

HiveSQL 执行计划详解

本文档来自公众号：五分钟学大数据

微信扫码关注



目录

一、前言.....	3
二、SQL 的执行计划.....	3
1. explain 的用法.....	3
2. explain 的使用场景.....	6
案例一：join 语句会过滤 null 的值吗？	6
案例二：group by 分组语句会进行排序吗？	7
案例三：哪条 sql 执行效率高呢？	8
案例四：定位产生数据倾斜的代码段.....	13
3. explain dependency 的用法.....	16
案例一：识别看似等价的代码.....	17
案例二：识别 SQL 读取数据范围的差别.....	19
4. explain authorization 的用法.....	20
最后.....	21

一、前言

Hive SQL 的执行计划描述 SQL 实际执行的整体轮廓，通过执行计划能了解 SQL 程序在转换成相应计算引擎的执行逻辑，掌握了执行逻辑也就能更好地把握程序出现的瓶颈点，从而能够实现更有针对性的优化。此外还能帮助开发者识别看似等价的 SQL 其实是不等价的，看似不等价的 SQL 其实是等价的 SQL。可以说执行计划是打开 SQL 优化大门的一把钥匙。

要想学 SQL 执行计划，就需要学习查看执行计划的命令：`explain`，在查询语句的 SQL 前面加上关键字 `explain` 是查看执行计划的基本方法。

学会 `explain`，能够给我们工作中使用 `hive` 带来极大的便利！

二、SQL 的执行计划

Hive 提供的执行计划目前可以查看的信息有以下几种：

- `explain`：查看执行计划的基本信息；
- `explain dependency`：`dependency` 在 `explain` 语句中使用会产生有关计划中输入的额外信息。它显示了输入的各种属性；
- `explain authorization`：查看 SQL 操作相关权限的信息；
- `explain vectorization`：查看 SQL 的向量化描述信息，显示为什么未对 Map 和 Reduce 进行矢量化。从 Hive 2.3.0 开始支持；
- `explain analyze`：用实际的行数注释计划。从 Hive 2.2.0 开始支持；
- `explain.cbo`：输出由 Calcite 优化器生成的计划。CBO 从 Hive 4.0.0 版本开始支持；
- `explain locks`：这对于了解系统将获得哪些锁以运行指定的查询很有用。LOCKS 从 Hive 3.2.0 开始支持；
- `explain ast`：输出查询的抽象语法树。AST 在 Hive 2.1.0 版本删除了，存在 bug，转储 AST 可能会导致 OOM 错误，将在 4.0.0 版本修复；
- `explain extended`：加上 `extended` 可以输出有关计划的额外信息。这通常是物理信息，例如文件名，这些额外信息对我们用处不大；

1. `explain` 的用法

Hive 提供了 `explain` 命令来展示一个查询的执行计划，这个执行计划对于我们了解底层原理，Hive 调优，排查数据倾斜等很有帮助。

使用语法如下：

```
explain query;
```

在 hive cli 中输入以下命令(hive 2.3.7):

```
explain select sum(id) from test1;
```

得到结果:

STAGE DEPENDENCIES:

```
Stage-1 is a root stage
```

```
Stage-0 depends on stages: Stage-1
```

STAGE PLANS:

```
Stage: Stage-1
```

```
Map Reduce
```

```
Map Operator Tree:
```

```
TableScan
```

```
alias: test1
```

```
Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column stat
```

s: NONE

```
Select Operator
```

```
expressions: id (type: int)
```

```
outputColumnNames: id
```

```
Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column st
```

ats: NONE

```
Group By Operator
```

```
aggregations: sum(id)
```

```
mode: hash
```

```
outputColumnNames: _col0
```

```
Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column s
```

tats: NONE

```
Reduce Output Operator
```

```
sort order:
```

```
Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column
```

stats: NONE

```
value expressions: _col0 (type: bigint)
```

```
Reduce Operator Tree:
```

```
Group By Operator
```

```
aggregations: sum(VALUE._col0)
```

```
mode: mergepartial
```

```
outputColumnNames: _col0
```

```
Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats:
```

NONE

```
File Output Operator
```

```
compressed: false
```

```
Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats:
```

NONE

```
table:
```

```

input format: org.apache.hadoop.mapred.SequenceFileInputFormat
output format: org.apache.hadoop.hive ql.io.HiveSequenceFileOutputF
ormat
serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

Stage: Stage-0
Fetch Operator
limit: -1
Processor Tree:
ListSink

```

看完以上内容有什么感受，是不是感觉都看不懂，不要着急，下面将会详细讲解每个参数，相信你学完下面的内容之后再看 explain 的查询结果将游刃有余。一个 HIVE 查询被转换为一个由一个或多个 stage 组成的序列（有向无环图 DAG）。这些 stage 可以是 MapReduce stage，也可以是负责元数据存储的 stage，也可以是负责文件系统的操作（比如移动和重命名）的 stage。

我们将上述结果拆分看，先从最外层开始，包含两个大的部分：

1. stage dependencies: 各个 stage 之间的依赖性
2. stage plan: 各个 stage 的执行计划

先看第一部分 stage dependencies，包含两个 stage，Stage-1 是根 stage，说明这是开始的 stage，Stage-0 依赖 Stage-1，Stage-1 执行完成后执行 Stage-0。

再看第二部分 stage plan，里面有一个 Map Reduce，一个 MR 的执行计划分为两个部分：

1. Map Operator Tree: MAP 端的执行计划树
2. Reduce Operator Tree: Reduce 端的执行计划树

这两个执行计划树里面包含这条 sql 语句的 operator：

1. **TableScan: 表扫描操作**，map 端第一个操作肯定是加载表，所以就是表扫描操作，常见的属性：
 - alias: 表名称
 - Statistics: 表统计信息，包含表中数据条数，数据大小等
2. **Select Operator: 选取操作**，常见的属性：
 - expressions: 需要的字段名称及字段类型
 - outputColumnNames: 输出的列名称
 - Statistics: 表统计信息，包含表中数据条数，数据大小等
3. **Group By Operator: 分组聚合操作**，常见的属性：

- aggregations: 显示聚合函数信息
 - mode: 聚合模式, 值有 hash: 随机聚合, 就是 hash partition; partial: 局部聚合; final: 最终聚合
 - keys: 分组的字段, 如果没有分组, 则没有此字段
 - outputColumnNames: 聚合之后输出列名
 - Statistics: 表统计信息, 包含分组聚合之后的数据条数, 数据大小等
4. **Reduce Output Operator: 输出到 reduce 操作**, 常见属性:
 - sort order: 值为空 不排序; 值为 + 正序排序, 值为 - 倒序排序; 值为 +- 排序的列为两列, 第一列为正序, 第二列为倒序
 5. **Filter Operator: 过滤操作**, 常见的属性:
 - predicate: 过滤条件, 如 sql 语句中的 where id>=1, 则此处显示(id >= 1)
 6. **Map Join Operator: join 操作**, 常见的属性:
 - condition map: join 方式, 如 Inner Join 0 to 1 Left Outer Join 0 to 2
 - keys: join 的条件字段
 - outputColumnNames: join 完成之后输出的字段
 - Statistics: join 完成之后生成的数据条数, 大小等
 7. **File Output Operator: 文件输出操作**, 常见的属性
 - compressed: 是否压缩
 - table: 表的信息, 包含输入输出文件格式化方式, 序列化方式等
 8. **Fetch Operator 客户端获取数据操作**, 常见的属性:
 - limit, 值为 -1 表示不限制条数, 其他值为限制的条数

2. explain 的使用场景

本节介绍 explain 能够为我们在生产实践中带来哪些便利及解决我们哪些迷惑

案例一: join 语句会过滤 null 的值吗?

现在, 我们在 hive cli 输入以下查询计划语句

```
select
  a.id,
  b.user_name
from test1 a
join test2 b
on a.id=b.id;
```

问: 上面这条 join 语句会过滤 id 为 null 的值吗

执行下面语句:

```
explain
select
  a.id,
  b.user_name
from test1 a
join test2 b
on a.id=b.id;
```

我们来看结果（为了适应页面展示，仅截取了部分输出信息）：

```
TableScan
  alias: a
  Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column stats: NONE
  Filter Operator
    predicate: id is not null (type: boolean)
    Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column stats: NONE
  Select Operator
    expressions: id (type: int)
    outputColumnNames: _col0
    Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column stats: N
ONE
  HashTable Sink Operator
    keys:
      0 _col0 (type: int)
      1 _col0 (type: int)
    ...
```

从上述结果可以看到 `predicate: id is not null` 这样一行，说明 `join` 时会自动过滤掉关联字段为 `null` 值的情况，但 `left join` 或 `full join` 是不会自动过滤 `null` 值的，大家可以自行尝试下。

案例二：group by 分组语句会进行排序吗？

看下面这条 sql

```
select
  id,
  max(user_name)
from test1
group by id;
```

问：group by 分组语句会进行排序吗

直接来看 explain 之后结果（为了适应页面展示，仅截取了部分输出信息）

```

TableScan
  alias: test1
  Statistics: Num rows: 9 Data size: 108 Basic stats: COMPLETE Column stats: NONE
  Select Operator
    expressions: id (type: int), user_name (type: string)
    outputColumnNames: id, user_name
    Statistics: Num rows: 9 Data size: 108 Basic stats: COMPLETE Column stats:
NONE
  Group By Operator
    aggregations: max(user_name)
    keys: id (type: int)
    mode: hash
    outputColumnNames: _col0, _col1
    Statistics: Num rows: 9 Data size: 108 Basic stats: COMPLETE Column stat
s: NONE
  Reduce Output Operator
    key expressions: _col0 (type: int)
    sort order: +
    Map-reduce partition columns: _col0 (type: int)
    Statistics: Num rows: 9 Data size: 108 Basic stats: COMPLETE Column st
ats: NONE
    value expressions: _col1 (type: string)
  ...

```

我们看 Group By Operator，里面有 keys: id (type: int) 说明按照 id 进行分组的，再往下看还有 sort order: +，说明是按照 id 字段进行正序排序的。

案例三：哪条 sql 执行效率高呢？

观察两条 sql 语句

```

SELECT
  a.id,
  b.user_name
FROM
  test1 a
JOIN test2 b ON a.id = b.id
WHERE
  a.id > 2;

SELECT
  a.id,
  b.user_name
FROM

```



```
(SELECT * FROM test1 WHERE id > 2) a
JOIN test2 b ON a.id = b.id;
```

这两条 sql 语句输出的结果是一样的，但是哪条 sql 执行效率高呢？

有人说第一条 sql 执行效率高，因为第二条 sql 有子查询，子查询会影响性能；有人说第二条 sql 执行效率高，因为先过滤之后，在进行 join 时的条数减少了，所以执行效率就高了。

到底哪条 sql 效率高呢，我们直接在 sql 语句前面加上 explain，看下执行计划不就知道了嘛！

在第一条 sql 语句前加上 explain，得到如下结果

```
hive (default)> explain select a.id,b.user_name from test1 a join test2 b on a.id=b.
id where a.id >2;
```

OK

Explain

STAGE DEPENDENCIES:

```
Stage-4 is a root stage
Stage-3 depends on stages: Stage-4
Stage-0 depends on stages: Stage-3
```

STAGE PLANS:

```
Stage: Stage-4
  Map Reduce Local Work
    Alias -> Map Local Tables:
      $hdt$_0:a
    Fetch Operator
      limit: -1
    Alias -> Map Local Operator Tree:
      $hdt$_0:a
      TableScan
      alias: a
      Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column statistics: NONE
    Filter Operator
      predicate: (id > 2) (type: boolean)
      Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column statistics: NONE
    Select Operator
      expressions: id (type: int)
      outputColumnNames: _col0
      Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column statistics: NONE
    HashTable Sink Operator
```

```

      keys:
        0 _col0 (type: int)
        1 _col0 (type: int)

Stage: Stage-3
  Map Reduce
    Map Operator Tree:
      TableScan
        alias: b
        Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column statistics: NONE
      Filter Operator
        predicate: (id > 2) (type: boolean)
        Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column statistics: NONE
      Select Operator
        expressions: id (type: int), user_name (type: string)
        outputColumnNames: _col0, _col1
        Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column statistics: NONE
      Map Join Operator
        condition map:
          Inner Join 0 to 1
        keys:
          0 _col0 (type: int)
          1 _col0 (type: int)
        outputColumnNames: _col0, _col2
        Statistics: Num rows: 2 Data size: 27 Basic stats: COMPLETE Column statistics: NONE
      Select Operator
        expressions: _col0 (type: int), _col2 (type: string)
        outputColumnNames: _col0, _col1
        Statistics: Num rows: 2 Data size: 27 Basic stats: COMPLETE Column statistics: NONE
      File Output Operator
        compressed: false
        Statistics: Num rows: 2 Data size: 27 Basic stats: COMPLETE Column statistics: NONE
        table:
          input format: org.apache.hadoop.mapred.SequenceFileInputFormat
          output format: org.apache.hadoop.hive ql.io.HiveSequenceFileOutputFormat
          serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

```

Local Work:

Map Reduce Local Work

Stage: Stage-0

Fetch Operator

limit: -1

Processor Tree:

ListSink

在第二条 sql 语句前加上 explain, 得到如下结果

```
hive (default)> explain select a.id,b.user_name from(select * from test1 where id>
2 ) a join test2 b on a.id=b.id;
```

OK

Explain

STAGE DEPENDENCIES:

Stage-4 is a root stage

Stage-3 depends on stages: Stage-4

Stage-0 depends on stages: Stage-3

STAGE PLANS:

Stage: Stage-4

Map Reduce Local Work

Alias -> Map Local Tables:

\$hdt\$_0:test1

Fetch Operator

limit: -1

Alias -> Map Local Operator Tree:

\$hdt\$_0:test1

TableScan

alias: test1

Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column stat

s: NONE

Filter Operator

predicate: (id > 2) (type: boolean)

Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column st

ats: NONE

Select Operator

expressions: id (type: int)

outputColumnNames: _col0

Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column

stats: NONE

HashTable Sink Operator

keys:

0 _col0 (type: int)

```

1 _col0 (type: int)

Stage: Stage-3
  Map Reduce
    Map Operator Tree:
      TableScan
        alias: b
        Statistics: Num rows: 6 Data size: 75 Basic stats: COMPLETE Column statistics: NONE
      Filter Operator
        predicate: (id > 2) (type: boolean)
        Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column statistics: NONE
      Select Operator
        expressions: id (type: int), user_name (type: string)
        outputColumnNames: _col0, _col1
        Statistics: Num rows: 2 Data size: 25 Basic stats: COMPLETE Column statistics: NONE
      Map Join Operator
        condition map:
          Inner Join 0 to 1
        keys:
          0 _col0 (type: int)
          1 _col0 (type: int)
        outputColumnNames: _col0, _col2
        Statistics: Num rows: 2 Data size: 27 Basic stats: COMPLETE Column statistics: NONE
      Select Operator
        expressions: _col0 (type: int), _col2 (type: string)
        outputColumnNames: _col0, _col1
        Statistics: Num rows: 2 Data size: 27 Basic stats: COMPLETE Column statistics: NONE
      File Output Operator
        compressed: false
        Statistics: Num rows: 2 Data size: 27 Basic stats: COMPLETE Column statistics: NONE
        table:
          input format: org.apache.hadoop.mapred.SequenceFileInputFormat
          output format: org.apache.hadoop.hive ql.io.HiveSequenceFileOutputFormat
          serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
    Local Work:
      Map Reduce Local Work

```

```
Stage: Stage-0
Fetch Operator
limit: -1
Processor Tree:
ListSink
```

大家有什么发现，除了表别名不一样，其他的执行计划完全一样，都是先进行 where 条件过滤，在进行 join 条件关联。说明 hive 底层会自动帮我们进行优化，所以这两条 sql 语句执行效率是一样的。

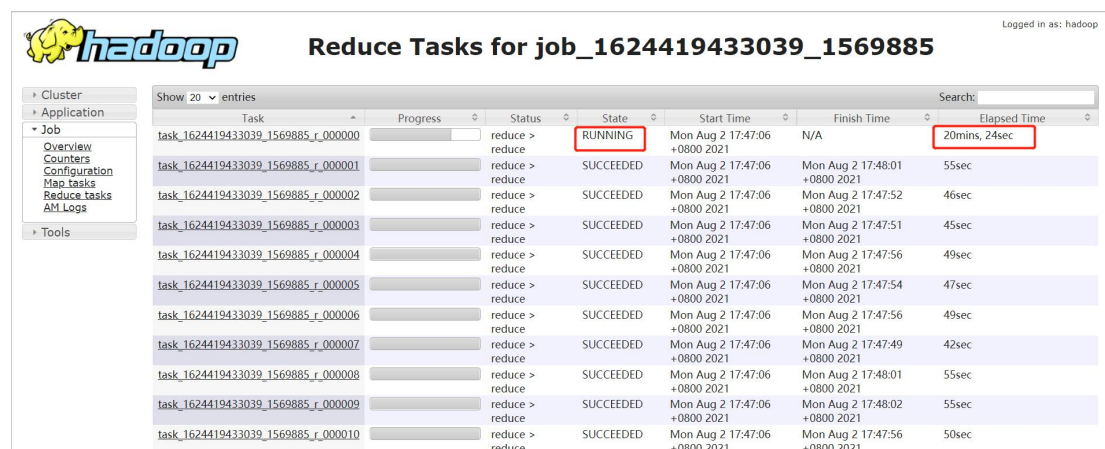
案例四：定位产生数据倾斜的代码段

数据倾斜大多数都是大 key 问题导致的。

如何判断是大 key 导致的问题，可以通过下面方法：

1. 通过时间判断

如果某个 reduce 的时间比其他 reduce 时间长的多，如下图，大部分 task 在 1 分钟之内完成，只有 r_000000 这个 task 执行 20 多分钟了还没完成。



Task	Progress	Status	State	Start Time	Finish Time	Elapsed Time
task_1624419433039_1569885_r_000000		reduce >	RUNNING	Mon Aug 2 17:47:06 +0800 2021	N/A	20mins, 24sec
task_1624419433039_1569885_r_000001		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:48:01 +0800 2021	55sec
task_1624419433039_1569885_r_000002		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:47:52 +0800 2021	46sec
task_1624419433039_1569885_r_000003		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:47:51 +0800 2021	45sec
task_1624419433039_1569885_r_000004		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:47:56 +0800 2021	49sec
task_1624419433039_1569885_r_000005		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:47:54 +0800 2021	47sec
task_1624419433039_1569885_r_000006		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:47:56 +0800 2021	49sec
task_1624419433039_1569885_r_000007		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:47:49 +0800 2021	42sec
task_1624419433039_1569885_r_000008		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:48:01 +0800 2021	55sec
task_1624419433039_1569885_r_000009		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:48:02 +0800 2021	55sec
task_1624419433039_1569885_r_000010		reduce >	SUCCEEDED	Mon Aug 2 17:47:06 +0800 2021	Mon Aug 2 17:47:56 +0800 2021	50sec

注意：要排除两种情况：

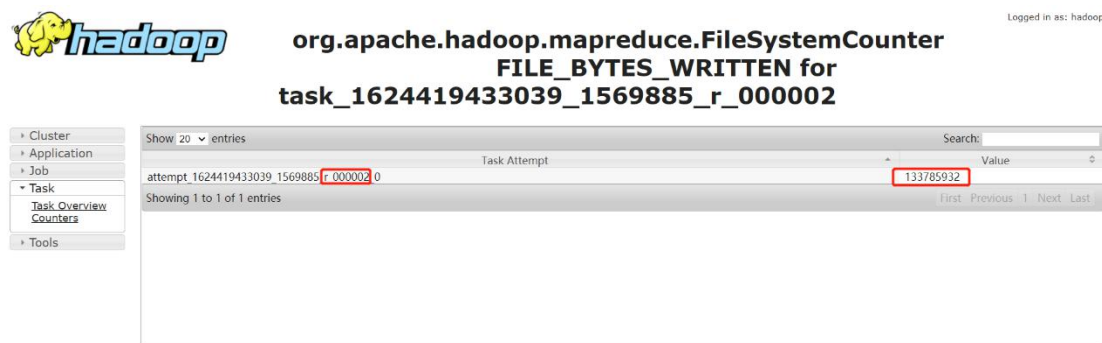
1. 如果每个 reduce 执行时间差不多，都特别长，不一定是数据倾斜导致的，可能是 reduce 设置过少导致的。
2. 有时候，某个 task 执行的节点可能有问题，导致任务跑的特别慢。这个时候，mapreduce 的推测执行，会重启一个任务。如果新的任务在很短时间能完成，通常则是由于 task 执行节点问题导致的个别 task 慢。但是如果推测执行后的 task 执行任务也特别慢，那更说明该 task 可能会有倾斜问题。

2. 通过任务 Counter 判断

Counter 会记录整个 job 以及每个 task 的统计信息。counter 的 url 一般类似：

http://bd001:8088/proxy/application_1624419433039_1569885/mapreduce/singletaskcounter/task_1624419433039_1569885_r_000000/org.apache.hadoop.mapreduce.FileSystemCounter

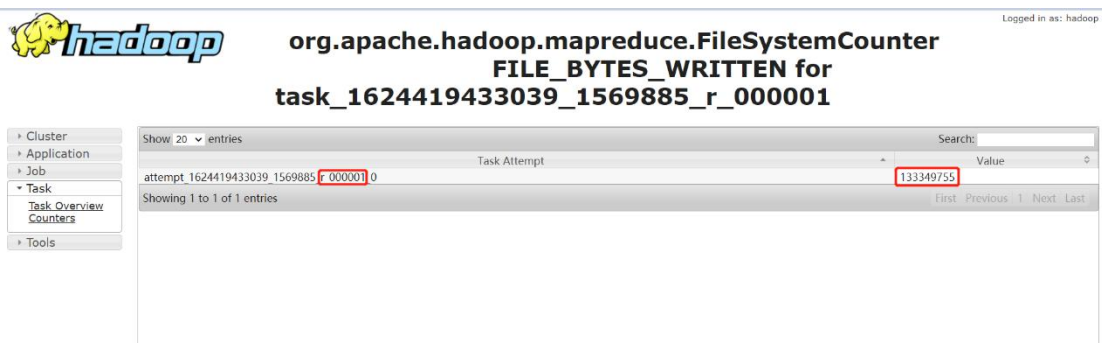
通过输入记录数，普通的 task counter 如下，输入的记录数是 13 亿多：



org.apache.hadoop.mapreduce.FileSystemCounter
FILE_BYTES_WRITTEN for
task_1624419433039_1569885_r_000002

Task Attempt	Value
attempt_1624419433039_1569885_r_000002_0	133785932

Showing 1 to 1 of 1 entries

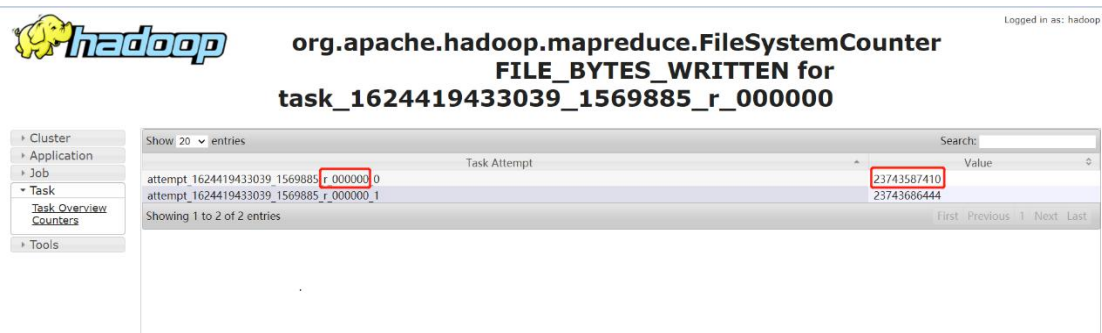


org.apache.hadoop.mapreduce.FileSystemCounter
FILE_BYTES_WRITTEN for
task_1624419433039_1569885_r_000001

Task Attempt	Value
attempt_1624419433039_1569885_r_000001_0	133349755

Showing 1 to 1 of 1 entries

而 task=000000 的 counter 如下，其输入记录数是 230 多亿。是其他任务的 100 多倍：



org.apache.hadoop.mapreduce.FileSystemCounter
FILE_BYTES_WRITTEN for
task_1624419433039_1569885_r_000000

Task Attempt	Value
attempt_1624419433039_1569885_r_000000_0	23743587410
attempt_1624419433039_1569885_r_000000_1	23743686444

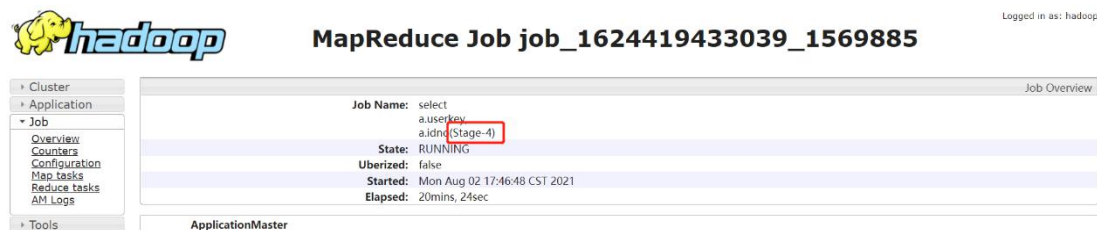
Showing 1 to 2 of 2 entries

定位 SQL 代码

1. 确定任务卡住的 stage

- 通过 jobname 确定 stage:

一般 Hive 默认的 jobname 名称会带上 stage 阶段，如下通过 jobname 看到任务卡住的为 Stage-4:

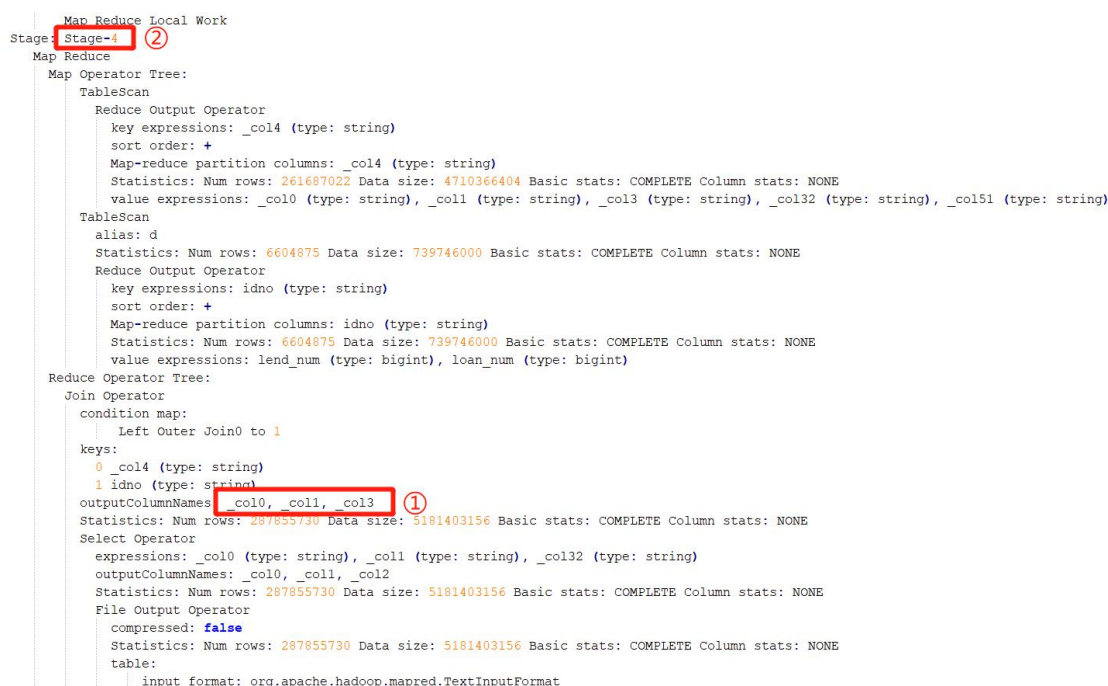


- 如果 jobname 是自定义的，那可能没法通过 jobname 判断 stage。需要借助于任务日志：

找到执行特别慢的那个 task，然后 Ctrl+F 搜索 “CommonJoinOperator: JOIN struct<_col0:string, _col1:string, _col3:string> totalsz = 3”。Hive 在 join 的时候，会把 join 的 key 打印到日志中。如下：

```
2021-08-02 18:05:40,259 INFO [Thread-7] org.apache.hadoop.hive.q1.exec.JoinOperator: Initializing Self JOIN[0]
2021-08-02 18:06:40,837 INFO [Thread-7] org.apache.hadoop.hive.q1.exec.CommonJoinOperator: JOIN struct<_col0:string, _col1:string, _col3:string> totalsz = 3
2021-08-02 18:07:41,415 INFO [Thread-7] org.apache.hadoop.hive.q1.exec.JoinOperator: Operator 0 JOIN initialized
2021-08-02 18:08:42,041 INFO [Thread-7] org.apache.hadoop.hive.q1.exec.JoinOperator: Initializing children of 0 JOIN
```

上图中的关键信息是：struct<_col0:string, _col1:string, _col3:string> 这时候，需要参考该 SQL 的执行计划。通过参考执行计划，可以断定该阶段为 Stage-4 阶段：



2. 确定 SQL 执行代码

确定了执行阶段，即 Stage-4 阶段。通过执行计划，则可以判断出是执行哪段代码时出现了倾斜。还是从此图，这个 Stage-4 阶段中进行连接操作的表别名是 d：


```

Map Reduce Local Work
Stage: Stage-4
Map Reduce
Map Operator Tree:
  TableScan
    Reduce Output Operator
      key expressions: _col4 (type: string)
      sort order: +
      Map-reduce partition columns: _col4 (type: string)
      Statistics: Num rows: 261687022 Data size: 4710366404 Basic stats: COMPLETE Column stats: NONE
      value expressions: _col0 (type: string), _col1 (type: string), _col3 (type: string), _col32 (type: string), _col51 (type: string)
  TableScan
    alias: d
    Statistics: Num rows: 6604875 Data size: 739746000 Basic stats: COMPLETE Column stats: NONE
    Reduce Output Operator
      key expressions: idno (type: string)
      sort order: +
      Map-reduce partition columns: idno (type: string)
      Statistics: Num rows: 6604875 Data size: 739746000 Basic stats: COMPLETE Column stats: NONE
      value expressions: lend_num (type: bigint), loan_num (type: bigint)
Reduce Operator Tree:
  Join Operator
    condition map:
      Left Outer Join0 to 1
    keys:
      0 _col4 (type: string)
      1 idno (type: string)
    outputColumnNames: _col0, _col1, _col3
    Statistics: Num rows: 287855730 Data size: 5181403156 Basic stats: COMPLETE Column stats: NONE
  Select Operator
    expressions: _col0 (type: string), _col1 (type: string), _col32 (type: string)
    outputColumnNames: _col0, _col1, _col2
    Statistics: Num rows: 287855730 Data size: 5181403156 Basic stats: COMPLETE Column stats: NONE
  File Output Operator
    compressed: false
    Statistics: Num rows: 287855730 Data size: 5181403156 Basic stats: COMPLETE Column stats: NONE
    table:
      input format: org.apache.hadoop.mapred.TextInputFormat
  
```

就可以推测出是在执行下面红框中代码时出现了数据倾斜，因为这行的表的别名是 d：

```

select
  a.userkey,
  a.idno,
  a.phone,
  a.name,
  b.user_active_at,
  c.intend_commodity,
  c.intend_rank,
  d.order_num,
  d.order_amount
from user_info a
left join user_active b on a.userkey = b.userkey
left join user_intend c on a.phone = c.phone
left join user_order d on a.idno = d.idno;
  
```

以上仅列举了 4 个我们生产中既熟悉又有点迷糊的例子，explain 还有很多其他的用途，如查看 stage 的依赖情况、hive 调优等，小伙伴们可以自行尝试。

3. explain dependency 的用法

explain dependency 用于描述一段 SQL 需要的数据来源，输出是一个 json 格式的数据，里面包含以下两个部分的内容：

- **input_partitions**: 描述一段 SQL 依赖的数据来源表分区，里面存储的是分区名的列表，如果整段 SQL 包含的所有表都是非分区表，则显示为空。
- **input_tables**: 描述一段 SQL 依赖的数据来源表，里面存储的是 Hive 表名的列表。

使用 **explain dependency** 查看 SQL 查询非分区普通表，在 hive cli 中输入以下命令：

```
explain dependency select s_age,count(1) num from student_orc;
```

得到结果：

```
{"input_partitions":[],"input_tables":[{"tablename":"default@student_tb _orc","tabletype":"MANAGED_TABLE"}]}
```

使用 **explain dependency** 查看 SQL 查询分区表，在 hive cli 中输入以下命令：

```
explain dependency select s_age,count(1) num from student_orc_partition;
```

得到结果：

```
{"input_partitions":[{"partitionName":"default@student_orc_partition@ part=0"}, {"partitionName":"default@student_orc_partition@part=1"}, {"partitionName":"default@student_orc_partition@part=2"}, {"partitionName":"default@student_orc_partition@part=3"}, {"partitionName":"default@student_orc_partition@part=4"}, {"partitionName":"default@student_orc_partition@part=5"}, {"partitionName":"default@student_orc_partition@part=6"}, {"partitionName":"default@student_orc_partition@part=7"}, {"partitionName":"default@student_orc_partition@part=8"}, {"partitionName":"default@student_orc_partition@part=9"}], "input_tables":[{"tablename":"default@student_orc_partition", "tabletype":"MANAGED_TABLE"}]}
```

explain dependency 的使用场景有两个：

- **场景一**：快速排除。快速排除因为读取不到相应分区的数据而导致任务数据输出异常。例如，在一个以天分区的任务中，上游任务因为生产过程不可控因素出现异常或者空跑，导致下游任务引发异常。通过这种方式，可以快速查看 SQL 读取的分区是否出现异常。
- **场景二**：理清表的输入，帮助理解程序的运行，特别是有助于理解有多重子查询，多表连接的依赖输入。

下面通过两个案例来看 explain dependency 的实际运用：

案例一：识别看似等价的代码

对于刚接触 SQL 的程序员，很容易将

```
select * from a inner join b on a.no=b.no and a.f>1 and a.f<3;
```

等价于

```
select * from a inner join b on a.no=b.no where a.f>1 and a.f<3;
```

我们可以通过案例来查看下它们的区别：

代码 1:

```
select
a.s_no
from student_orc_partition a
inner join
student_orc_partition_only b
on a.s_no=b.s_no and a.part=b.part and a.part>=1 and a.part<=2;
```

代码 2:

```
select
a.s_no
from student_orc_partition a
inner join
student_orc_partition_only b
on a.s_no=b.s_no and a.part=b.part
where a.part>=1 and a.part<=2;
```

我们看下上述两段代码 explain dependency 的输出结果：

代码 1 的 explain dependency 结果:

```
{"input_partitions":
[{"partitionName":"default@student_orc_partition@part=1"},
{"partitionName":"default@student_orc_partition@part=2"},
{"partitionName":"default@student_orc_partition_only@part=0"},
{"partitionName":"default@student_orc_partition_only@part=1"},
{"partitionName":"default@student_orc_partition_only@part=2"}],
"input_tables": [{"tablename":"default@student_orc_partition","tabletype":"MANAGED_
TABLE"}, {"tablename":"default@student_orc_partition_only","tabletype":"MANAGED_TAB
LE"}]}
```

代码 2 的 explain dependency 结果:

```
{"input_partitions":
[{"partitionName":"default@student_orc_partition@part=1"},
{"partitionName" : "default@student_orc_partition@part=2"},
{"partitionName" : "default@student_orc_partition_only@part=1"},
{"partitionName":"default@student_orc_partition_only@part=2"}],
"input_tables": [{"tablename":"default@student_orc_partition","tabletype":"MANAGED_
```

```
TABLE"}, {"tablename":"default@student_orc_partition_only","tabletype":"MANAGED_TABLE"}]]
```

通过上面的输出结果可以看到，其实上述的两个 SQL 并不等价，代码 1 在内连接（inner join）中的连接条件（on）中加入非等值的过滤条件后，并没有将内连接的右表按照过滤条件进行过滤，内连接在执行时会多读取 part=0 的分区数据。而在代码 2 中，会过滤掉不符合条件的分区。

案例二：识别 SQL 读取数据范围的差别

代码 1:

```
explain dependency
select
a.s_no
from student_orc_partition a
left join
student_orc_partition_only b
on a.s_no=b.s_no and a.part=b.part and b.part>=1 and b.part<=2;
```

代码 2:

```
explain dependency
select
a.s_no
from student_orc_partition a
left join
student_orc_partition_only b
on a.s_no=b.s_no and a.part=b.part and a.part>=1 and a.part<=2;
```

以上两个代码的数据读取范围是一样的吗？答案是不一样，我们通过 explain dependency 来看下：

代码 1 的 explain dependency 结果：

```
{"input_partitions":
[{"partitionName": "default@student_orc_partition@part=0"},
{"partitionName":"default@student_orc_partition@part=1"}, ...中间省略 7 个分区
{"partitionName":"default@student_orc_partition@part=9"},
{"partitionName":"default@student_orc_partition_only@part=1"},
{"partitionName":"default@student_orc_partition_only@part=2"}],
"input_tables": [{"tablename":"default@student_orc_partition","tabletype":"MANAGED_TABLE"}, {"tablename":"default@student_orc_partition_only","tabletype":"MANAGED_TABLE"}]]
```

代码 2 的 explain dependency 结果：

```
{
  "input_partitions": [
    {
      "partitionName": "default@student_orc_partition@part=0",
      "partitionName": "default@student_orc_partition@part=1",
      ...中间省略 7 个分区
      "partitionName": "default@student_orc_partition@part=9",
      "partitionName": "default@student_orc_partition_only@part=0",
      "partitionName": "default@student_orc_partition_only@part=1",
      ...中间省略 7 个分区
      "partitionName": "default@student_orc_partition_only@part=9"
    ]
  },
  "input_tables": [
    {
      "tablename": "default@student_orc_partition",
      "tabletype": "MANAGED_TABLE"
    },
    {
      "tablename": "default@student_orc_partition_only",
      "tabletype": "MANAGED_TABLE"
    }
  ]
}
```

可以看到，对左外连接在连接条件中加入非等值过滤的条件，如果过滤条件是作用于右表（b 表）有起到过滤的效果，则右表只要扫描两个分区即可，但是左表（a 表）会进行全表扫描。如果过滤条件是针对于左表，则完全没有起到过滤的作用，那么两个表将进行全表扫描。这时的情况就如同全外连接一样都需要对两个数据进行全表扫描。

在使用过程中，容易认为代码片段 2 可以像代码片段 1 一样进行数据过滤，通过查看 explain dependency 的输出结果，可以知道不是如此。

4. explain authorization 的用法

通过 explain authorization 可以知道当前 SQL 访问的数据来源（INPUTS）和数据输出（OUTPUTS），以及当前 Hive 的访问用户（CURRENT_USER）和操作（OPERATION）。

在 hive cli 中输入以下命令：

```
explain authorization
select variance(s_score) from student_tb_orc;
```

结果如下：

```
INPUTS:
  default@student_tb_orc
OUTPUTS:
  hdfs://node01:8020/tmp/hive/hdfs/cbf182a5-8258-4157-9194-90f1475a3ed5/-mr-10000
CURRENT_USER:
  hdfs
OPERATION:
  QUERY
AUTHORIZATION_FAILURES:
  No privilege 'Select' found for inputs { database:default, table:student_tb_orc,
  columnName:s_score}
```

从上面的信息可知：

上面案例的数据来源是 defalut 数据库中的 student_tb_orc 表；
数据的输出路径是

```
hdfs://node01:8020/tmp/hive/hdfs/cbf182a5-8258-4157-9194-90f1475a3ed5/-  
mr-10000;
```

当前的操作用户是 hdfs，操作是查询；

观察上面的信息我们还会看到 AUTHORIZATION_FAILURES 信息，提示对当前的输入没有查询权限，但如果运行上面的 SQL 的话也能够正常运行。为什么会出现这种情况？Hive 在默认不配置权限管理的情况下不进行权限验证，所有的用户在 Hive 里面都是超级管理员，即使不对特定的用户进行赋权，也能够正常查询。

最后

通过上面对 explain 的介绍，可以发现 explain 中有很多值得我们去研究的内容，读懂 explain 的执行计划有利于我们优化 Hive SQL，同时也能提升我们对 SQL 的掌控力。

搜索公众号：五分钟学大数据，学更多大数据技术！

其他大数据技术文档可下方扫码关注获取：



微信搜一搜



五分钟学大数据