大数据开发优化规范

(Hive 篇)

(v1.0)

目录

更改记录

- 20161117：1.0 版成稿。

修改项	修改内容	
第一版内容	第一版内容	修改时间：2016-11-17 修改人：　张宇龙 审阅人：　张宇龙

1 概述

本文主要介绍 Hive、MapReduce 以及流程方面的优化。

2 介绍

在 Hadoop 集群使用 MapRedcue 引擎执行任务，包括 Hive、Pig 等组件，会衍生哪些问题?

- 文件存储格式是否合适
- 文件压缩格式是否合适
- 同一个 SQL 中产生的多段 MapReduce 是否合理，是否可以减少 job 数
- Map 数目是否合理
- Reduce 数目是否合理
- 如何合理的进行小文件合并
- 合理分配集群资源给任务和流程
- 流程上整体优化大于单个优化

3 思路

Hive 性能优化时，既要关注 Hive 本身的一些优化特性，又要把 HiveQL 当做 M/R 程序来读，即从 M/R 的运行角度来考虑优化性能，从更底层思考如何优化运算性能，而不仅仅局限于逻辑代码的替换层面。

首先要了解 Hive 和 MapReduce 的特性，MapReduce 将一个大型的任务，拆解成了一个个小型任务，可以比作一个巨大的货轮，出发前需要把所有的集装箱搬移到货轮上，所以 MapReduce 启动开销很大。

MapReduce 任务分为 Map 阶段和 Reduce 阶段。

Map 阶段主要就是把需要加工的数据读取到各个小任务中，主要的操作是加载数据和过滤等操作。Map 阶段关注的是加载数据的文件格式和压缩格式，因为这直接涉及到加载和解析的效率。如何使得 Map 加载数据能用充分利用分配给 Map 任务的计算资源是这部分优化的关键。

Reduce 阶段主要做的是 sort 及其他操作。同一 key 值过多，会导致数据倾斜，也就导致某几个 Reduce 计算的数据量相对其他 Reduce 会大很多。如何避免数据倾斜，抑制过多不必要的 Reduce 数目是这部分优化的关键。

在引入 Yarn 资源管理之后，如何对每个 Map 任务和 Reduce 任务分配合适的资源，不仅影响到单个任务的计算效率，还直接关系到整体流程运行的效率。记住，整体优化比单个优化来的更有意义。

流程优化上的原则是，在任务一定的情况下，如何合理的利用计算资源，将流程整体时间压缩到最短。

Hive 和 MapReduce 中拥有较多在特定情况下优化的特性，如何利用好这些特性，也是优化的关键。

4 Hive 优化特性

4.1 本地模式

4.1.1 特性介绍

当一个 MapReduce 任务的数据量和计算任务很小的时候，在 MapReduce 框架中 Map 任务和 Reduce 任务的启动过程占用的任务执行的大部分时间，真正的逻辑处理其实占用时间很少，但是给用户的感受就是：很小的任务，同样执行较长的时间。比如对一张码表进行计算，总时间可能接近 1~2 分钟，这个对于用户来说，感受很差。

那么在 0.7 版本之后，Hive 引入了本地模式，那么对于小任务的执行，Hive 客户端不再需要到 Yarn 上申请 Map 任务和 Reduce 任务，只需要在本地进行 Map 和 Reduce 的执行，大大的加快了小任务的执行时间，通常可以把分钟级别任务的执行时间降低秒级。

4.1.2 参数设置

- 如下参数设置是否开启本地模式

```
hive.exec.mode.local.auto
Default Value: false
Added In: Hive 0.7.0 with HIVE-1408
Lets Hive determine whether to run in local mode automatically.
```

- 如下参数限定了 Map 输入的文件大小.如果是高压缩率列存文件，可适当减小此值，避免进入本地模式，一般选择默认值即可。

```
hive.exec.mode.local.auto.inputbytes.max
Default Value: 134217728
Added In: Hive 0.7.0 with HIVE-1408
```

```
When hive.exec.mode.local.auto is true, input bytes should be less than this for local mode.
```

- 如下参数限定了 Map 输入的文件个数.如果是多个小文件，可适当增大此值，一般选择默认值即可

```
hive.exec.mode.local.auto.input.files.max
Default Value: 4
Added In: Hive 0.9.0 with HIVE-2651
When hive.exec.mode.local.auto is true, the number of tasks should be less than this for local mode.
```

4.1.3 性能对比

使用本地模式之后，对于小表的计算查询从 34 秒减少到了 2 秒。

- 非本地模式

```
hive (default)> set hive.exec.mode.local.auto=false;
hive (default)> select count(*) from test;
Query ID = hadoop_20161115132159_cc9537d6-77cf-4f01-9983-3f914cd9a02d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1478583841022_0007, Tracking URL =
http://breath:23188/proxy/application_1478583841022_0007/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1478583841022_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-11-15 13:22:20,922 Stage-1 map = 0%,  reduce = 0%
2016-11-15 13:22:30,988 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 12.79 sec
2016-11-15 13:22:32,100 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 13.96 sec
MapReduce Total cumulative CPU time: 13 seconds 960 msec
Ended Job = job_1478583841022_0007
MapReduce Jobs Launched:
```

```
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 13.96 sec   HDFS Read: 10683 HDFS Write:
485280 SUCCESS
Total MapReduce CPU Time Spent: 13 seconds 960 msec
OK
4
Time taken: 34.331 seconds, Fetched: 1 row(s)
```

- 本地模式

```
hive (default)> set hive.exec.mode.local.auto=true;
hive (default)> select count(*) from test;
Automatically selecting local only mode for query
Query ID = hadoop_20161115132307_e6dbeb81-8990-42cf-9a35-b4dae4a128c1
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2016-11-15 13:23:09,096 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1985871023_0002
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 300 HDFS Write: 82624694 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
4
Time taken: 2.201 seconds, Fetched: 1 row(s)
```

4.2 MAPJOIN

4.2.1 特性介绍

MAPJOIN 是将小表以 hash map 的方式装入内存，然后用 Key 值跟大表去匹配，分解后的步骤如下所示：

- Local work:

    - read records via standard table scan (including filters and projections) from source on local machine
    - build hashtable in memory
    - write hashtable to local disk
    - upload hashtable to dfs
    - add hashtable to distributed cache
- Map task
    - read hashtable from local disk (distributed cache) into memory
    - match records' keys against hashtable
    - combine matches and write to output
- No reduce task

使用 MAPJION 的主要作用是有效的减少一个 SQL 任务中的 Stage 层数。同一个 SQL 任务下，越少的 Stage 层代表越快的效率。

4.2.2 参数设置

- 如下参数设置是否开启自动转换为 MAPJOIN

```
hive.auto.convert.join
Default Value: false in 0.7.0 to 0.10.0; true in 0.11.0 and later (HIVE-3297)
Added In: 0.7.0 with HIVE-1642
Whether Hive enables the optimization about converting common join into mapjoin based
on the input file size. (Note that hive-default.xml.template incorrectly gives the
default as false in Hive 0.11.0 through 0.13.1.)
```

```
hive.mapjoin.followby.map.aggr.hash.percentmemory
Default Value: 0.3
Added In: Hive 0.7.0 with HIVE-1830
Portion of total memory to be used by map-side group aggregation hash table, when this
group by is followed by map join.
```

```
hive.mapjoin.smalltable.filesize
Default Value: 25000000
The threshold (in bytes) for the input file size of the small tables; if the file size
is smaller than this threshold, it will try to convert the common join into map join.
```

```
hive.auto.convert.join.noconditionaltask
Default Value: true
Added In: 0.11.0 with HIVE-3784 (default changed to true with HIVE-4146)
```

```
Whether Hive enables the optimization about converting common join into mapjoin based
on the input file size. If this parameter is on, and the sum of size for n-1 of the
tables/partitions for an n-way join is smaller than the size specified by
hive.auto.convert.join.noconditionaltask.size, the join is directly converted to a
mapjoin (there is no conditional task).
```

```
hive.auto.convert.join.noconditionaltask.size
Default Value: 10000000
Added In: 0.11.0 with HIVE-3784
If hive.auto.convert.join.noconditionaltask is off, this parameter does not take effect.
However, if it is on, and the sum of size for n-1 of the tables/partitions for an n-way
join is smaller than this size, the join is directly converted to a mapjoin (there is
no conditional task). The default is 10MB.
```

```
hive.auto.convert.join.use.nonstaged
Default Value: false
Added In: 0.13.0 with HIVE-6144 (default originally true, but changed
to false with HIVE-6749 also in 0.13.0)
For conditional joins, if input stream from a small alias can be directly applied to
the join operator without filtering or projection, the alias need not be pre-staged in
the distributed cache via a mapred local task. Currently, this is not working with
vectorization or Tez execution engine.
```

If hive.auto.convert.join is set to true the optimizer not only converts joins to mapjoins but also merges MJ* patterns as much as possible.

4.2.3 性能对比

使用 TPC-DS 中的一个测试案例进行对比测试，发现使用 MAPJOIN 特性之后将原来的 6 个 stage 层减少到 4 个 stage 层，时间也有原来的 123 秒减少到 72 秒。

- 未使用 MAPJOIN

```
hive (hahaha)> set hive.auto.convert.join=false;
hive (hahaha)> Select count(*) cnt
            > From store_sales ss
            >     join household_demographics hd on (ss.ss_hdemo_sk = hd.hd_demo_sk)
            >     join time_dim t on (ss.ss_sold_time_sk = t.t_time_sk)
            >     join store s on (s.s_store_sk = ss.ss_store_sk)
            > Where
            >     t.t_hour = 8 and
            >     t.t_minute >= 30 and
            >     hd.hd_dep_count = 2
            > order by cnt;
Query ID = hadoop_20161115133812_89f93a1f-16e8-44bd-b381-881e0c707d03
Total jobs = 5
Launching Job 1 out of 5
Number of reduce tasks not specified. Estimated from input data size: 16
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Cannot run job locally: Input Size (= 3999072949) is larger than
hive.exec.mode.local.auto.inputbytes.max (= 134217728)
Starting Job = job_1479182500047_0003, Tracking URL =
http://hadoop001:23188/proxy/application_1479182500047_0003/
```

Kill Command = /opt/beh/core/hadoop/bin/hadoop job -kill job_1479182500047_0003

```
Hadoop job information for Stage-1: number of mappers: 16; number of reducers: 16
2016-11-15 13:38:24,252 Stage-1 map = 0%,  reduce = 0%
2016-11-15 13:38:32,857 Stage-1 map = 6%,  reduce = 0%, Cumulative CPU 2.83 sec
2016-11-15 13:38:36,054 Stage-1 map = 27%,  reduce = 0%, Cumulative CPU 76.33 sec
2016-11-15 13:38:37,127 Stage-1 map = 35%,  reduce = 0%, Cumulative CPU 111.93 sec
2016-11-15 13:38:39,276 Stage-1 map = 59%,  reduce = 0%, Cumulative CPU 146.56 sec
2016-11-15 13:38:40,341 Stage-1 map = 75%,  reduce = 0%, Cumulative CPU 165.16 sec
2016-11-15 13:38:41,623 Stage-1 map = 94%,  reduce = 0%, Cumulative CPU 177.09 sec
2016-11-15 13:38:42,695 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 187.6 sec
2016-11-15 13:38:53,586 Stage-1 map = 100%,  reduce = 12%, Cumulative CPU 204.66 sec
2016-11-15 13:38:54,651 Stage-1 map = 100%,  reduce = 37%, Cumulative CPU 239.36 sec
2016-11-15 13:38:55,720 Stage-1 map = 100%,  reduce = 63%, Cumulative CPU 274.43 sec
2016-11-15 13:38:56,808 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 328.45 sec
MapReduce Total cumulative CPU time: 5 minutes 28 seconds 450 msec
Ended Job = job_1479182500047_0003
Launching Job 2 out of 5
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Cannot run job locally: Number of Input Files (= 17) is larger than
hive.exec.mode.local.auto.input.files.max(= 4)
```

```
Starting Job = job_1479182500047_0004, Tracking URL =
http://hadoop001:23188/proxy/application_1479182500047_0004/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1479182500047_0004
Hadoop job information for Stage-2: number of mappers: 5; number of reducers: 1
2016-11-15 13:39:10,862 Stage-2 map = 0%,  reduce = 0%
2016-11-15 13:39:15,210 Stage-2 map = 20%,  reduce = 0%, Cumulative CPU 7.67 sec
2016-11-15 13:39:18,392 Stage-2 map = 40%,  reduce = 0%, Cumulative CPU 11.41 sec
2016-11-15 13:39:21,576 Stage-2 map = 60%,  reduce = 0%, Cumulative CPU 14.4 sec
2016-11-15 13:39:24,851 Stage-2 map = 80%,  reduce = 0%, Cumulative CPU 18.34 sec
2016-11-15 13:39:25,915 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 19.7 sec
2016-11-15 13:39:31,236 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 26.47 sec
MapReduce Total cumulative CPU time: 26 seconds 470 msec
Ended Job = job_1479182500047_0004
Launching Job 3 out of 5
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Selecting local mode for task: Stage-3
Starting Job = job_1479182500047_0005, Tracking URL =
http://hadoop001:23188/proxy/application_1479182500047_0005/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1479182500047_0005
Hadoop job information for Stage-3: number of mappers: 2; number of reducers: 1
2016-11-15 13:39:43,982 Stage-3 map = 0%,  reduce = 0%
2016-11-15 13:39:46,088 Stage-3 map = 50%,  reduce = 0%, Cumulative CPU 4.74 sec
2016-11-15 13:39:47,149 Stage-3 map = 100%,  reduce = 100%, Cumulative CPU 5.76 sec
MapReduce Total cumulative CPU time: 5 seconds 760 msec
Ended Job = job_1479182500047_0005
Launching Job 4 out of 5
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Selecting local mode for task: Stage-4
Starting Job = job_1479182500047_0006, Tracking URL =
http://hadoop001:23188/proxy/application_1479182500047_0006/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1479182500047_0006
Hadoop job information for Stage-4: number of mappers: 1; number of reducers: 1
```

```
2016-11-15 13:39:59,223 Stage-4 map = 100%,  reduce = 0%, Cumulative CPU 1.31 sec
2016-11-15 13:40:00,289 Stage-4 map = 100%,  reduce = 100%, Cumulative CPU 2.01 sec
MapReduce Total cumulative CPU time: 2 seconds 10 msec
Ended Job = job_1479182500047_0006
Launching Job 5 out of 5
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Selecting local mode for task: Stage-5
Starting Job = job_1479182500047_0007, Tracking URL =
http://hadoop001:23188/proxy/application_1479182500047_0007/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1479182500047_0007
Hadoop job information for Stage-5: number of mappers: 1; number of reducers: 1
2016-11-15 13:40:11,475 Stage-5 map = 0%,  reduce = 0%
2016-11-15 13:40:12,534 Stage-5 map = 100%,  reduce = 100%, Cumulative CPU 2.15 sec
MapReduce Total cumulative CPU time: 2 seconds 150 msec
Ended Job = job_1479182500047_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 16  Reduce: 16   Cumulative CPU: 328.54 sec   HDFS Read: 4003080870
HDFS Write: 56737002 SUCCESS
Stage-Stage-2: Map: 5  Reduce: 1   Cumulative CPU: 26.47 sec   HDFS Read: 265825077 HDFS
Write: 2559106 SUCCESS
Stage-Stage-3: Map: 2  Reduce: 1   Cumulative CPU: 7.7 sec   HDFS Read: 3027535 HDFS Write:
756687 SUCCESS
Stage-Stage-4: Map: 1  Reduce: 1   Cumulative CPU: 2.01 sec   HDFS Read: 6064 HDFS Write:
496989 SUCCESS
Stage-Stage-5: Map: 1  Reduce: 1   Cumulative CPU: 2.15 sec   HDFS Read: 6472 HDFS Write:
497451 SUCCESS
Total MapReduce CPU Time Spent: 6 minutes 6 seconds 870 msec
OK
53810
Time taken: 123.253 seconds, Fetched: 1 row(s)
```

- 使用 MAPJOIN

```
hive (hahaha)> set hive.auto.convert.join=false;
hive (hahaha)> Select count(*) cnt
            > From store_sales ss
            >     join household_demographics hd on (ss.ss_hdemo_sk = hd.hd_demo_sk)
            >     join time_dim t on (ss.ss_sold_time_sk = t.t_time_sk)
```

```
           >     join store s on (s.s_store_sk = ss.ss_store_sk)
           > Where
           >     t.t_hour=8 and
           >     t.t_minute >= 30 and
           >     hd.hd_dep_count = 2
           > order by cnt;
Query ID = hadoop_20161115133557_eb0dc288-5a83-4c42-baac-669f30fd50c0
Total jobs = 2
Execution log at:
/tmp/hadoop/hadoop_20161115133557_eb0dc288-5a83-4c42-baac-669f30fd50c0.log
2016-11-15 13:36:09     Starting to launch local task to process map join;      maximum
memory = 4261937152
2016-11-15 13:36:12     Dump the side-table for tag: 1 with group count: 102 into file:
file:/tmp/hadoop/5bbcaa76-29a5-42c1-87f6-ec77a9324431/hive_2016-11-15_13-35-57_345_
8090821540652594783-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile01--.hashtable
2016-11-15 13:36:12     Uploaded 1 File to:
file:/tmp/hadoop/5bbcaa76-29a5-42c1-87f6-ec77a9324431/hive_2016-11-15_13-35-57_345_
8090821540652594783-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile01--.hashtable
(2104 bytes)
2016-11-15 13:36:12     Dump the side-table for tag: 1 with group count: 1800 into file:
file:/tmp/hadoop/5bbcaa76-29a5-42c1-87f6-ec77a9324431/hive_2016-11-15_13-35-57_345_
8090821540652594783-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile11--.hashtable
2016-11-15 13:36:12     Uploaded 1 File to:
file:/tmp/hadoop/5bbcaa76-29a5-42c1-87f6-ec77a9324431/hive_2016-11-15_13-35-57_345_
8090821540652594783-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile11--.hashtable
(36435 bytes)
2016-11-15 13:36:12     Dump the side-table for tag: 1 with group count: 720 into file:
file:/tmp/hadoop/5bbcaa76-29a5-42c1-87f6-ec77a9324431/hive_2016-11-15_13-35-57_345_
8090821540652594783-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile21--.hashtable
2016-11-15 13:36:12     Uploaded 1 File to:
file:/tmp/hadoop/5bbcaa76-29a5-42c1-87f6-ec77a9324431/hive_2016-11-15_13-35-57_345_
8090821540652594783-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile21--.hashtable
(14714 bytes)
2016-11-15 13:36:12     End of local task; Time Taken: 3.287 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
```

```
Cannot run job locally: Input Size (= 3998921296) is larger than
hive.exec.mode.local.auto.inputbytes.max (= 134217728)
Starting Job = job_1479182500047_0001, Tracking URL =
http://hadoop001:23188/proxy/application_1479182500047_0001/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1479182500047_0001
Hadoop job information for Stage-4: number of mappers: 15; number of reducers: 1
2016-11-15 13:36:31,077 Stage-4 map = 0%,  reduce = 0%
2016-11-15 13:36:42,947 Stage-4 map = 20%,  reduce = 0%, Cumulative CPU 21.47 sec
2016-11-15 13:36:44,018 Stage-4 map = 22%,  reduce = 0%, Cumulative CPU 28.26 sec
2016-11-15 13:36:45,082 Stage-4 map = 58%,  reduce = 0%, Cumulative CPU 84.59 sec
2016-11-15 13:36:46,156 Stage-4 map = 76%,  reduce = 0%, Cumulative CPU 87.13 sec
2016-11-15 13:36:47,230 Stage-4 map = 80%,  reduce = 0%, Cumulative CPU 88.87 sec
2016-11-15 13:36:49,362 Stage-4 map = 100%,  reduce = 0%, Cumulative CPU 110.82 sec
2016-11-15 13:36:53,873 Stage-4 map = 100%,  reduce = 100%, Cumulative CPU 112.81 sec
MapReduce Total cumulative CPU time: 1 minutes 52 seconds 810 msec
Ended Job = job_1479182500047_0001
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Selecting local mode for task: Stage-5
Starting Job = job_1479182500047_0002, Tracking URL =
http://hadoop001:23188/proxy/application_1479182500047_0002/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1479182500047_0002
Hadoop job information for Stage-5: number of mappers: 1; number of reducers: 1
2016-11-15 13:37:05,864 Stage-5 map = 0%,  reduce = 0%
2016-11-15 13:37:07,305 Stage-5 map = 100%,  reduce = 100%, Cumulative CPU 2.31 sec
MapReduce Total cumulative CPU time: 2 seconds 310 msec
Ended Job = job_1479182500047_0002
MapReduce Jobs Launched:
Stage-Stage-4: Map: 15  Reduce: 1   Cumulative CPU: 112.81 sec   HDFS Read: 4002958172
HDFS Write: 116 SUCCESS
Stage-Stage-5: Map: 1  Reduce: 1   Cumulative CPU: 2.31 sec   HDFS Read: 6467 HDFS Write:
496441 SUCCESS
Total MapReduce CPU Time Spent: 1 minutes 55 seconds 120 msec
OK
53810
Time taken: 72.396 seconds, Fetched: 1 row(s)
```

4.2.4 解释计划对比

● 未使用 MAPJOIN

```
hive (hahaha)> set hive.auto.convert.join=false;
hive (hahaha)> explain Select count(*) cnt
            > From store_sales ss
            >     join household_demographics hd on (ss.ss_hdemo_sk = hd.hd_demo_sk)
            >     join time_dim t on (ss.ss_sold_time_sk = t.t_time_sk)
            >     join store s on (s.s_store_sk = ss.ss_store_sk)
            > Where
            >     t.t_hour = 8 and
            >     t.t_minute >= 30 and
            >     hd.hd_dep_count = 2
            > order by cnt;
OK
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-3 depends on stages: Stage-2
  Stage-4 depends on stages: Stage-3
  Stage-5 depends on stages: Stage-4
  Stage-0 depends on stages: Stage-5

略
Time taken: 0.843 seconds, Fetched: 180 row(s)
```

● 使用 MAPJOIN

```
hive (hahaha)> set hive.auto.convert.join=true;
hive (hahaha)> explain Select count(*) cnt
            > From store_sales ss
            >     join household_demographics hd on (ss.ss_hdemo_sk = hd.hd_demo_sk)
            >     join time_dim t on (ss.ss_sold_time_sk = t.t_time_sk)
            >     join store s on (s.s_store_sk = ss.ss_store_sk)
            > Where
            >     t.t_hour = 8 and
            >     t.t_minute >= 30 and
            >     hd.hd_dep_count = 2
            > order by cnt;
OK
STAGE DEPENDENCIES:
  Stage-12 is a root stage
  Stage-4 depends on stages: Stage-12
  Stage-5 depends on stages: Stage-4
  Stage-0 depends on stages: Stage-5
略
Time taken: 0.652 seconds, Fetched: 141 row(s)
```

4.3 Parallel

4.3.1 特性介绍

PARALLEL 特性可以是的某些任务中的 stage 子任务可以一并行执行模式同时执行，相对于一直串行执行 stage 任务来说有效的提升资源利用率。

PARALLEL 特性主要针对如下几种情况：

- 多个数据源表关联
- 插入多个目标表
- UNION ALL

4.3.2 参数设置

- 如下参数设置是否开启自动转换为 PARALLEL

```
hive.exec.parallel
Default Value: false
Added In: Hive 0.5.0
Whether to execute jobs in parallel.  Applies to MapReduce jobs that can run in parallel, for example jobs processing different source tables before a join.  As ofHive 0.14, also applies to move tasks that can run in parallel, for example moving files to insert targets during multi-insert.
```

```
hive.exec.parallel.thread.number
Default Value: 8
Added In: Hive 0.6.0
How many jobs at most can be executed in parallel.
```

4.3.3 性能对比

测试 SQL 为 TPC-DS 中 q11，从对比来看，在使用 PARALLEL 特性之后，由原来的 743 秒减少到 600 秒，在并行任务数据量较大，集群资源较充足，计算较复杂的情况下，任务执行效率提升会更加明显。

SQL 内容如下：

```
with year_total as (
 select c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(ss_ext_list_price-ss_ext_discount_amt) year_total
       ,'s' sale_type
 from customer
     ,store_sales
     ,date_dim
 where c_customer_sk = ss_customer_sk
   and ss_sold_date_sk = d_date_sk
 group by c_customer_id
         ,c_first_name
         ,c_last_name
         ,c_preferred_cust_flag
         ,c_birth_country
         ,c_login
         ,c_email_address
         ,d_year
 union all
 select c_customer_id customer_id
       ,c_first_name customer_first_name
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
```

```
      ,sum(ws_ext_list_price-ws_ext_discount_amt) year_total
      ,'w' sale_type
 from customer
    ,web_sales
    ,date_dim
 where c_customer_sk = ws_bill_customer_sk
   and ws_sold_date_sk = d_date_sk
 group by c_customer_id
        ,c_first_name
        ,c_last_name
        ,c_preferred_cust_flag
        ,c_birth_country
        ,c_login
        ,c_email_address
        ,d_year
        )
 select
                t_s_secyear.customer_id
               ,t_s_secyear.customer_first_name
               ,t_s_secyear.customer_last_name
               ,t_s_secyear.customer_email_address
 from year_total t_s_firstyear
    ,year_total t_s_secyear
    ,year_total t_w_firstyear
    ,year_total t_w_secyear
 where t_s_secyear.customer_id = t_s_firstyear.customer_id
        and t_s_firstyear.customer_id = t_w_secyear.customer_id
        and t_s_firstyear.customer_id = t_w_firstyear.customer_id
        and t_s_firstyear.sale_type = 's'
        and t_w_firstyear.sale_type = 'w'
        and t_s_secyear.sale_type = 's'
        and t_w_secyear.sale_type = 'w'
        and t_s_firstyear.dyear = 2001
        and t_s_secyear.dyear = 2001+1
        and t_w_firstyear.dyear = 2001
        and t_w_secyear.dyear = 2001+1
        and t_s_firstyear.year_total > 0
        and t_w_firstyear.year_total > 0
        and case when t_w_firstyear.year_total > 0 then t_w_secyear.year_total /
t_w_firstyear.year_total else 0.0 end
            > case when t_s_firstyear.year_total > 0 then t_s_secyear.year_total /
t_s_firstyear.year_total else 0.0 end
 order by t_s_secyear.customer_id
        ,t_s_secyear.customer_first_name
        ,t_s_secyear.customer_last_name
        ,t_s_secyear.customer_email_address
limit 100;
```

由于 SQL 运行过程中日志信息内容过多，故只截取关键部分。

- 未使用 PARALLEL 特性

```
略
MapReduce Jobs Launched:
Stage-Stage-3: Map: 15  Reduce: 16   Cumulative CPU: 739.29 sec   HDFS Read: 4003246772
HDFS Write: 13748588 SUCCESS
Stage-Stage-10: Map: 6  Reduce: 6   Cumulative CPU: 165.05 sec   HDFS Read: 1513085545
HDFS Write: 576 SUCCESS
Stage-Stage-15: Map: 15  Reduce: 16   Cumulative CPU: 699.89 sec   HDFS Read: 4003249140
HDFS Write: 26495831 SUCCESS
Stage-Stage-21: Map: 6  Reduce: 6   Cumulative CPU: 166.3 sec   HDFS Read: 1513086229
HDFS Write: 576 SUCCESS
Stage-Stage-26: Map: 15  Reduce: 16   Cumulative CPU: 488.0 sec   HDFS Read: 4003252055
HDFS Write: 1536 SUCCESS
Stage-Stage-32: Map: 6  Reduce: 6   Cumulative CPU: 217.88 sec   HDFS Read: 1513086637
HDFS Write: 4719924 SUCCESS
Stage-Stage-37: Map: 15  Reduce: 16   Cumulative CPU: 478.58 sec   HDFS Read: 4003251265
HDFS Write: 1536 SUCCESS
Stage-Stage-43: Map: 6  Reduce: 6   Cumulative CPU: 211.87 sec   HDFS Read: 1513086271
HDFS Write: 4739457 SUCCESS
Stage-Stage-47: Map: 8   Cumulative CPU: 57.69 sec   HDFS Read: 172769643 HDFS Write:
4372580 SUCCESS
Stage-Stage-5: Map: 1  Reduce: 1   Cumulative CPU: 3.27 sec   HDFS Read: 662060 HDFS Write:
523870 SUCCESS
Total MapReduce CPU Time Spent: 53 minutes 47 seconds 820 msec
OK
Time taken: 743.388 seconds, Fetched: 100 row(s)
```

- 使用 PARALLEL 特性

```
略
Launching Job 1 out of 14
Launching Job 2 out of 14
Launching Job 3 out of 14
Launching Job 4 out of 14
Launching Job 5 out of 14
Launching Job 6 out of 14
Launching Job 7 out of 14
Launching Job 8 out of 14
略
MapReduce Jobs Launched:
Stage-Stage-26: Map: 15  Reduce: 16   Cumulative CPU: 526.66 sec   HDFS Read: 4003253807
HDFS Write: 1536 SUCCESS
Stage-Stage-37: Map: 15  Reduce: 16   Cumulative CPU: 499.5 sec   HDFS Read: 4003251281
HDFS Write: 1536 SUCCESS
Stage-Stage-3: Map: 15  Reduce: 16   Cumulative CPU: 731.74 sec   HDFS Read: 4003252802
HDFS Write: 13748588 SUCCESS
Stage-Stage-21: Map: 6  Reduce: 6   Cumulative CPU: 183.95 sec   HDFS Read: 1513087243
HDFS Write: 576 SUCCESS
Stage-Stage-43: Map: 6  Reduce: 6   Cumulative CPU: 215.95 sec   HDFS Read: 1513085941
HDFS Write: 4739457 SUCCESS
Stage-Stage-15: Map: 15  Reduce: 16   Cumulative CPU: 769.94 sec   HDFS Read: 4003252628
HDFS Write: 26495831 SUCCESS
Stage-Stage-32: Map: 6  Reduce: 6   Cumulative CPU: 216.31 sec   HDFS Read: 1513086979
HDFS Write: 4719924 SUCCESS
Stage-Stage-10: Map: 6  Reduce: 6   Cumulative CPU: 174.82 sec   HDFS Read: 1513087231
HDFS Write: 576 SUCCESS
Stage-Stage-47: Map: 8   Cumulative CPU: 62.69 sec   HDFS Read: 179400672 HDFS Write:
4451516 SUCCESS
Stage-Stage-5: Map: 1  Reduce: 1   Cumulative CPU: 3.19 sec   HDFS Read: 662038 HDFS Write:
523977 SUCCESS
Total MapReduce CPU Time Spent: 56 minutes 24 seconds 750 msec
Time taken: 600.607 seconds, Fetched: 100 row(s)
```

5 文件格式

5.1 内部表

- 经验总结如下(不同数据文件有一定误差)：

ORC 比 PARQUET 的压缩率高大约 10~20%。

(TEXT:10G -> ORC:2~3G or PARQUET:3~4G)

PARQUET 比 ORC 的执行效率高大约 10~20%。

- Hive 中内部表推荐使用存储格式 ORC->PARQUET->RCFILE-> TEXTFILE

- 如果使用 Spark 引擎使用较多，内部表定义推荐使用存储格式 PARQUET –> ORC ->RCFILE-> TEXTFILE

- ORC 建表语句

```
create table Addresses (
  name string,
  street string,
  city string,
  state string,
  zip int)
PARTITIONED BY (MONTH_PART STRING comment '月份分区')
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS ORC stored as orc tblproperties ("orc.compress"="SNAPPY");
```

- PARQUET 建表语句

```
CREATE TABLE parquet_test (
 id int,
 str string,
 mp MAP<STRING,STRING>,
 lst ARRAY<STRING>,
 strct STRUCT<A:STRING,B:STRING>)
PARTITIONED BY (MONTH_PART STRING comment '月份分区')
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS PARQUET;
```

5.2 外部表

因为 Hive 采取的是读时校验，所以建立外部表是指定的存储格式需要和入库文件保持一致。

一般默认为 TEXTFILE.若外部表建表指定为其他格式(非 TEXTFILE)那么读取时会报错。

```
create external table Addresses (
  name string,
  street string,
  city string,
  state string,
  zip int)
```

```
PARTITIONED BY (MONTH_PART STRING comment '月份分区')
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS TEXTFILE;
```

如果对于入库及接口表加工追求极致性能，可以通过自定义 JAVA 程序将 TEXTFILE 加工成 ORC 或者 PARQUET 文件同时写入 HDFS.

6 压缩格式

我们将数据压缩格式的使用环境分为入库、加工、出库三个部分来分别阐述，同时结合文件存储格式来进行细分说明。

常用压缩格式为 SNAPPY,GZIP,BZIP2,但是对于使用场景来说非常严格，切勿乱使用。

1、 特殊情况下考虑 BZIP2，对文件压缩率有很高要求，主要用来应对文件网络传输。目前主要用途是生成上报文件，低带宽下进行传输使用，其他地方不用考虑了。大数据平台上几乎无用途。因为压缩率的问题可以通过文件存储格式来解决（无论是使用 ORC 存储还是 PARQUET，都能将文件的空间占用大小减少到一个合适的范围。）

2、 GZIP 的场景主要在入库时，处理 GZIP 时由于 gzip 命令较为常用，并且 gzip 文件可以直接多个进行叠加不影响读取的特定，所以用于在接口机合并小文件只用，后面再详细阐述。

3、 SNAPPY 文件的使用范围非常广泛。后面详细讲。

可使用文件压缩格式如下：

```
<property>
    <name>io.compression.codecs</name>
    <value>org.apache.hadoop.io.compress.DefaultCodec,org.apache.hadoop.io.compress.
GzipCodec,org.apache.hadoop.io.compress.BZip2Codec,org.apache.hadoop.io.compress.Sn
appyCodec</value>
```

```
</property>
```

- 为什么选择 SNAPPY?

1、 SNAPPY 的压缩和解压缩效率基本所有场景下都是最高的，不会过多的占用有限的 CPU 资源。

2、 SNAPPY 不会影响 Map 的分片和 Reduce 合并，能够有效的避免资源利用不足和过多小文件。

3、 SNAPPY 的压缩比，虽然比其他压缩格式来说确实较小。但是压缩起到重要作用的地方是 MapReduce 中间各 Stage 层的磁盘读写。文件系统上长期存储的文件，可以通过设置文件格式为 ORC 或者 PARQUET 达到很好的空间利用率。

6.1 入库

入库：数据由本地文件系统上传到 HDFS，或者通过 hive load 命令加载本地文件数据。

上传文件格式需与 Hive 中表的存储定义格式匹配。

入库文件的压缩格式要求并不是特别严格，主要就是压缩之后上传文件的大小是否合适。单个文件太大会导致单个 Map 处理时间过长，太小又会导致占用太多 container，浪费过多计算资源，Map 启动的代价过高，过多的 Map 导致任务执行时间都浪费在 Container 的申请和 Map 的启动上了。

由于 GZIP 文件在 Map 处理时并不会自动切分，也就是说一个 GZIP 文件对应一个 Map(如果你处理单个 100G 的 GZIP 文件，那么一个 4G 内存的 Map 处理一个 100G 的压缩文件，效率之低突破天际)。

6.2 加工

加工：来源表和目标表的存储都在 Hive 或者 HDFS

6.2.1 Map 阶段

现在默认使用 SNAPPY.

```
<property>
   <name>mapreduce.map.output.compress</name>
   <value>true</value>
</property>

<property>
   <name>mapreduce.map.output.compress.codec</name>
   <value>org.apache.hadoop.io.compress.SnappyCodec</value>
</property>
```

6.2.2 Reduce 阶段

现在默认使用 SNAPPY 压缩格式.

```
<property>
   <name>mapreduce.output.fileoutputformat.compress.type</name>
   <value>BLOCK</value>
</property>

<property>
   <name>mapreduce.output.fileoutputformat.compress</name>
   <value>true</value>
</property>

<property>
   <name>mapreduce.output.fileoutputformat.compress.codec</name>
   <value>org.apache.hadoop.io.compress.SnappyCodec</value>
</property>
```

备用参数

```
set hive.exec.compress.output=true;--控制 hive 最终生成结果为压缩。
set hive.exec.compress.intermediate=true;--控制 hive 中间中间各 Stage 层结果是否为压缩。
```

6.3 出库

出库：将 HDFS 上 Hive 中数据通过 Insert 方式导出至本地文件系统或者直接将 HDFS 文件 get 到本地。

- 使用 Insert 的方式进行数据导出，则会启动 MapReduce 任务，这时候可以忽略来源表的存储格式和压缩格式，但是可以设置输出文件的压缩格式，但是个数由表中文件的个数决定（也就是 Map 数目）。
- 使用 hadoop fs -get 方式来将 HDFS 文件拖拽到本地时，由于命令只是单纯的拷贝，所以来源表中的存储格式和压缩格式不会改变。如果你需要 get 出来的文件是 TEXTFILE 文件那么需要在来源表创建定义时进行设置。如果你需要 get 出来的文件是 GZIP 压缩，那么需要指定 Hive 输出为压缩且设置压缩编码为 GZIP。

7 小文件合并

当大量小文件，文件数目非常多的时候，容易使得 HDFS 上块数目相对过多，给 Namenode 带来压力，影响 HDFS 文件读写性能，而且导致 MapReduce 任务执行效率降低。因此，在 MapReduce 任务的 Map 阶段和 Reduce 阶段都要进行文件合并，一方面可以更加合理的利用集群资源，另一方面提高任务的执行效率。

7.1 入库合并

比如这里总共 500G 的 GZIP 文件需要入库。下面几种方式来处理：

单个文件	文件个	Map 个	执行时	问题及说明

大小	数	数	间	
50G	10	10	20X	启动 Map 太少，单个 Map 执行时间异常长，整体资源利用很低，任务执行时间异常长。
5G	100	100	3X	启动 Map 较少，单个 Map 执行时间较长，整体资源利用率低，任务执行时间很长。
1G	500	500	X	启动 Map 合适，单个 Map 在 4G 内存的 Map 下执行合适，整体资源利用率较好，任务执行时间可接受。
250M	2000	2000	X	启动 Map 较多，单个 Map 在 1~2G 内存的 Map 下执行时间合适，整体资源利用率高，任务执行时间可接受。
20M	25000	25000	20X	启动 Map 太多，单个 Map 执行时间段，启动过多 Map 导致时间都浪费在 Container 申请和 Map 任务启动上。资源利用率异常高，导致其他任务无法获取 container，任务执行时间异常长。

7.2 MAP 合并

Map 阶段如果使用合适文件格式和压缩方式，比如 input 读取的都是小文件，Map 会自动进行合并，一般会合并至接近块大小。

```
hive.merge.mapfiles
Default Value: true
Added In: Hive 0.4.0
Merge small files at the end of a map-only job.
```

```
hive.mergejob.maponly
Default Value: true
Added In: Hive 0.6.0
Removed In: Hive 0.11.0
Try to generate a map-only job for merging files if CombineHiveInputFormat is
supported. (This configuration property was removed in release 0.11.0.)
```

```
hive.merge.size.per.task
Default Value: 256000000
Added In: Hive 0.4.0
Size of merged files at the end of the job.
```

```
hive.merge.smallfiles.avgsize
Default Value: 16000000
默认值略小，可以考虑增加到 100000000
Added In: Hive 0.5.0
```

```
When the average output file size of a job is less than this number, Hive will start an additional map-reduce job to merge the output files into bigger files. This is only done for map-only jobs if hive.merge.mapfiles is true, and for map-reduce jobs if hive.merge.mapredfiles is true.
```

```
备用参数
set mapred.max.split.size=256000000;
set mapred.min.split.size.per.node=100000000;
set mapred.min.split.size.per.rack=100000000;
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

7.3 REDUCE 合并

Reduce 阶段如果不控制合并小文件，有如下影响：

- 大量小文件对于 NameNode 造成压力
- Reduce 任务多造成集群资源浪费
- 文件作为来源表继续上层加工时可能出现过多 Map，造成集群资源浪费

```
hive.merge.mapredfiles
Default Value: false
Added In: Hive 0.4.0
Merge small files at the end of a map-reduce job.
```

```
hive.merge.size.per.task
Default Value: 256000000
Added In: Hive 0.4.0
Size of merged files at the end of the job.
```

```
hive.merge.smallfiles.avgsize
Default Value: 16000000
默认值略小，可以考虑增加到 100000000
Added In: Hive 0.5.0
When the average output file size of a job is less than this number, Hive will start an additional map-reduce job to merge the output files into bigger files. This is only done for map-only jobs if hive.merge.mapfiles is true, and for map-reduce jobs if hive.merge.mapredfiles is true.
```

```
备用参数
set mapred.reduce.tasks=n;  (n 为整数)
```

可以通过控制 Reduce 数目来达到合并小文件的作用，如果生成了 100 个几 M 的文件，可以将 reduce 数目设置为 1，这样最后会生成一个几百 M 的文件，对于整体流程来说有极大的优化作用。

7.4 RCFILE 和 ORC 文件合并

7.4.1 命令

Alter Table/Partition Concatenate

Version information

In Hive release 0.8.0 RCFile added support for fast block level merging of small RCFiles using concatenate command. In Hive release 0.14.0 ORC files added support fast stripe level merging of small ORC files using concatenate command.

ALTER TABLE table_name [PARTITION (partition_key = 'partition_value' [, ...])] CONCATENATE;

If the table or partition contains many small RCFiles or ORC files, then the above command will merge them into larger files. In case of RCFile the merge happens at block level whereas for ORC files the merge happens at stripe level thereby avoiding the overhead of decompressing and decoding the data.

7.4.2 测试

- ORC 文件格式

```
[hadoop@ hadoop]$ hadoop fs -ls /user/hive/warehouse/test.db/user_info_orc/
Found 2 items
-rwxr-xr-x   1 hadoop supergroup        349 2016-11-16 12:52
/user/hive/warehouse/test.db/user_info_orc/000000_0
-rwxr-xr-x   1 hadoop supergroup        349 2016-11-16 12:53
hdfs://breath:9000/user/hive/warehouse/test.db/user_info_orc/000000_0_copy_1
[hadoop@ hadoop]$ hive -e "Alter Table test.user_info_orc Concatenate;"
Logging initialized using configuration in
file:/opt/beh/core/hive/conf/hive-log4j.properties
Starting Job = job_1478583841022_0009, Tracking URL =
http://breath:23188/proxy/application_1478583841022_0009/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1478583841022_0009
Hadoop job information for null: number of mappers: 1; number of reducers: 0
2016-11-16 12:54:35,790 null map = 0%,  reduce = 0%
2016-11-16 12:54:45,806 null map = 100%,  reduce = 0%, Cumulative CPU 4.37 sec
MapReduce Total cumulative CPU time: 4 seconds 370 msec
Ended Job = job_1478583841022_0009
Loading data to table test.user_info_orc
```

```
Table test.user_info_orc stats: [numFiles=1, numRows=0, totalSize=546, rawDataSize=0]
MapReduce Jobs Launched:
Stage-null: Map: 1   Cumulative CPU: 4.37 sec   HDFS Read: 3184 HDFS Write: 238118 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 370 msec
OK
Time taken: 30.898 seconds
[hadoop@ hadoop]$ hadoop fs -ls /user/hive/warehouse/test.db/user_info_orc/
Found 1 items
-rwxr-xr-x   1 hadoop supergroup        546 2016-11-16 12:54
/user/hive/warehouse/test.db/user_info_orc/000000_0
```

● RCFILE 文件格式

```
[hadoop@ hadoop]$ hadoop fs -ls /user/hive/warehouse/test.db/user_info_rcfile/
Found 2 items
-rwxr-xr-x   1 hadoop supergroup        109 2016-11-16 12:55
/user/hive/warehouse/test.db/user_info_rcfile/000000_0
-rwxr-xr-x   1 hadoop supergroup        109 2016-11-16 12:55
/user/hive/warehouse/test.db/user_info_rcfile/000000_0_copy_1

[hadoop@ hadoop]$ hive -e "Alter Table test.user_info_rcfile Concatenate;"
Starting Job = job_1478583841022_0010, Tracking URL =
http://breath:23188/proxy/application_1478583841022_0010/
Kill Command = /opt/beh/core/hadoop/bin/hadoop job  -kill job_1478583841022_0010
Hadoop job information for null: number of mappers: 1; number of reducers: 0
2016-11-16 12:56:17,241 null map = 0%,  reduce = 0%
2016-11-16 12:56:18,329 null map = 100%,  reduce = 0%, Cumulative CPU 1.86 sec
MapReduce Total cumulative CPU time: 1 seconds 860 msec
Ended Job = job_1478583841022_0010
Loading data to table test.user_info_rcfile
Table test.user_info_rcfile stats: [numFiles=1, numRows=0, totalSize=162,
rawDataSize=0]
MapReduce Jobs Launched:
Stage-null: Map: 1   Cumulative CPU: 1.86 sec   HDFS Read: 1780 HDFS Write: 237757 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 860 msec
OK
Time taken: 19.671 seconds
[hadoop@ hadoop]$ hadoop fs -ls /user/hive/warehouse/test.db/user_info_rcfile/
Found 1 items
-rwxr-xr-x   1 hadoop supergroup        162 2016-11-16 12:56
/user/hive/warehouse/test.db/user_info_rcfile/000000_0
```

下面参数用来调整 ORC 文件合并的大小

```
hive.exec.orc.default.stripe.size
Default Value: 256*1024*1024 (268,435,456) in 0.13.0;
                 64*1024*1024 (67,108,864) in 0.14.0
Added In: Hive 0.13.0 with HIVE-5425; default changed in 0.14.0
with HIVE-7231 and HIVE-7490
Define the default ORC stripe size, in bytes.
```

- TEXTFILE 文件格式

在 Hive0.14 之后版本中测试目前只有 RCFILE 和 ORC 文件格式支持 Concatenate 合并。

```
[hadoop@ hadoop]$ Alter Table user_info_textfile Concatenate;
FAILED: SemanticException org.apache.hadoop.hive.ql.parse.SemanticException: Only
RCFile and ORCFile Formats are supportted right now.
```

8 资源调度优化

8.1 任务资源设置

原则上最优为：给特定的数据以合适的资源。过多会浪费，过少运行慢。

8.1.1 调度器

需要注意的是，默认情况，各个调度器只会对内存资源进行调度，不会考虑 CPU 资源，你需要在调度器配置文件中进行相关设置

也就是说在 Capacity Scheduler 中默认 DefaultResourceCalculator 下，通过

set mapreduce.map.cpu.vcores=2;

并不会使得每个 Map 任务可以使用 2 个 vcore(Reduce 同理，以前没怎么注意，Fair Scheduler 未测试)。

```
<property>
    <name>yarn.scheduler.capacity.resource-calculator</name>
    <value>org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator</value>
    <description>
      The ResourceCalculator implementation to be used to compare
      Resources in the scheduler.
      The default i.e. DefaultResourceCalculator only uses Memory while
      DominantResourceCalculator uses dominant-resource to compare
      multi-dimensional resources such as Memory, CPU etc.
    </description>
</property>
```

8.1.2 MapReduce 内存

比如我们的 Map 输入是 100 个 250M 大小的 GZIP 文件。我们已知解压文件之后大约 TXT 空间占用为 1.5G。我们通过设置 Map 内存来看看如何分析,CPU 暂不考虑默认为 1vcore。由于是 GZIP 文件，Map 数为 100 个。

单个 Map 分配内存	问题及说明
8G	Map 整体占用集群内存过多，服务器的实际内存利用率很低，造成资源浪费。每个 Map 的内存其实没有充分利用，8G 的 container 中只有不到一半被利用。
4G	单个 Map 可以内存容纳所有数据，对于出现负责 combine 操作时，避免数据落地。占用集群资源可接受，运行效率也较高。
2G	单个 Map 可以内存容纳所有数据，对于简单的 map 计算，已经足够。占用集群资源少，运行效率较高。
1G	单个 Map 内存偏少，在无复杂 Map 计算时也能保证运行效率，运行效率一般。

对于 Map 任务来说，内存在 1~4G 之间可以选择合适的值，既要保证 Map 任务的效率，又不会对集群资源造成过度浪费。

对于 Reduce 任务来说，由于考虑到 Reduce 合并及可能出现的数据倾斜问题，内存一般需要比 Map 内存设置的高一些，2~8G 之间可以选择合适的值，设置一个合适的值既能保证 Reduce 任务不会出现内存溢出的错误，又不会对集群资源造成过度浪费。

```
<property>
   <name>mapreduce.map.memory.mb</name>
   <value>2048</value>
</property>
```

```
<property>
   <name>mapreduce.reduce.memory.mb</name>
   <value>2048</value>
</property>

<property>
   <name>yarn.app.mapreduce.am.resource.mb</name>
   <value>2048</value>
   <description>The amount of memory the MR AppMaster needs.</description>
</property>

<property>
   <name>mapred.child.java.opts</name>
   <value>-Xmx2048m</value>
</property>

<property>
   <name>mapreduce.map.java.opts</name>
   <value>-Xmx2048m</value>
</property>

<property>
   <name>mapreduce.reduce.java.opts</name>
   <value>-Xmx2048m</value>
</property>

<property>
   <name>yarn.app.mapreduce.am.command-opts</name>
   <value>-Xmx2048m</value>
</property>

<property>
   <name>yarn.app.mapreduce.am.admin-command-opts</name>
   <value>-Xmx2048m</value>
</property>
```

8.2 集群资源设置

最小可申请资源量内存：yarn.scheduler.minimum-allocation-mb

最大可申请资源量内存：yarn.scheduler.maximum-allocation-mb

最小可申请资源量 vcore：yarn.scheduler.minimum-allocation-vcores

最大可申请资源量 vcore：yarn.scheduler.maximum-allocation-vcores

程序申请的内存满足一下两个条件：

- 若应用程序申请的资源量不大于最小可申请资源量，则 YARN 会将其大小改为最小可申请量，也就是说，应用程序获得资源不会小于最小可申请资源，但也不一定相等。

  Map：Max(yarn.scheduler.minimum-allocation-mb, mapreduce.map.memory.mb)
  Reduce：Max(yarn.scheduler.minimum-allocation-mb, mapreduce.reduce.memory.mb)

- 如果应用程序申请的资源量大于最大可申请资源量，则会抛出异常，无法申请成功；

  Map： mapreduce.map.memory.mb < yarn.scheduler.maximum-allocation-mb
  Reduce：mapreduce.reduce.memory.mb < yarn.scheduler.maximum-allocation-mb

如下图模拟情况

最小可申请内存	最大可申请内存	程序申请内存	情况描述	实际使用内存
1024	4096	2048	程序申请介于最大和最小之间	2048
1024	4096	512	程序申请低于最小	1024
1024	4096	8192	程序申请高于最大	报错

最小可申请内存

```
<property>
<name>yarn.scheduler.minimum-allocation-mb</name>
    <value>1024</value>
</property>
```

最大可申请内存

```
<property>
     <name>yarn.scheduler.maximum-allocation-mb</name>
     <value>8196</value>
</property>
```

最小可申请 vcore

```
<property>
<name>yarn.scheduler.minimum-allocation-vcores</name>
    <value>1</value>
```

```
</property>
```

最小可申请 vcore

```
<property>
<name>yarn.scheduler.maximum-allocation-vcores</name>
    <value>8</value>
</property>
```

8.3 Uber 模式

Uber 模式是 Yarn 上针对 MR 小作业的优化机制， 如果 job 任务足够小，则直接让任务串行地在 MRAppMaster 完成，这样整个 Application 只会使用一个 Container，相对于分配多个 Container 来说执行效率要高很多。

对比类型	Uber 模式	非 Uber 模式
AppMaster 个数	1	1
AppMaster 占用 container 个数	1	1
Map 个数	1~9	1~9
Map 占用 container 个数	0	同上(根据 Map 数而定)
Reduce 个数	0~1	0~1
Reduce 占用 container 个数	0	同上(根据 Reduce 数而定)
总共占用 Container 个数	1	AppMaster+Map+Reduce 之和

那么什么才是足够小的 Job 呢？满足如下参数设置：

- mapreduce.job.ubertask.maxmaps 最大的 map 数。默认值 9
- mapreduce.job.ubertask.maxreduces 最大的 reduce 数，默认为 1
- mapreduce.job.ubertask.maxbytes 最大的字节数，默认和 dfs.block.size 一样

Uber 任务执行还应满足如下条件：

- yarn.app.mapreduce.am.resource.mb 大于 mapreduce.map.memory.mb
- yarn.app.mapreduce.am.resource.mb 大于 mapreduce.reduce.memory.mb
- yarn.app.mapreduce.am.resource.cpu-vcores 大于 mapreduce.map.cpu.vcores
- yarn.app.mapreduce.am.resource.cpu-vcores 大于 mapreduce.reduce.cpu.vcores

启动 Uber 优化模式

```
<property>
   <name>mapreduce.job.ubertask.enable</name>
   <value>true</value>
   <description>Whether to enable the small-jobs "ubertask" optimization, which runs "sufficiently small" jobs sequentially within a single JVM. "Small" is defined by the following maxmaps, maxreduces, and maxbytes settings. Note that configurations for application masters also affect the "Small" definition - yarn.app.mapreduce.am.resource.mb must be larger than both mapreduce.map.memory.mb and mapreduce.reduce.memory.mb, and yarn.app.mapreduce.am.resource.cpu-vcores must be larger than both mapreduce.map.cpu.vcores and mapreduce.reduce.cpu.vcores to enable ubertask. Users may override this value.
   </description>
</property>
```

设置 Uber 模式的 Map 数限制

```
<property>
   <name>mapreduce.job.ubertask.maxmaps</name>
   <value>9</value>
   <description>Threshold for number of maps, beyond which job is considered too big for the ubertasking optimization. Users may override this value, but only downward.</description>
</property>
```

设置 Uber 模式的 Reduce 数限制

```
<property>
   <name>mapreduce.job.ubertask.maxreduces</name>
   <value>1</value>
   <description>Threshold for number of reduces, beyond which job is considered too big for the ubertasking optimization. CURRENTLY THE CODE CANNOT SUPPORT MORE THAN ONE REDUCE and will ignore larger values. (Zero is a valid max, however.) Users may override this value, but only downward.</description>
</property>
```

设置 Uber 模式下输入数据量最大值

```
<property>
   <name>mapreduce.job.ubertask.maxbytes</name>
   <value>134217728</value>
   <description>Threshold for number of input bytes, beyond which job is considered too big for the ubertasking optimization. If no value is specified, dfs.block.size is used as a default. Be sure to specify a default value in mapred-site.xml if the underlying filesystem is not HDFS. Users may override this value, but only downward.</description>
</property>
```

设置 Uber 模式下的环境变量

```
<property>
   <name>yarn.app.mapreduce.am.env</name>
   <value>LD_LIBRARY_PATH=/opt/beh/core/hadoop/lib/native</value>
</property>
<property>
   <name>yarn.app.mapreduce.am.admin.user.env</name>
   <value>LD_LIBRARY_PATH=/opt/beh/core/hadoop/lib/native</value>
</property>
```

8.4 Reduce 的延迟启动

```
<property>
    <name>mapreduce.job.reduce.slowstart.completedmaps</name>
    <value>0.05</value>
<description> Fraction of the number of maps in the job which should be complete before reduces are scheduled for the job.</description>
</property>
```

mapreduce.job.reduce.slowstart.completedmaps 参数的主要作用是限制某个 MapReduce 任务中 Reduce 启动的实际，参数值为在 Map 完成多少之后启动 Reduce 任务。

对于计算量较大的任务，Map 和 Reduce 书目非常多(比如一个任务即用尽集群大部分 Container)，那么参数 mapreduce.job.reduce.slowstart.completedmaps 应设置为 1.0。避免 Map 和 Reduce 过多的任务出现 Map 未执行完，Reduce 任务先开始执行，过多任务并行时所需资源相互等待的窘境。更有甚至任务之间互相等待释放 container，互相又不释放 container，形成资源竞争死锁。

对于计算量较小的任务，任务占用资源相对于集群总资源相对很少，可以适当将此值减少。 如此，任务执行时，部分 Map 接近完毕，Reduce 即启动（最优时间为在 Redcue 任务开始获取最后一个 Map 数据的时候，最后一个 Map 刚刚执行完毕），这样集群的计算资源能够充分被 Map 和 Reduce 任务利用。

如果集群的并发任务较多，集群资源利用率总是维持在满载状态(即 container 全部分配完毕)，那么从集群和流程稳定性的角度考虑，此值设为 1.0

8.5 推测执行

推测执行(Speculative Execution)是指在 MpaReduce 的集群环境下，因为 Nodemanager 硬件配置不同，负载不平均等原因，使得不同 Nodemanger 服务器上的 Map 或者 Reduce 任务的执行时间差距较大，有的 Map、Redcue 任务运行时间明显比其他任务长。这些执行时间长的任务会明显拖慢了整个 MapReduce 的执行进度，为了避免这种情况发生，推测执行会为该慢任务启动备份任务，让该 speculative task 与原始 task 同时处理一份数据，哪个先运行完，则将谁的结果作为最终结果。

由于目前各生产集群的硬件配置比较接近，所以 Map 或者 Reduce 任务执行的时间比较接近，一般不需要推测执行任务。如果集群资源利用率一直非常高甚至满载，推测执行任务 speculative task 会加剧集群资源的繁忙，所以推测执行不建议使用。

默认情况启动，直接设置为关闭状态。

```
<property>
   <name>mapreduce.map.speculative</name>
   <value>false</value>
</property>
```

```
<property>
   <name>mapreduce.reduce.speculative</name>
   <value>false</value>
</property>
```

8.6 规整化因子

规整化因子是用来规整化应用程序资源的，应用程序申请的资源如果不是该因子的整数倍，

则将被修改为大于或者等于指定表达式的最小整数。

公式为 ceil(a/b)*b，其中 a 是应用程序申请的资源，b 为规整化因子。

下面模拟内存规整化因子为 1024 内存的情况：

程序任务申请内存	任务实际使用内存	情况描述
512	1024	ceil(512/1024)*1024
1024	1024	ceil(512/1024)*1024
1025	2048	ceil(1025/1024)*1024;
1500	2048	ceil(1500/1024)*1024;
2000	2048	ceil(2000/1024)*1024;

默认情况下内存规整化因子为 1024，可以调整为 512 或保持默认。

如果需要对 Map 和 Reduce 使用内存绝对控制，那么可以将此参数设置为 200 或 100。

内存规整化因子

```
<property>
   <name>yarn.scheduler.increment-allocation-mb</name>
   <value>1024</value>
</property>
```

vcore 规整化因子

```
<property>
   <name>yarn.scheduler.increment-allocation-vcores</name>
   <value>1</value>
</property>
```

9 流程优化

9.1 拆分可并发子流程

在 SHELL 过程中存在多段 SQL 语句，SQL1 与 SQL2 无依赖关系，SQL3 依赖于 SQL1 和 SQL2. 一般情况下，SHELL 脚本中 SQL1、SQL2、SQL3 串行执行， 如果 SQL1 和 SQL2 都是执行时间较长的语句，那么应该是的 SQL1 和 SQL2 并行执行(parallel)。

```
SHELL(SQL1   -> SQ2    -> SQL3)
```

9.1.1 优化方法一

将 SQL1 装入 SHELL1 中,将 SQL2 装入 SHELL2,将 SQL3 装入 SHELL3;通过 ETL 流程调度 SHELL1

和 SHELL2 并行执行，然后执行 SHELL3

```
SHELL1  (SQL1) =>          |
                            |          => SHELL3(SQL3)
SHELL2  (SQL2) =>          |
```

9.1.2 优化方法二

使用 SHELL 中 for 循环加入并行来运行子进程。下位图示及相关代码示例。

```
SHELL    (SQL1) =>          |              )
Parallel (                  |     =>    SQL3)
          (SQL2) =>         |              )
```

简单代码示例

```
#!/bin/bash
SQL1="INSERT INTO TAB_MID1 SELECT * FROM TAB_SOURCE1"
SQL2="INSERT INTO TAB_MID2 SELECT * FROM TAB_SOURCE2"
for i in {1..2}
do
```

```
{
eval HQL=\$SQL${i}
echo $HQL
hive -e "$HQL"
} &
done
wait
hive -e "USE DB;
INSERT INTO TAB_TARGET
SELECT * FROM TAB_MID1 M1
JOIN TAB_MID2 M2
ON M1.K=M2.K"
```

9.2 合并子流程

多段插入的 SQL 可以通过 UNION ALL 合并一个 SQL，增加集群资源利用率。

```
SHELL1:
INSERT INTO TABLE TAB_TARGET SELECT * FROM (SQL);
INSERT INTO TABLE TAB_TARGET SELECT * FROM (SQL2);
```

转化为

```
SHELL2:
INSERT INTO TABLE TAB_TARGET
SELECT * FROM (SQL);
UNION ALL
SELECT * FROM (SQL2);
```

此项优化与 hive.exec.parallel 相关

10 SQL 优化

我们知道了性能低下的根源，同样，我们也可以从 Hive 的配置解读去优化。Hive 系统内部已针对不同的查询预设定了优化方法，用户可以通过调整配置进行控制， 以下举例介绍部分优化的策略以及优化控制选项

10.1 分区裁剪

去除查询中不必要的分区。 关联和 WHERE 条件中尽量使用分区键，不要使用其他的同含义字段。可以减少读入的分区数目。有效的减少 Map 读入的数据量。

下面例子里的 MONTH_ID 和 MONTH_PART 的数据虽然相同，SQL 中使用 **MONTH_PART** 而不要使用 **MONTH_ID**

```
create table Addresses (
  MONTH_ID string,
  name string,
  street string,
  city string,
  state string,
  zip int)
PARTITIONED BY (MONTH_PART STRING comment '月份分区')
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'
NULL DEFINED AS ''
STORED AS ORC stored as orc tblproperties ("orc.compress"="SNAPPY");
```

10.2 GROUP BY 数据倾斜

10.2.1 参数设置

使用 GROUP BY 有数据倾斜时进行负载均衡:

```
hive.groupby.skewindata=true;
hive.groupby.skewindata
Default Value: false
Added In: Hive 0.3.0
Whether there is skew in data to optimize group by queries.

hive.groupby.mapaggr.checkinterval
Default Value: 100000
Added In: Hive 0.3.0
Number of rows after which size of the grouping keys/aggregation classes is performed.
```

10.2.2 特性介绍

解释计划会成成两个 MapReduce 任务：

在第一个 MapReduce 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，这样处理之后，相同的 Group By Key 有可能分发到不同的 reduce 中，从而达到负载均衡的目的

在第二个 MapReduce 任务再根据第一步中处理的数据按照 Group By Key 分布到 reduce 中，（这一步中相同的 Key 在同一个 Reduce 中），最终生成聚合操作结果。

10.2.3 过程对比

- 未使用参数配置

```
hive (hahaha)> set hive.groupby.skewindata=false;
hive (hahaha)> select i_item_sk,count(distinct i_item_id),count(distinct i_brand_id)
from item group by i_item_sk limit 10;
Automatically selecting local only mode for query
Query ID = hadoop_20161116102127_795416ae-56ec-4ed0-92b3-5511fe8a6c9f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2016-11-16 10:21:31,479 Stage-1 map = 0%,  reduce = 0%
2016-11-16 10:21:34,493 Stage-1 map = 100%,  reduce = 0%
2016-11-16 10:21:35,505 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local2137032971_0002
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 115421334 HDFS Write: 91 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
```

```
1       1       1
2       1       1
3       1       1
4       1       1
5       1       1
6       1       1
7       1       1
8       1       1
9       1       1
10      1       1
Time taken: 8.0 seconds, Fetched: 10 row(s)
```

- 未使用参数配置

```
hive (hahaha)> set hive.groupby.skewindata=true;
hive (hahaha)> select i_item_sk,count(distinct i_item_id) from item group by i_item_sk
limit 10;
Automatically selecting local only mode for query
Query ID = hadoop_20161116102156_733079f4-508d-42c4-9f82-254323f4ba6f
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2016-11-16 10:21:58,232 Stage-1 map = 0%,  reduce = 0%
2016-11-16 10:22:01,249 Stage-1 map = 100%,  reduce = 0%
2016-11-16 10:22:02,259 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1263714272_0003
Launching Job 2 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2016-11-16 10:22:04,341 Stage-2 map = 0%,  reduce = 0%
2016-11-16 10:22:05,352 Stage-2 map = 100%,  reduce = 100%
```

```
Ended Job = job_local1082580567_0004
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 173132098 HDFS Write: 2199987 SUCCESS
Stage-Stage-2:  HDFS Read: 177531968 HDFS Write: 4399877 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
1       1
2       1
3       1
4       1
5       1
6       1
7       1
8       1
9       1
10      1
Time taken: 10.494 seconds, Fetched: 10 row(s)
```

- 参数不支持多列上的去重操作

```
hive (hahaha)> select i_item_sk,count(distinct i_item_id),count(distinct i_brand_id)
from item group by i_item_sk limit 10;
FAILED: SemanticException [Error 10022]: DISTINCT on different columns not supported
with skew in data
```

10.3 无效 ID 在关联时的数据倾斜问题

10.3.1 问题描述

在某个关联查询中，某关联字段在表中出现相同无效值过多的情况，比如 USER_ID,DEVICE_NUMBER 为空值，并且这种值占总记录数的比例较高，容易导致这些相同值在 Reduce 阶段集中在某个或某几个 Reduce 上，导致 Reduce 运行时间非常长。

10.3.2 原 SQL 场景

```
#测试过程只是为了演示，并未使用生产数据
```

```
create table user_info(user_id int,username string);
create table customer_info(user_id int,customer_id int);

insert into table user_info values (1,'zyl'),(2,'dlw'),(3,'lyj'),(4,'st');
insert into table user_info values (NULL,NULL) ,(NULL,'ly'),(NULL,'csy'),(NULL,'pyf');
insert into table customer_info values (1,11),(2,12),(3,13);
```

```
SELECT A.user_Id,A.username,b.customer_id
FROM user_info A
LEFT JOIN customer_info b
ON A.user_id= b.user_id;
```

```
#结果
1    zyl  11
2    dlw  12
3    lyj  13
4    st   NULL
NULL NULL NULL
NULL ly   NULL
NULL csy  NULL
NULL pyf  NULL
```

10.3.3 优化方法一

思路：将 NULL 值提取出来最后合并，这一部分只有 Map 操作；非 NULL 值的数据分散到不同 Reduce 上，不会出现某个 Redcue 任务数据加工时间过长的情况，整体效率提升明显。总部出现两次 MapReuce 由于出现两次 Table Scan 导致 Map 增多，稍逊于优化方法 2。

```
SELECT A.user_Id,A.username,b.customer_id
FROM user_info A
LEFT JOIN customer_info b
 ON A.user_id = b.user_id
where A.user_id IS NOT NULL
UNION ALL
SELECT A.user_Id,A.username,NULL
FROM user_info A
WHERE A.user_id IS NULL
```

10.3.4 优化方法二

#思路：在关联阶段直接把 NULL 值打散成随机值来作为 Reduce 的 key 值，不会出现某个 Redcue 任务数据加工时间过长的情况，整体效率提升明显。解释计划只有一次 MapReduce，效率一般优于优化方法 1.

```
SELECT A.user_Id,A.username,b.customer_id
FROM user_info A
LEFT JOIN customer_info b
ON
CASE WHEN
 A.user_id IS NULL
THEN
 CONCAT ('dp_hive', RAND())
ELSE
 A.user_id
END = b.user_id;
```