开源用于解决海量结构化日志的数据统计，后成为Apache Hive作为一个开源项目。

Hive是基于Hadoop的一个**数据仓库**工具，可以将**结构化的**数据文件映射成一张表，并提供类SQL查询功能；

使用HDFS存储；本质是将HQL转化成MapReduce程序,Hive的表其实就是HDFS上的目录和文件。

1.2、Hive的架构

1）用户接口：Client  
  CLI（hive shell）、JDFB/ODBC（java访问hive）、WEBUI（浏览器访问hive）

2）Metastore

Hive将元数据存储在数据库中，比如mysql ,derby.Hive中的元数据包括表的名称，表的列和分区及其属性，表的数据所在的目录

3）驱动器：Driver

解析器：将SQL字符串转换成抽象语法树AST，这一步一般都是第三方工具完成。

编译器：将AST编译成逻辑执行计划。

优化器：对逻辑执行计划进行优化。

执行器：把逻辑执行计划转换成可以运行的物理计划。对与hive来说就是MR、Spark。

4）Hive数据仓库和MySQL数据库的异同

MySQL数据库可以用在 Online 的应用中，Hive 主要进行离线的大数据分析

查询语句 HQL 和 SQL

存储位置 HDFS 和 LocalFS

索引 hive 没有

Hive 执行 MapReduce，mysql 执行 Executor

延迟性 hive 高

可扩展性 hive 高

数据规模 hive 大

5）hive优点与使用场景

操作接口采用类SQL语法，提供快速开发的能力（简单、容易上手）。  
避免了去写MR，减少开发人员的学习成本。  
统一的元数据管理，可以与impala/spark等共享元数据。  
易扩展（HDFS+MapReduce:可以扩展集群规模；支持自定义函数）。  
数据的离线处理；比如：日志分析、海量结构化数据离线分析。  
Hive的执行延迟比较高，因此hive常用于离线数据分析的，对于实时性要求不高的场合。  
Hive优势在于处理大数据，对于处理小数据没有优势，因为Hive的执行延迟比较高。

2、Hive的基本使用

1）创建表

hive> create table user (

>id int,

>name string,

>age int,

>sex string

>)

>row format delimited fields terminated by “\t”;

创建一个新表new\_table，结构与user表一样,但没有user中的数据。

hive>create table new\_table like user;

//用来指定原文件的列分隔符

row format delimited fields terminated by '\t';

2）查看表的信息

hive>desc formatted user;

hive>desc user;

3）清空表信息

hive>truncate table user;

4）删除表

hive>drop table user;

5）hdfs操作

hive>dfs -ls /user/hive/warehouse;

hive>dfs -mkdir /input/hellohive;

6）linux命令

hive>!clear;

hive>!ls /home/hadoop;

7）hive的脚本

-e 执行sql -f 执行sql文件 -S 静默执行

${HIVE\_HOME} bin/hive -e “select \* from emp”;

$ bin/hive -S -f /home/hadoop/emp.sql >~/result.txt;

8）加载数据

// 从linux本地加载数据到hive的user表中

hive>load data local inpath ”/home/hadoop/user.data” into table user;

// 从HDFS上加载数据到hive的user表中

hive>load data inpath “/input/user.log” into table user; //追加数据

// 覆盖已有的数据

hive>load data inpath “/input/user.log” overwrite into table user;

9）查询数据

hive>select \* from user;

3、内部表和外部表

1）创建

(1) 内部表(管理表) managed\_table

hive>create table emp (

>empid int,

>empName string,

>job string,

>salary flaot,

>deptid int

)

>row format delimited fields terminated by “\t”;

hive>load data inpath “/input/emp.log” into table emp;

1. 外部表(external\_table)

hive>create external table emp\_out (

>empId int,

>name string,

>job string,

>salary float,

>deptId int

>)

>row format delimited fields terminated by “\t”

>location “/hive/emp\_out”;

**--location之后的目录可以不存在，创建表会自动创建，但作为外部表推荐目录和数据已经存在**

hive>load data inpath “/input/emp.log” into table emp\_out;

2）区别

(1) 创建表的时候，外部表需要location记录hdfs表位置，此时也就完成了数据的加载。内部表默认存储在hive仓库中，加载数据的时候，会把hdfs上的数据移动到表所在的目录。

(2) 删除表的时候，外部表删除时，只是删除元数据，HDFS的目录和数据还在；

删除内部表时，删除了元数据和HDFS该表的目录和文件。

4、分区表

4.1、概念：

表的一个Partition对应表在HDFS的一个目录。可以根据时间划分，如果查询某个时间段，只要读取对应文件夹的数据即可，不用全部读取。

4.2、创建分区表

hive>create table emp\_part (

>id int,

>name string,

>job string,

>salary float,

>deptId int

>)

>partitoned by (province string)

>row format delimited fields terminated by “\t”;

4.3、分区操作

(1) 添加分区

alter table emp\_part add partition (province=”shanghai”)

(2) 删除分区

alter table emp\_part drop partition (province=”shanghai”);

1. 向分区添加数据

hive>Load data local inpath “/home/hadoop/emp.log” into table emp\_part partition(province=”shanghai”);

1. 查询分区数据

Select \* from emp\_part where province=”shanghai”;

4.4、二级分区

(1) 创建二级分区

hive>create table emp\_second(

>id int,

>name string,

>job string,

>salary float,

>dept int

>)

>partitioned by (day string,hour string)

>row format delimited fields terminated by “\t”;

(2) 添加分区

alter table emp\_second add partition (day=”20180331”,hour=”20”);

1. 删除分区

alter table emp\_second drop partition (day=”20180331”);

(4) 添加数据的时候指定分区(没有该分区会创建)

load data local inpath “/home/hadoop/emp.log” into table emp\_second partition (day=”201803031”,hour=”17”)

4.5、动态分区

1）开启动态分区

**set hive.exec.dynamic.partition=true**

默认值：false

是否开启动态分区功能，默认false关闭。

使用动态分区时候，该参数必须设置成true;

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

默认值：strict

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。

一般需要设置为nonstrict

set hive.exec.max.dynamic.partitions.pernode

默认值：100

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

默认值：1000

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。

同上参数解释。

set hive.exec.max.created.files

默认值：100000

整个MR Job中，最大可以创建多少个HDFS文件。

一般默认值足够了，除非你的数据量非常大，需要创建的文件数大于100000，可根据实际情况加以调整。

set hive.error.on.empty.partition

默认值：false

当有空分区生成时，是否抛出异常。一般不需要设置。

2）创建表

create table clear\_dynamic (

id string,

url string,

guid string

)

partitioned by (date string, hour string)

row format delimited fields terminated by '\t';

3）使用动态分区加载数据

insert into table clear\_dynamic partition (date='20150828', hour) select id,url,guid ,hour from tracklogs where date='20150828';

注：使用静态分区加载数据

insert into table clear partition(date='20150828', hour='18') select id,url,guid from tracklogs where date='20150828' and hour='18';

5、数据的导入和导出

5.1、数据的导入

(1) 本地导入

Hive>load data local inpath “本地路径” (overwrite) into table 表名

$HADOOP\_HOME bin/hdfs dfs -put 本地路径 hdfs路径(hive的表位置)

1. HDFS上导入

load data inpath “本地路径” (overwrite) into table 表名

1. 通过insert语句将select 的结果 插入到一张表中

insert into table test\_db select \* from emp\_p

(4) 创建表时加载数据

create external table test\_tb(

id int,

Name string

)

row format delimited fields terminated by “\t”

location “/hive/test\_tb”;

(5) 创建表时通过select查询语句加载数据

create table tableB row format delimited fields terminated by “,” as select \* from tableA;

区别：

load:仅仅进行的是拷贝或者是剪切操作

insert:底层运行的是mapreduce，可以进行自由变换表的分隔符，前提是必须有已经存在的表。

5.2、数据导出

1）hive脚本

(1) hive -e

bin/hive -e “use test\_db;select \* from emp\_p”

> /home/hadoop/result.txt

(2) hive -f 执行sql 文件

bin/hive -f 路径 >> /home/hadoop/result.txt

2）hive>

(1) 导出到本地（默认分隔符是 ASSII 001）

hive>insert overwrite local directory “/home/hadoop/data” select \* from emp\_p;

hive>insert overwrite local directory “/home/hadoop/data” row format delimited fields terminated by “^” select \* from emp\_p;

(2) 导出到HDFS

hive>insert overwrite directory “/data” select \* from emp\_p;

(3) export 和 import

hive> export table emp\_p to “/input/export”;

hive> import table emp\_imp from “hdfs\_path”;

6、HQL

6.1、常用语法

1）通配符\*

select \* from emp;

2）查询表中的几个字段

select empno,empname from emp;

3）distinct去重

select distinct(depno) from emp;

4）count 统计计数

select count(\*) from emp;

select count(1) from emp;

select count(empno) from emp;

5）max min sum avg

select sum(sal) from emp\_p;

6）limit

select \* from emp limit 2;

7）group by

求每个部门的薪资总和

select depno,sum(salary) sum\_salary from emp group by depno;

8）having 条件过滤

select depno, avg(salary) avg\_sal from emp group by depno hvaing avg\_sal >3000;

6.2、where

1）条件查询 < > = >= <=

select \* from emp\_p where salary > 10000;

2）between....... and...... and or

select \* from emp\_p where salary between 10000 and 15000;

3）is null | is not null

select \* from user where email is not null;

4）in() | not in()

select \* from emp\_p where did in (1,2,3);

6.3、子查询

Hive只支持where和from后面的子查询

select eid,ename,salary ,did from emp where emp.did in (select did from dept where dname='人事部');

6.4、表连接

emp.eid emp.ename emp.salary emp.did

1001 jack 10000.0 1

1002 tom 2000.0 2

1003 lily 20000.0 3

1004 aobama 10000.0 5

1005 yang 10000.0 6

dept.did dept.dname dept.dtel

1 人事部 021-456

2 财务部 021-234

3 技术部 021-345

4 BI部 021-31

5 产品部 021-232

1）积

select \* from dept, emp;

select \* from emp, dept where emp.did=dept.did;

2）join

select t1.eid, t1.ename, t1.salary,t2.did ,t2.dname from emp t1 **join** dept t2 **on** t1.did=t2.did;

3）外连查询

A) left join

select eid,ename, salary,t2.did, t2.dname from emp t1 **left join** dept t2 **on** t1.did = t2.did;

1. right join

select eid,ename, salary,t2.did, t2.dname from emp t1 **right join** dept t2 **on** t1.did = t2.did;

4) 全连接

select eid,ename, salary,t2.did, t2.dname from emp t1 **full join** dept t2 **on** t1.did = t2.did;

7、hql的四种排序

### 7.1、**order by**

order by 会对输入做全局排序，因此只有一个reducer（多个reducer无法保证全局有序），只有一个reducer，会导致当输入规模较大时，需要较长的计算时间。

set hive.mapred.mode=nonstrict; (default value / 默认值)

set hive.mapred.mode=strict;

 order by 和数据库中的Order by 功能一致，按照某一项 & 几项 排序输出。

 与数据库中 order by 的区别在于在hive.mapred.mode = strict 模式下 必须指定 limit 否则执行会报错。

hive> select \* from test order by id;

FAILED: Error in semantic analysis: 1:28 In strict mode, if ORDER BY is specified, LIMIT must also be specified. Error encountered near token 'id'

 原因：在order by 状态下所有数据会到一台服务器进行reduce操作也即只有一个reduce，如果在数据量大的情况下会出现无法输出结果的情况，如果进行limit n ，那只有n \* map number 条记录而已。只有一个reduce也可以处理过来。

### 7.2、**sort by**

sort by不是全局排序，其在数据进入reducer前完成排序。因此，如果用sort by进行排序，并且设置mapred.reduce.tasks > 1， 则sort by只保证每个reducer的输出有序，不保证全局有序。

sort by 不受 hive.mapred.mode 是否为strict ,nostrict 的影响

sort by 的数据只能保证在同一reduce中的数据可以按指定字段排序。

使用sort by 你可以指定执行的reduce 个数 （set mapred.reduce.tasks=<number>）,对输出的数据再执行归并排序，即可以得到全部结果。

注意：可以用limit子句大大减少数据量。使用limit n后，传输到reduce端（单机）的数据记录数就减少到n \* (map个数）。否则由于数据过大可能出不了结果。

### 7.3、**distribute by**

按照指定的字段对数据进行划分到不同的输出reduce文件中。此方法会根据name的长度划分到不同的reduce中，最终输出到不同的文件中。length 是内建函数，也可以指定其他的函数或这使用自定义函数。

分区排序（通过distribute by设置分区 ，使用 sort by设置分区内排序）

**set mapreduce.job.reduces=3;**

insert overwrite local directory “/home/hadoop/result” select \* from emp\_part distribute by province sort by salary;

7.4、Cluster By

distribute by 和sort by条件相同时 使用cluster by

但是排序只能是倒序排序，不能指定排序规则为asc 或者desc。

8、udf (用户自定义函数)

8.1、udf概念

使用内置的函数无法完成分析任务，那么需要写自定义函数

类型： UDF（一进一出） UDAF（多进一出） UDTF（一进多出）

8.2、编写udf步骤

1）必须继承UDF类

2）必须evaluate函数 支持重载

3）必须要有返回类型,可以返回null,但是返回类型不能为void

4）建议使用Text/LongWritable

8.3、案例

import org.apache.hadoop.hive.ql.exec.UDF;

import org.apache.hadoop.io.Text;

public class SalaryUDF extends UDF{

public Text evaluate(Text salaryText){

Text text = new Text();

//1.判断salaryText是否为null

if (salaryText == null) {

return null;

}

//2.判断salaryText是否可转换为一个double类型

double salary = 0;

try {

salary = Double.valueOf(salaryText.toString());

} catch (NumberFormatException e) {

e.printStackTrace();

return null;

}

if (salary > 3000) {

text.set("大于3000的一组...");

return text;

}else if (salary <= 3000 && salary > 2000) {

text.set("小于等于3000并且大于2000的一组...");

return text;

}else {

text.set("小于等于2000的一组");

return text;

}

}

}

package com.ibeifeng.udfdemo;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Locale;

import org.apache.commons.lang.StringUtils;

import org.apache.hadoop.hive.ql.exec.UDF;

import org.apache.hadoop.io.Text;

/\*\*

\*

\* 日期字符串 不易读->sdf1->日期对象(new Date)->sdf2->日期字符串 易读

\* 31/Aug/2015:00:04:37 +0800

\* 2015-8-31 00:04:37

\*/

public class DateFormatUDF extends UDF{

SimpleDateFormat inFormat =

new SimpleDateFormat("dd/MMM/yyyy:HH:mm:ss",Locale.ENGLISH);

SimpleDateFormat outFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

public Text evaluate(Text val) {

//1 判断是否为null和空

if(val==null) {

return null;

}

String dateStr = val.toString();

if(StringUtils.isBlank(dateStr)) {

return null;

}

//2 开始解析格式化

String result = null;//用于存储结果

try {

Date date = inFormat.parse(dateStr);

result = outFormat.format(date);

} catch (ParseException e) {

e.printStackTrace();

}

return new Text(result);

}

public static void main(String[] args) {

System.out.println(new DateFormatUDF().

evaluate(new Text("31/Aug/2015:00:04:37 +0800")));

}

}

8.4、实现步骤:

1）自定义类实现UDF类

2）打jar包 不需要指定主类 上传至Linux上

3）添加到hive中

hive (hive\_db)> add jar /home/hadoop/dateudf.jar;

hive (hive\_db)> create temporary function dateudf as

'com.ibeifeng.udfdemo.DateFormatUDF';

##### dateudf 函数名, 'com.ibeifeng.udfdemo.DateFormatUDF' 全限定名

hive (hive\_db)> show functions;

9、hive数据存储

9.1、**textfile (默认)**

1）textfile为默认格式

2）存储方式：行存储  
3）磁盘开销大 数据解析开销大  
4）压缩的text文件 hive无法进行合并和拆分

**9.2、**Parquet  
 1）Parquet也是一种行式存储，

2）同时具有很好的压缩性能；

3）同时可以减少大量的表扫描和反序列化的时间。

**9.3、rcfile**

1）存储方式：数据按行分块 每块按照列存储  
2）压缩快 快速列存取  
3）读记录尽量涉及到的block最少  
4）读取需要的列只需要读取每个row group 的头部定义。  
5）读取全量数据的操作 性能可能比sequencefile没有明显的优势

**9.4、orc（推荐使用）**

1）存储方式：数据按行分块 每块按照列存储  
2）压缩快 快速列存取  
3）效率比rcfile高,是rcfile的改良版本

**9.5、自定义格式**

用户可以通过实现inputformat和 outputformat来自定义输入输出格式。

### 9.6、根据不同的格式创建表

1）textfile格式

create table textfile(

id int,

name string,

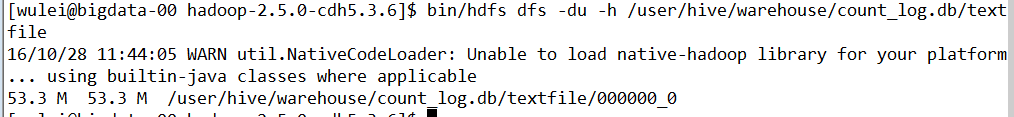
age int

)

row format delimited fields terminated by '\t'

stored as textfile;

insert into table textfile select \* from source\_log;



2）parquet格式

create table parquet(

id int,

name string,

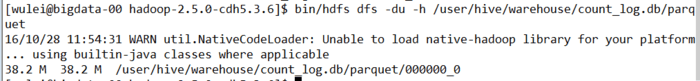
age int

)

row format delimited fields terminated by '\t'

stored as parquet;

insert into table parquet select \* from source\_log;



3）orc格式

create table orc(

id int,

name string,

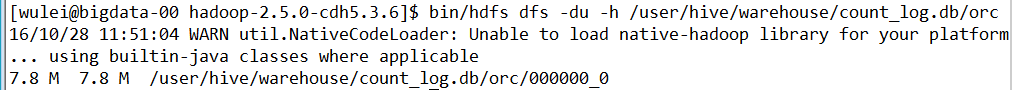
age int

)

row format delimited fields terminated by '\t'

stored as orc;

insert into table orc select \* from source\_log;

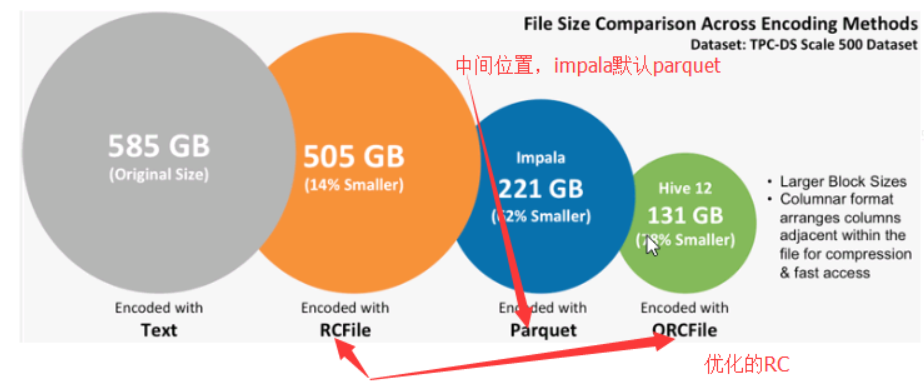


磁盘空间占用大小比较 orc(7.8M)<parquet(38.2M)<textfile(53.3M)

9.7、总结：

textfile 存储空间消耗比较大，并且压缩的text 无法分割和合并 查询的效率最低,可以直接存储，加载数据的速度最高。  
 parquet存储空间消耗最大,压缩的文件可以分割和合并查询效率高，需要通过text文件转化来加载。  
 orc 存储空间最小，查询的效率最高 ，需要通过text文件转化来加载，加载的速度最低。

**加载数据 ,一定不要用load，应使用insert。**



10、hive的压缩

10.1、压缩格式特点

压缩格式: bzip2, gzip, lzo, snappy等

压缩比：bzip2>gzip>lzo

bzip2最节省存储空间，但是同时需要消耗最多的CPU开销。

解压速度：lzo>gzip>bzip2 lzo解压速度是最快的

10.2、使用压缩的优缺点

优势：可以最小化所需要的磁盘存储空间，以及减小磁盘和网络I/O操作。

缺点：压缩过程和解压过程会增加CPU开销。

10.3、开启中间压缩

对中间数据进行压缩可以减少job中map和reduce task间的数据传输量。对于中间数据压缩，选择一个低CPU开销的编/解码器要比选择一个压缩率高的编/解码器要重要得多。

开启中间压缩：

<property>

    <name>hive.exec.compress.intermediate</name>

    <value>true</value>

</property>

10.4、最终输出结果压缩

属性hive.exec.compress.output，默认值false.输出就是非压缩的纯文本文件。

如果设置为true，需要指定一个编解码器。

<property>

    <name>hive.exec.compress.output</name>

    <value>true</value>

</property>

<property>

    <name>mapred.output.compression.codec</name>

    <value>org.apache.hadoop.io.compress.GzipCodec</value>

</property>

11、hive中的数据倾斜及解决方案

11.1、hive中的数据倾斜

11.1.1、操作

|  |  |  |
| --- | --- | --- |
| 关键字 | 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null值过多 | 这些空值都有一个reduce处理，非常慢 |
| group by | group by维度过小，某值得数量过多 | 处理某值的reduce灰常耗时 |
| Count Distinct | 某特殊值过多 | 处理此特殊值的reduce耗时 |

#### 11.1.2、数据倾斜的原因

1) key分布不均匀

2) 业务数据本身的特性

3) 建表时考虑不周

4) 某些SQL语句本身就有数据倾斜

#### 11.1.3、数据倾斜的表现

任务进度长时间维持在99%（或100%），查看任务监控页面，发现只有少量（1个或几个）reduce子任务未完成。因为其处理的数据量和其他reduce差异过大。

单一reduce的记录数与平均记录数差异过大，通常可能达到3倍甚至更多。 最长时长远大于平均时长。

### 11.2、数据倾斜的解决方案

#### 11.2.1参数调节

1）hive.map.aggr=true

Map 端部分聚合，相当于Combiner

2）hive.groupby.skewindata=true

有数据倾斜的时候进行负载均衡，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 的输出结果集合会随机分布到 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。

11.2.2 SQL语句调节：

****1）如何Join：****

关于驱动表的选取，选用join key分布最均匀的表作为驱动表

做好列裁剪和filter操作，以达到两表做join的时候，数据量相对变小的效果。

****2）大小表Join：****

使用map join让小的维度表（1000条以下的记录条数） 先进内存。在map端完成reduce.

****3）大表Join大表****：

把空值的key变成一个字符串加上随机数，把倾斜的数据分到不同的reduce上，由于null值关联不上，处理后并不影响最终结果。

****4）count distinct大量相同特殊值****

count distinct时，将值为空的情况单独处理，如果是计算count distinct，可以不用处理，直接过滤，在最后结果中加1。如果还有其他计算，需要进行group by，可以先将值为空的记录单独处理，再和其他计算结果进行union。

****5）group by维度过小：****

采用sum() group by的方式来替换count(distinct)完成计算。

**特殊情况特殊处理：**

在业务逻辑优化效果的不大情况下，有些时候是可以将倾斜的数据单独拿出来处理。最后union回去。

### 11.3、**典型的业务场景**

#### **11.3.1、空值产生的数据倾斜**

**场景：**如日志中，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 关联，会碰到数据倾斜的问题。

**解决方法1：** user\_id为空的不参与关联（红色字体为修改后）

select \* from log a

join users b

on a.user\_id is not null

and a.user\_id = b.user\_id union all select \* from log a

where a.user\_id is null;

****解决方法2：****赋与空值分新的key值

select \* from log a

left outer join users b

on

case when a.user\_id is null

then concat(‘hive’,rand() )

else a.user\_id end = b.user\_id;

**结论：**方法2比方法1效率更好，不但io少了，而且作业数也少了。解决方法1中 log读取两次，jobs是2。解决方法2 job数是1 。这个优化适合无效 id (比如 -99 , ’’, null 等) 产生的倾斜问题。把空值的 key 变成一个字符串加上随机数，就能把倾斜的数据分到不同的reduce上 ,解决数据倾斜问题。

#### **11.3.2不同数据类型关联产生数据倾斜**

**场景：**用户表中user\_id字段为int，log表中user\_id字段既有string类型也有int类型。当按照user\_id进行两个表的Join操作时，默认的Hash操作会按int型的id来进行分配，这样会导致所有string类型id的记录都分配到一个Reducer中。

**解决方法：**把数字类型转换成字符串类型

select \* from users a

left outer join logs b

on a.usr\_id = cast(b.user\_id as string)

#### **11.3.3、小表不小不大，怎么用 map join 解决倾斜问题**

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。

**以下例子:**

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。

**解决方法：**

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。

### **11.4、总结**

使map的输出数据更均匀的分布到reduce中去，是我们的最终目标。由于Hash算法的局限性，按key Hash会或多或少的造成数据倾斜。大量经验表明数据倾斜的原因是人为的建表疏忽或业务逻辑可以规避的。在此给出较为通用的步骤：

1、采样log表，哪些user\_id比较倾斜，得到一个结果表tmp1。由于对计算框架来说，所有的数据过来，他都是不知道数据分布情况的，所以采样是并不可少的。

2、数据的分布符合社会学统计规则，贫富不均。倾斜的key不会太多，就像一个社会的富人不多，奇特的人不多一样。所以tmp1记录数会很少。把tmp1和users做map join生成tmp2,把tmp2读到distribute file cache。这是一个map过程。

3、map读入users和log，假如记录来自log,则检查user\_id是否在tmp2里，如果是，输出到本地文件a,否则生成<user\_id,value>的key,value对，假如记录来自member,生成<user\_id,value>的key,value对，进入reduce阶段。

4、最终把a文件，把Stage3 reduce阶段输出的文件合并起写到hdfs。

**如果确认业务需要这样倾斜的逻辑，考虑以下的优化方案：**

1、对于join，在判断小表不大于1G的情况下，使用map join

2、对于group by或distinct，设定 hive.groupby.skewindata=true

3、尽量使用上述的SQL语句调节进行优化

12、hive优化

1）大表拆分成子表，提取中间结果集，减少每次加载数据。

多维度分析，多个分析模块。

每个分析模块涉及字段不一样，而且并不是表的全部字段。

2）分区表及外部表

设计二级分区表（一级字段为天，二级字段设置小时）。

创建外部表，创建表时直接指定数据所在目录即可，不用再用load加载数据。

3）推测执行：例如一个Job应用有10个MapReduce任务（map 及reduce），其中9个任务已经完成，那么application Master会在另外启动一个相同的任务来运行未完成的那个，最后哪个先运行完成就把另一个kill掉。

启用speculative最大的好处是，一个map执行的时候，系统会在其他空闲的服务器上启动相同的map来同时运行，哪个运行的快就使用哪个的结果，另一个运行慢的在有了结果之后就会被kill。

**hive-site.xml**

<property>

<name>hive.mapred.reduce.tasks.speculative.execution</name>

<value>true</value>

</property>

4）并行执行：一个sql有1、2、3 三个job，其中1、2 job是在join后再与3 job结果进行join，那么1、2 job在join同时job 3可以同时并行执行。

set hive.exec.parallel=true;

<description>Whether to execute jobs in parallel</description>

set hive.exec.parallel.thread.number=8;

<description>How many jobs at most can be executed in parallel</description>

eg:

select num from (select count(city) as num from city

union all select count(province) as num from province)tmp;

select a.id,a.name form user a left join

(select b.id,b.cityid from order) b on

a.id=b.id join

(select c.city,c.cityname from area) c

on b.cityid=c.ctiyid;

5）JVM重用

每次任务创建一个JVM实例(默认情况下一个任务对应一个JVM)

修改mapred-site.xml mapreduce.job.jvm.num.tasks(新版) 默认为1

也可在hive的执行设置： set mapred.job.reuse.jvm.num.tasks=10;

6）MapReduce过程的map、shuffle、reduce端的snappy压缩

需要先替换hadoop的native本地包开启压缩

在mapred-site.xml文件设置启用压缩及压缩编码

在执行SQL执行时设置启用压缩和指定压缩编码

set mapreduce.output.fileoutputformat.compress=true;

set mapreduce.output.fileoutputformat.compress.codec=

org apache.hadoop.io.compress.SnappyCodec;

7）合并小文件  
 hive.merg.mapfiles=true：合并map输出  
 hive.merge.mapredfiles=false：合并reduce输出  
 hive.merge.size.per.task=256\*1000\*1000：合并文件的大小  
 hive.mergejob.maponly=true：

如果支持CombineHiveInputFormat则生成只有Map的任务执行merge  
hive.merge.smallfiles.avgsize=16000000：

文件的平均大小小于该值时，会启动一个MR任务执行merge。

8）使用索引：  
 hive.optimize.index.filter：自动使用索引  
 hive.optimize.index.groupby：使用聚合索引优化GROUP BY操作

9）动态分区：  
 hive.exec.dynamic.partition.mode=strict：该模式下必须指定一个静态分区  
 hive.exec.max.dynamic.partitions=1000  
 hive.exec.max.dynamic.partitions.pernode=100：

在每一个mapper/reducer节点允许创建的最大分区数

DATANODE：dfs.datanode.max.xceivers=8192：允许DATANODE打开多少个文件