1 关于慢的sql执行背景。

java调用impala-shell在本地（Impalad节点）执行任务；

连接Impalad有负载均衡机制，工具是自己开发开发的hadoop-tool，机制是随机分发到可用的Impalad节点。

任务执行Driver有五个线程，分别提交SQL任务，这些任务本身逻辑依赖。

本机资源大量占用 impalad CPU全满 4800% 内存 high。

Java提交ImpalaSQL任务，只监控任务提交成功，不监控任务执行结束；任务提交时一次全部提交？任务是分模块的，模块之间有依赖，但是模块可得任务没有依赖；依赖的处理逻辑是否合理有待确认。

2 过程可以复现 java

2 关于慢的具体描述。

大集群执行的比小集群慢。

3 是否有类似的测试环境与数据可以做测试。

4 数据类型。

5 中间表都落地。

6 4万条sql的执行过程，是串行还是并行？

7 查询分析：

4万条SQL，查询时间约为17小时，平均2s一条。

8 执行复杂的sql之后，会产生很多的thrift-server-connection，后端连接数上到10k之后，在同一节点提交节点的SQL也会执行的很慢，关于这个问题Impala原厂提供了补丁。

优化过程：

1 硬件层面，检查disk的IO吞吐

Disk IO对比，发现吞吐差异。

小集群是大集群的两倍。

2 检查应用的执行架构

通过impala-shell –f 提交sql任务，华为内部做的随机负载均衡机制。

建议改为通过HiveJdbc的方式提交SQL，同时根据Haproxy做负载均衡。

进入Haproxy+Keepalived 双活均衡负载机制。

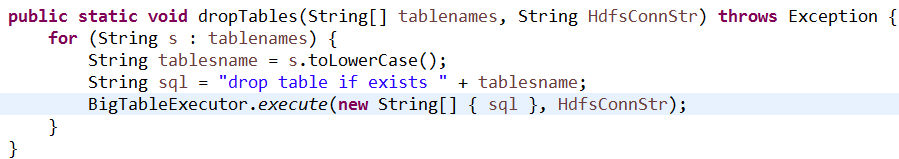
3 测试环境参数调整：

query\_log\_size 25 –-> 1024

mem\_limit 调整内存上限：100G --> 128G

4 代码优化

1 初始化时批量删除表，该多次提交为一次提交批量



2 分区字段：

表为speedtest\_mbb\_raw.t\_speedtest；

client\_country\_code ；

networktype；

月份

3 load\_catalog\_in\_background=true 启动的时候加载全部元数据，避免每次查询的时候加载

可以设置idle\_session\_timeout和idle\_query\_timeout两个参数，前面参数控制空闲session的超时时间，防止有query在waiting to be closed中停留过久，后面一个控制空闲query的超时时间。

1. 其他参数调整

1）在高并发情况下的connection storm：

1. 如果是hs2 frontend connection很高，请确保通过JDBC连接的Java代码及时关闭statement；同时可以通过设置fe\_service\_threads等参数控制前端连接数。
2. 如果是backend connection很高，可以通过Impala admission control控制资源池队列最大query数及设置mem\_limit等方式控制并发执行的query数量，这是一个常用的做法，和YARN资源池类似。

2）修改启动参数：accepted\_cnxn\_queue\_depth，可以通过增加Impala启动选项accepted\_cnxn\_queue\_depth以允许更多的backend connection缓存到队列里（队列的默认大小是10000，Cloudera建议增加到100000，通过调整这个参数，TPC-DS query 14可以运行起来）

3）因为在大集群环境下，执行非常复杂query（如：TPC-DS query 14），会因为大量runtime filter的传递造成网络拥塞并给coordinator极大的压力，所以建议在执行这类query时指定以下query option：

set MAX\_NUM\_RUNTIME\_FILTERS=50;

set RUNTIME\_FILTER\_WAIT\_TIME\_MS=10000;

set RUNTIME\_FILTER\_MIN\_SIZE=4096;

set RUNTIME\_FILTER\_MAX\_SIZE=33554432;

set RUNTIME\_BLOOM\_FILTER\_SIZE=2097152;

*关于Runtime\_fileter，由于网络传输缓慢。。。若runtime filters数据从connection数高的节点中发出，可能拖慢整个query运行速度。故和运行复杂query时相反，可以通过关闭runtime filter以节省时间。关闭命令为：DISABLE\_ROW\_RUNTIME\_FILTERING=false;*

另外出现某些节点connection数量高的情况时，可增加设置命令：set DISABLE\_CODEGEN=true;

hadoop013短时间内由于无法及时响应客户端连接, 导致远程 Daemon 的client cache用尽,因此需要创建新的连接, 这反过来又加大了 hadoop013的负载. 我建议您试一下给 Impala 配置以下参数, 延长 Fragment 状态汇报的时间间隔, 对于大负载的集群, 应该可以一定程度上降低连接数: Impala 默认以5秒(各个 fragment 取随机值,避免同时发送)间隔向 Coordinator 汇报状态. 我建议您可以试一下设置为30秒:

Cloudera Manager > Impala > Configurations > Advanced > Impala command line argument safety valve: --status\_report\_interval=30

以上修改不会影响程序正确性, 但是会影响实时查看 profile, 不过查询结束后 profile 会自动更新到正确状态.

关于本 case 的问题现象:

1) 执行一次复杂查询后, 再次往同一个Coordinator 上提交某个其他查询, 执行性能较之前有明显下降

2) 同样的查询提交到其他 Coordinator 上没有明显问题

3) 出现此问题的 Coordinator 可以观察到较多的连接, 并且从pstack日志中可以看到类似数量的线程都等待在recv()上

4) 重新启动此 Coordinator, 然后再次提交查询, 可以观察到连接数会迅速上升

我这边通过分析日志以及结合 Impala 的运行机制, 目前的结论如下:

1) Impala 执行完查询后, 已建立的连接并不会释放, 以便下次执行查询时可以节省网络连接的时间. 因此连接数上升的现象是正常的.

2) 查询变慢的 profile 显示, 主要的区别在于 Coordinator 上执行的Fragment Instance的CodeGenTime变长了:

慢查询:

- CodegenTime: 3m27s

正常查询 :

- CodegenTime: 15s696ms

CodegenTime记录的是Impala Daemon 在执行Fragment Instance 前所用去的即时编译的时间, 也就是说, 是个纯计算的过程, 如果变慢, 意味着 Impala 进程由于某些原因无法获得足够的 CPU 时间片. 由于主机的内存情况良好, 并且没有明显的 IO 问题, 因此目前推断可能与创建了大量的线程, 导致争抢时间片有一定关系. 这个问题可以在后续重现后继续做深入调查.

针对以上问题原因, 我们有以下解决方案:

1) 避免让Coordinator 执行 Fragment Instance, 这样可以获得更稳定的性能, query 执行性能不会因为 Coordinator 的负载较高而下降, 这是解决此问题的最佳方案. Impala 在新版本(5.12以上)中提供了角色分离的机制, 也就是说ImpalaD 可以配置为仅运行Coordinator或仅运行backend, 或者两者都允许. 对于较大的集群, 由于集群规模所带来的开销会增大, 因此我们建议将部分 ImpalaD 配置为仅允许运行Coordinator [1].

2) 如果升级到5.12尚未列入计划, 那么避免目前的查询性能下降的办法是尽量减少 Coordinator 的连接数以及由此导致的运行线程数量. 当执行复杂查询时, Coordinator 在短时间内由于无法及时响应客户端连接, 导致远程 Daemon 的client cache用尽,因此需要创建新的连接, 这反过来又加大了Coordinator的负载. 因此我们可以延长 Fragment 状态汇报的时间间隔, 对于大负载的集群, 可以一定程度上降低连接数, 也就是预防措施:

Impala 默认以5秒(各个 fragment 取随机值,避免同时发送)间隔向 Coordinator 汇报状态. 我建议您可以试一下设置为30秒:

Cloudera Manager > Impala > Configurations > Advanced > Impala command line argument safety valve:

--status\_report\_interval=30

3) 关闭 Codegen (set disable\_codegen=true) 则是在已经出现此问题后的权宜之计. 您的测试证明, 关闭codegen后查询性能提高明显, 但是还是比正常执行情况要慢.