Day12_Hive的优化及分区分桶操作

大数据-张军锋 Day12 Hive 优化 分区 分桶

Hive 优化

> 在我们学习阶段，集群都是搭在虚拟机上，相对于真机而言，是很卡的，下面是一些对于**集群优化的操作**。
> 的是将程序发布在本节点上进行执行，不通过yarn进行发布，避免了RPC的传输过程，从而减少服务器的压力。

设置是否自动开启本地模式
`set hive.exec.mode.local.auto=true`
本地模式容忍的最大文件个数
`hive.exec.mode.local.auto.input.files.max=1`
本地模式最大文件字节数
`hive.exec.mode.local.auto.inputbytes.max=1`
reduce处理的数据量 `hive.exec.reducers.bytes.per.reducer=256000000`
reduce最大个数 `hive.exec.reducers.max=1009`

运行程序，测试一下

```
select *
from dw_employee a
inner join txt_department b
on a.dep_id=b.dep_id
```

Map端聚合操作

> 在Map端进行聚合，在一定程度上可以减少数据的传输量，从侧面上提高服务器的性能。对于map端聚合操作，需要在hive上设置一些参数

决定着是否开启自动的map端join
`hive.auto.convert.join=false`
决定是否使用map端join，如果关联表有一个小于这个参数的配置则自动开map端join
`hive.mapjoin.smalltable.filesize=25000000`

> 我们一般情况下是通过sql语句，想要进行Map端的聚合操作，下面是Map端聚合的示例代码

特殊情况，**手动开启map端join**

```
select /*+MAPJOIN(txt_department )*/
       a.*,b.**
from dw_employee a
inner join txt_department b
on a.dep_id=b.dep_id
```

或者进行如下设置，**新版本比较推荐使用下面的版本**

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

select count(*)
from store_sales
join time_dim
on (ss_sold_time_sk = t_time_sk)
```

Order & Sort & Cluster & Distribute By

> 对于order，sort，cluster，distribute这几概念，在面试的过程中是很容易被提问到的，下面对于这些概念做简单的介绍

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。

使用order by需要进行下面的设置

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

> order by和数据库中的操作基本上是一样的，下面对于一些区别简单描述一下

order by的使用上与mysql**最大的不同**

```
select cardno,count(*)
from tableA
group by idA
order by count(*) desc limit 10
```

这个语句在mysql中查询的时候，肯定是没有问题的，而且我们实际上也经常这么干。但是如果将上述语句提交给hive，会报以下错误：

```
FAILED: SemanticException [Error 10128]: Line 4:9 Not yet supported
place for UDAF 'count'
```

怎么样可以呢？将count(*)给一个别名就好：

```
select cardno,count(*) as num
from tableA
group by idA
order by num desc limit 10
```

这样就可以了。本博主没查源码，估计是因为hive查询的时候起的是mr任务，mr任务里排序的时候，不认得count(*)是什么东东，所以给个别名就好。

> hive的底层原理是将order by进行了全排序，在单个节点上可以进行排序的，在多个节点就显的力不从心了。因此，对于大数据order by就无能为力了

Sort by

sort by不是全局排序，其在数据进入reducer前完成排序.

- 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= ）,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。

```
set mapreduce.job.reduces = 2
create table dep_sort as
-- 计算每一个部门的人数和薪水支出
select * from(
select a.dep_id
      ,a.dep_name
      ,a.dep_address
      ,count(b.emp_id) p_num
      ,sum(nvl(salary,0)) a_salary
from dep a
left join dw_employee b
on cast(a.dep_id as int) = b.dep_id
Group by a.dep_id
        ,a.dep_name
        ,a.dep_address
)a
sort by p_num
```

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。

```
insert overwrite local directory '/home/hadoop/out'
select *
from test
order by name
distribute by length(name);
```

- 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。

```
-- distribute by
create table emp_distribute as
select * from dw_employee
distribute by status

dfs -cat /user/hive/warehouse/db14.db/emp_distribute/000000_0
dfs -cat /user/hive/warehouse/db14.db/emp_distribute/000001_0

select *
from dw_employee
distribute by status
sort by status,salary desc
```

Cluster by

Cluster by 不够灵活，因为使用哪个字段进行分区，就要使用哪个字段进行排序

- Cluster by 是distribute by 和sort by 的结合
- 只允许升序，不允许升序

> **比如：** `Cluster by column_1` 等价于 `Distribute by column_1 sort by column_1`

```
select *
from dw_employee
cluster by dep_id
```

复杂数据类型

复杂数据类型array、map、union、struct等等，下面我们以一个例子来说明

首先，我们创建一个对应数据类型的表，对表进行操作，下面是创建表的sql

```
drop table test_serializer
create table test_serializer(
     string1 string
    ,int1 int
    ,tinnyint1 tinyint
    ,smallint1 smallint
    ,bigint1 bigint
    ,boolean1 boolean
    ,float1 float
    ,double1 double
    ,list1 array<string>
    ,map1 map<string,int>
    ,struct1 struct<sint:int,sboolean:boolean,sstring:string>
    ,union1 uniontype<float,boolean,string>
    ,enum1 string
    ,nullableint int
    ,bytes1 binary
    ,fixed1 binary
)
row format delimited
fields terminated by ','
collection items terminated by ':'
MAP KEYS TERMINATED BY '#'
lines terminated by '\n'
NULL DEFINED AS 'NULL'
stored as textfile
```

> 对于基本类型的操作，相信大家已经熟记于心，下面对于复杂数据类型做简单的描述

array

```
0: jdbc:hive2://master:10000/> select list1 from test_serializer;
+---------------------------+
|           list1           |
+---------------------------+
| ["alpha","beta","gamma"]  |
| ["beta"]                  |
| ["alpha","gamma"]         |
+---------------------------+
```

```
0: jdbc:hive2://master:10000/> select list1[0] from test_serializer;
+--------+
|  _c0   |
+--------+
| alpha  |
| beta   |
| alpha  |
+--------+
```

```
0: jdbc:hive2://master:10000/> select size(list1) from test_serializer;
+------+
| _c0  |
+------+
| 3    |
| 1    |
| 2    |
+------+
```

- **展平array记录**

```
0: jdbc:hive2://master:10000> select explode(list1) from test_serializer;
+--------+
|  col   |
+--------+
| alpha  |
| beta   |
| gamma  |
| beta   |
| alpha  |
| gamma  |
+--------+
```

Map

```
0: jdbc:hive2://master:10000/> select map1 from test_serializer;
+-----------------------------------------------------+
|                        map1                         |
+-----------------------------------------------------+
| {"Earth":42,"Control":86,"Bob":31}                  |
| {"Earth#":101}                                      |
| {"Earth":237,"Bob":7723}                            |
+-----------------------------------------------------+
```

```
0: jdbc:hive2://master:10000/> select map1['Earth'] from test_seria
lizer;
+-------+
|  _c0  |
+-------+
| 42    |
| 101   |
| 237   |
+-------+
```

```
0: jdbc:hive2://master:10000/> select size(map1) from test_serializ
er;
+------+
| _c0  |
+------+
| 3    |
| 1    |
| 2    |
+------+
```

```
0: jdbc:hive2://master:10000/> select map_keys(map1) from test_seri
alizer;
+-------------------------------------+
|                 _c0                 |
+-------------------------------------+
| ["Earth","Control","Bob"]           |
| ["Earth"]                           |
| ["Earth","Bob"]                     |
+-------------------------------------+
```

```
0: jdbc:hive2://master:10000/> select map_keys(map1)[0] from test_s
erializer;
+------------+
|    _c0     |
+------------+
| Earth      |
| Earth      |
| Earth      |
+------------+
```

```
0: jdbc:hive2://master:10000/> select map_values(map1) from test_se
rializer;
+--------------------+
|        _c0         |
+--------------------+
| [42,86,31]         |
| [101]              |
| [237,7723]         |
+--------------------+
```

```
0: jdbc:hive2://master:10000/> select map_values(map1)[0] from tes
t_serializer;
+-------+
|  _c0  |
+-------+
| 42    |
| 86    |
| 31    |
+-------+
```

- **展平map记录**

```
0: jdbc:hive2://master:10000> select explode(map1) from test_serial
izer;
+-------------+--------+
|     key     | value  |
+-------------+--------+
| Earth       | 42     |
| Control     | 86     |
| Bob         | 31     |
| Earth       | 101    |
| Earth       | 237    |
| Bob         | 723    |
+-------------+--------+
```

Struct

```
0: jdbc:hive2://master:10000> select struct1 from test_serializer;
+-----------------------------------------------------+
|                       struct1                       |
+-----------------------------------------------------+
| {"sint":17,"sboolean":true,"sstring":"Abe Linkedin"} |
| {"sint":1134,"sboolean":false,"sstring":"wazzup"}   |
| {"sint":102,"sboolean":false,"sstring":"BNL"}       |
+-----------------------------------------------------+
```

- **Struct取数据**

```
0: jdbc:hive2://master:10000> select struct1.sboolean from test_serializer;
+-----------+
| sboolean  |
+-----------+
| true      |
| false     |
| false     |
+-----------+
```

```
0: jdbc:hive2://master:10000> select struct1.sstring from test_serializer;
+---------------+
|    sstring    |
+---------------+
| Abe Linkedin  |
| wazzup        |
| BNL           |
+---------------+
```

文件的保存格式

> hive 默认支持的文件格式有很多，其中**arvo、orc、Parquet、Compressed Data Storage、LZO Compression**等等，详细介绍请参考[https://cwiki.apache.org/confluence/display/Hive/LanguageManual][8]

创建avro文件格式的表 avro_emploree

```
create table avro_emploree

stored as avro
as
select * from dw_employee
select * from avro_emploree
dfs -cat /user/hive/warehouse/db14.db/avro_emploree/000000_0
```

hive分区

> 在Hive Select查询中一般会扫描整个表内容，会消耗很多时间做没必要的工作。有时候只需要扫描表中关心的一部分数据，因此建表时引入了**partition概念**。
> **分区表指的是在创建表时指定的partition的分区空间。**

Hive可以对数据按照某列或者某些列进行分区管理，所谓分区我们可以拿下面的例子进行解释。
当前互联网应用每天都要存储大量的日志文件，几G、几十G甚至更大都是有可能。存储日志，其中必然有个属性是日志产生的日期。在产生分区时，就可以按照日志产生的日期列进行划分。把每一天的日志当作一个分区。
将数据组织成分区，主要可以提高数据的查询速度。至于用户存储的每一条记录到底放到哪个分区，由用户决定。即用户在加载数据的时候必须显示的指定该部分数据放到哪个分区。

> **最常用的分区条件**
>
> 1. 时间(年月日)
> 2. 行政区划(省，地市区县)
> 3. 具体的业务类型(不太常用)

创建分区文件

创建分区表

```
drop table p_orders
create table p_orders(
     order_id int
    ,order_date string
    ,customer_id int
    ,order_status string
)
partitioned by (date_month string)
row format delimited
fields terminated by '|'
```

新增分区

```
alter table p_orders add partition(date_month='201709');
alter table p_orders add partition(date_month='201708');
```

删除分区

```
alter table p_orders drop partition(date_month='201709');
```

静态导入数据

```
load data inpath '/orderdata/orders' overwrite into table p_orders
partition(date_month='201709')
```

动态导入数据

> 当数据使用load静态导入时，hive是不会对数据做任何转换的， 它只是单纯的把数据复制到表分区的目录下而已

动态导入数据到分区

0. 设置参数

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

1. 使用insert into select语句来完成数据的动态导入

```
create TEMPORARY table temp_orders(
     order_id int
    ,order_date string
    ,customer_id int
    ,order_status string
)
row format delimited
fields terminated by '|'
stored as textfile
```

2. 把数据加载到临时表

```
load  data local inpath '/root/orderdata/orders' overwrite into tab
le temp_orders
```

3. 用insert into select从临时表中取数据转换分区字段到分区表p_orders中

```
select * from temp_orders

insert into table p_orders partition(date_month)
select order_id
      ,order_date
      ,customer_id
      ,order_status
      ,date_format(to_date(order_date),'yyyyMM') as date_month
from temp_orders

select * from p_orders where date_month='201310'

show tables
```

二级分区

> 二级分区是按照目录结构进行分层操作的

```
drop table p_test
create table p_test(
     test1 string
    ,test2 string
)
partitioned by (date_day string,date_hour string)

show tables;

alter table p_test add partition(date_day='20171025',date_hour='0
1');
alter table p_test add partition(date_day='20171025',date_hour='0
2');
alter table p_test add partition(date_day='20171025',date_hour='0
3');
alter table p_test add partition(date_day='20171026',date_hour='0
1');
alter table p_test add partition(date_day='20171026',date_hour='0
2');
alter table p_test add partition(date_day='20171026',date_hour='0
3');
```

分桶

> 对于每一个表（table）或者分区， Hive可以进一步组织成桶，也就是说桶是更为细粒度的数据范围划分。Hive也是 针对某一列进行桶的组织。Hive采用对列值哈希，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。

把表（或者分区）组织成桶（Bucket）有两个理由：

1. 获得更高的查询处理效率。桶为表加上了额外的结构，Hive 在处理有些查询时能利用这个结构。具体而言，连接两个在（包含连接列的）相同列上划分了桶的表，可以使用 Map 端连接 （Map-side join）高效的实现。比如JOIN操作。对于JOIN操作两个表有一个相同的列，如果对这两个表都进行了桶操作。那么将保存相同列值的桶进行JOIN操作就可以，可以大大较少JOIN的数据量。
2. 使取样（sampling）更高效。在处理大规模数据集时，在开发和修改查询的阶段，如果能在数据集的一小部分数据上试运行查询，会带来很多方便。

> 注意：分桶对应的是文件，和分区是不同的。

![分桶原理示意图][9]

创建分桶

```
create table pb_orders(
     order_id int
    ,order_date string
    ,customer_id int
    ,order_status string
)
partitioned by (date_month string)
clustered by(customer_id) sorted by(customer_id) into 2 buckets
stored as textfile

insert into table pb_orders partition(date_month)
select order_id
      ,order_date
      ,customer_id
      ,order_status
      ,date_format(to_date(order_date),'yyyyMM') as date_month
from temp_orders


dfs -cat /user/hive/warehouse/db14.db/pb_orders/date_month=201307/0
00000_0
dfs -cat /user/hive/warehouse/db14.db/pb_orders/date_month=201307/0
00001_0

select *
from pb_orders
where date_month='201307' and customer_id=5125

select *
from temp_orders
where date_format(to_date(order_date),'yyyyMM')='201307' and custom
er_id=5125
```

hive压缩

> 对于为什么要压缩，以及压缩的格式，在这里就不说了，不明白的请找度娘，这里直接上代码了

压缩

```
set hive.exec.compress.output=true

create table compress_order as
select * from temp_orders
```

设置压缩格式为gzip

```
set mapreduce.output.fileoutputformat.compress.codec=org.apache.had
oop.io.compress.GzipCodec

create table compress_order as
select * from temp_orders
```

maven 更换国内镜像

> 在maven的安装目录下的conf文件夹，找到setting.xml文件添加如下代码、

```
<mirrors>
  <!-- mirror
   | Specifies a repository mirror site to use instead of a given repository. The r
   | this mirror serves has an ID that matches the mirrorOf element of this mirror.
   | for inheritance and direct lookup purposes, and must be unique across the set
   |
  <mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://my.repository.com/repo/path</url>
  </mirror>
   -->
  <mirror>
      <id>alimaven</id>
      <name>aliyun maven</name>
      <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
      <mirrorOf>central</mirrorOf>
  </mirror>


</mirrors>
```

```
<mirror>
      <id>alimaven</id>
      <name>aliyun maven</name>
      <url>http://maven.aliyun.com/nexus/content/groups/public/</ur
l>
      <mirrorOf>central</mirrorOf>
</mirror>
```