Parquet与ORC对比测试报告

一、环境说明

Hadoop集群：使用测试Hadoop集群，节点：

```
hadoop230.photo.163.org
hadoop231.photo.163.org
hadoop232.photo.163.org
hadoop233.photo.163.org
```

这几台机器配置一样，具体参数可参考：http://cmdb.netease.com/cmdb/server/maintenance?searchword=hadoop231 CPU数量：2个 CPU线程数：32个 内存：128GB 磁盘：48TB

使用测试机群上的ndir队列，使用整个集群的资源

Hive使用官方的hive 1.2.1版本，使用hiveserver2的方式启动，使用本机的mysql存储元数据。

二、测试数据生成

测试数据为TPC-DS基准测试的数据，官方文档：http://www.tpc.org/information/current_specifications.asp，这个数据集一共24个表：7个事实表，17个维度表，每一个事实表和大部分的维度表组成雪花模型，scale_factor设置为100，也就是生成100GB的数据。

2.1 下载hive-testbench

git clone https://github.com/hortonworks/hive-testbench

这个项目是用于生成TPC-DS数据集并且将其导入到hive，在使用之前需要保证已经将hive、hadoop等命令加入到PATH中。

2.2 编译

进入该目录，执行./tpcds-build.sh，该命令会从TPC-DS下载源代码，并编译，初始化metaStore，为导入数据到hive做准备。

2.3 导入数据

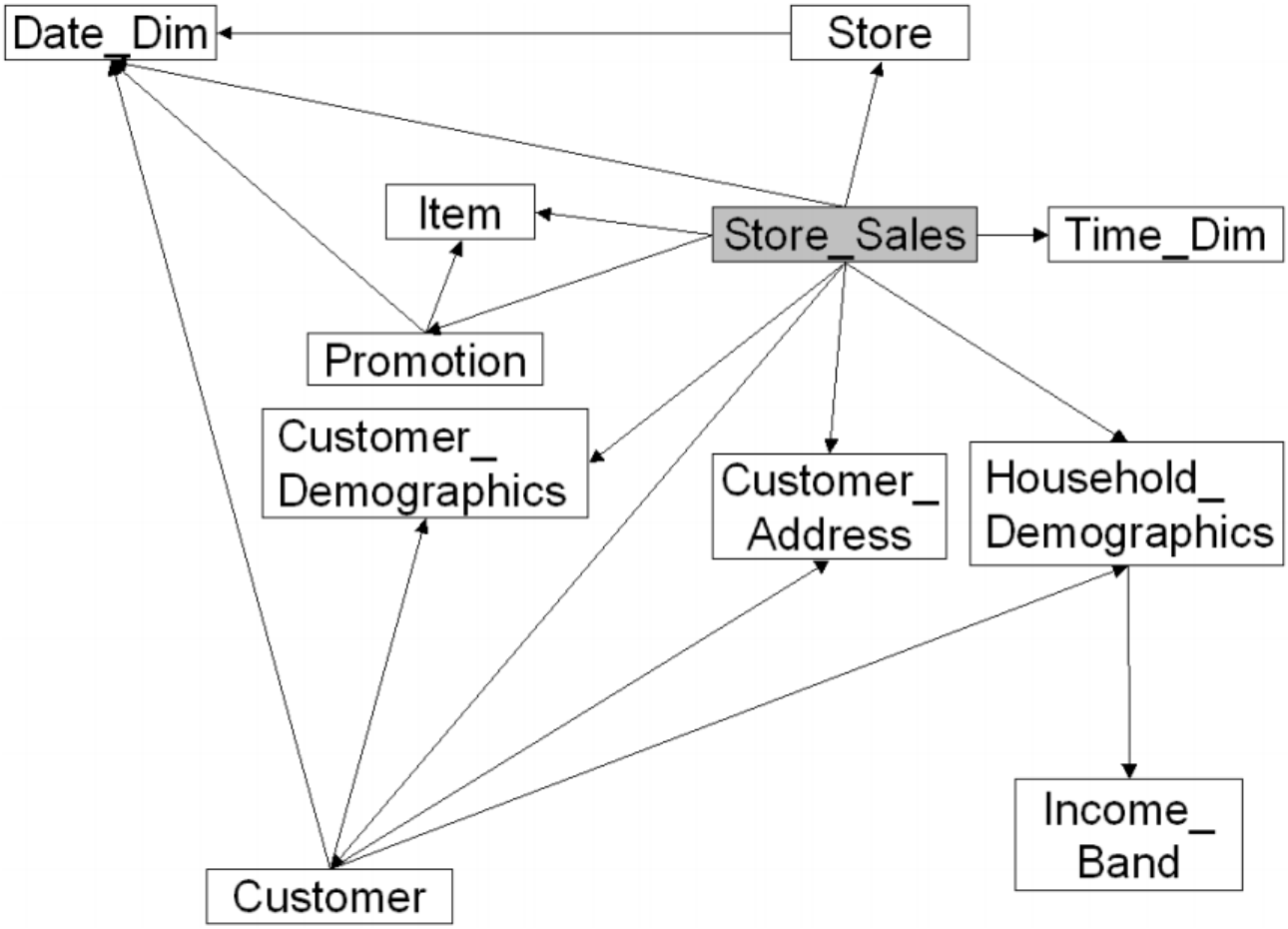
导入ORC数据：./tpcds-setup.sh 100 默认的文件格式就是ORC，所以不需要指定存储格式。 导入Parquet数据：FORMAT=parquet ./tpcds-setup.sh 100 指定文件格式为Parquet 参数100表示生成100GB的数据，改程序首先会生成text格式的数据到临时目录，然后再将这些数据转换成orc或者parquet格式。

2.4 执行测试

该项目在sample-queries-tpcds/testbench.settings文件中给出了一些hive的配置，在启动hive之前设置这些参数，在sample-queries-tpcds目录下给出了65个测试SQL，这部分是可以在hive上跑的TPC-DS查询，通过参考之前的测试，选择了其中的50个查询用于本次测试。

本次测试主要对比orc文件格式和parquet文件格式之间的查询性能，并和未压缩的text格式进行对比。hive通过hiveserver2后台运行，客户端通过jdbc的方式分别执行每一条查询，对比查询使用的Wall Time（每次查询结束的时间戳减去查询之前的时间戳）。

2.5 测试数据模型



测试使用的数据是TPC-DS数据集中以store_sales为事实表的一个雪花状模型，不同的场景基于这个数据集构造不同的表结构。

三、测试程序

- 输入参数为待测试的数据库列表、待测的SQL和循环轮次N
- 重复执行N轮查看，每一轮开始之前向hiveserver2为每一个数据库创建一个connection，然后顺序的执行每一个待测的SQL，* 执行顺序为对dbA执行query1、对dbB执行query1，对dbC执行query1，对dbA执行query2...由于每一个数据存储在不同的目录下，可以避免缓存等其他因素的影响。
- 每一轮结束之后重新销毁之前的connection，重新重复2.
- 时间统计：统计执行每一个sql消耗的时间。
- 对比的存储格式：TEXT格式、ORC格式和Parquet格式

四、场景设置

场景一：多个事实表、多个维度表，复杂的join查询

不需要修改任何数据，直接在TPC-DS数据集上执行查询。text表不是分区表（100个reducer生成），另外两种存储格式的事实表是根据data_id进行分区的分区表，执行了两次测试：1、执行全部的50个查询，2、选取符合测试数据模型的查询，分别为query27,query7,query28,query67,query82,query42,query43,query46,query73,query96，执行N此查询，计算平均值。

场景二：维度表和事实表join之后生成的宽表，只在一个表上做查询。

选取数据模型中的store*sales, household*demographics, customer*address, datedim, store表生成一个扁平式宽表(storesales*wide_table)，基于这个表执行查询，由于场景一种选择的query大多数不能完全match到这个宽表，所以使用了一些新的查询sql，表的模型、load数据的sql和查询可以参考附件。

场景三：复杂的数据结构组成的宽表，struct、list、map等（1层）

在场景二的基础上，将维度表（除了store*sales表）转换成一个struct或者map对象，源*store*sales表中的字段保持不变。生成有一层嵌套的新表（store*sales*wide*table*one_nested），使用的查询逻辑和场景二中相同，修改sql以满足新的表结构。表的模型、load数据的sql和查询可以参考附件。

场景四：复杂的数据结构，多层嵌套。（3层）

在场景三的基础上，将部分维度表的struct内的字段再转换成struct或者map对象，只存在struct中嵌套map的情况，最深的嵌套为三层。生成一个多层嵌套的新表（store*sales*wide_table*more*nested），使用的查询逻辑和场景三中相同，修改sql以满足新的表结构。表的模型、load数据的sql和查询可以参考附件。

五、测试结果

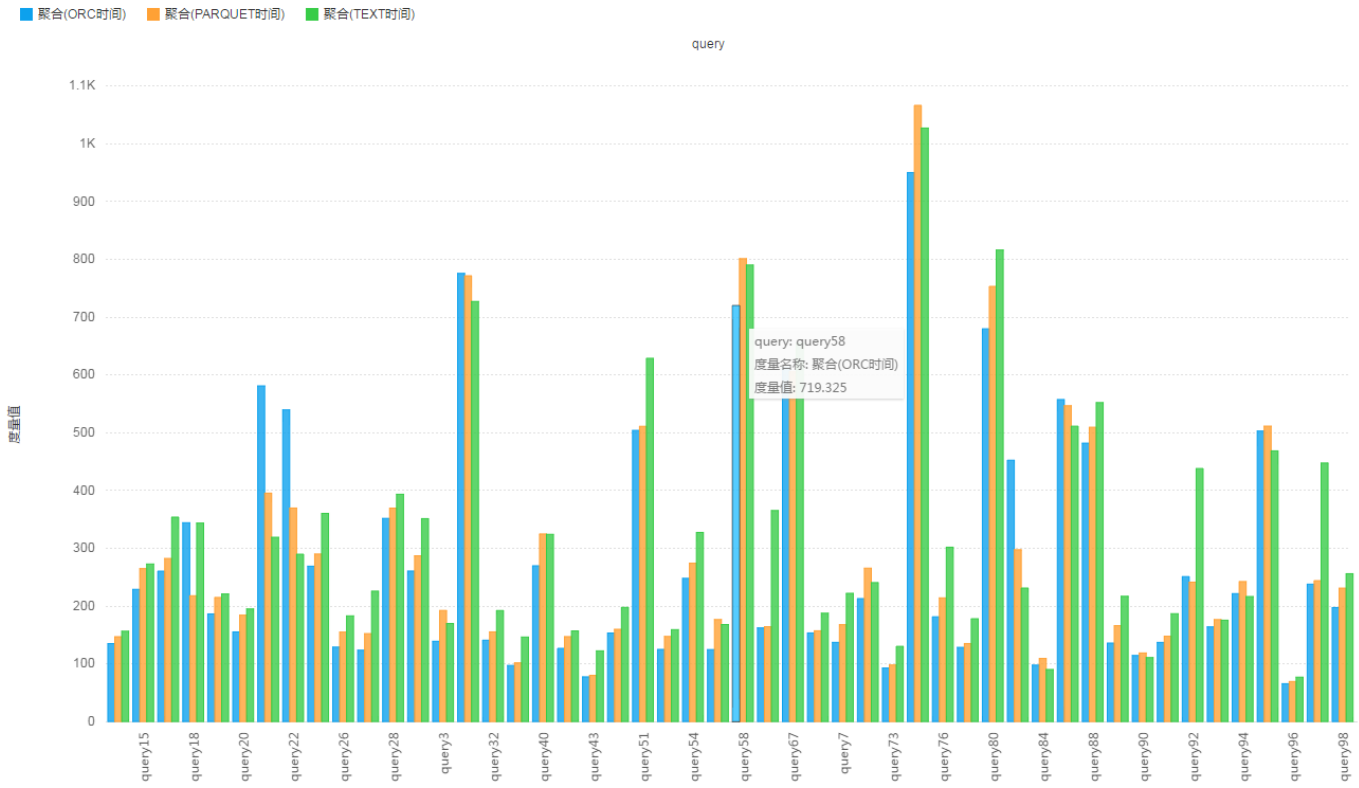
每一种场景下对比测试主要关注两个方面：1、测试数据的大小，以对比不同的文件格式在存储上的优劣，2、相同场景中的不同文件格式查询时间对比。另外，在场景二、场景三、场景四中向新表中导数据的速度为ORC > parquet > text

场景一

该场景中设计到TPC-DS中全部的表，以store_sales表为例，该表的记录数：287,997,024，文件大小为：

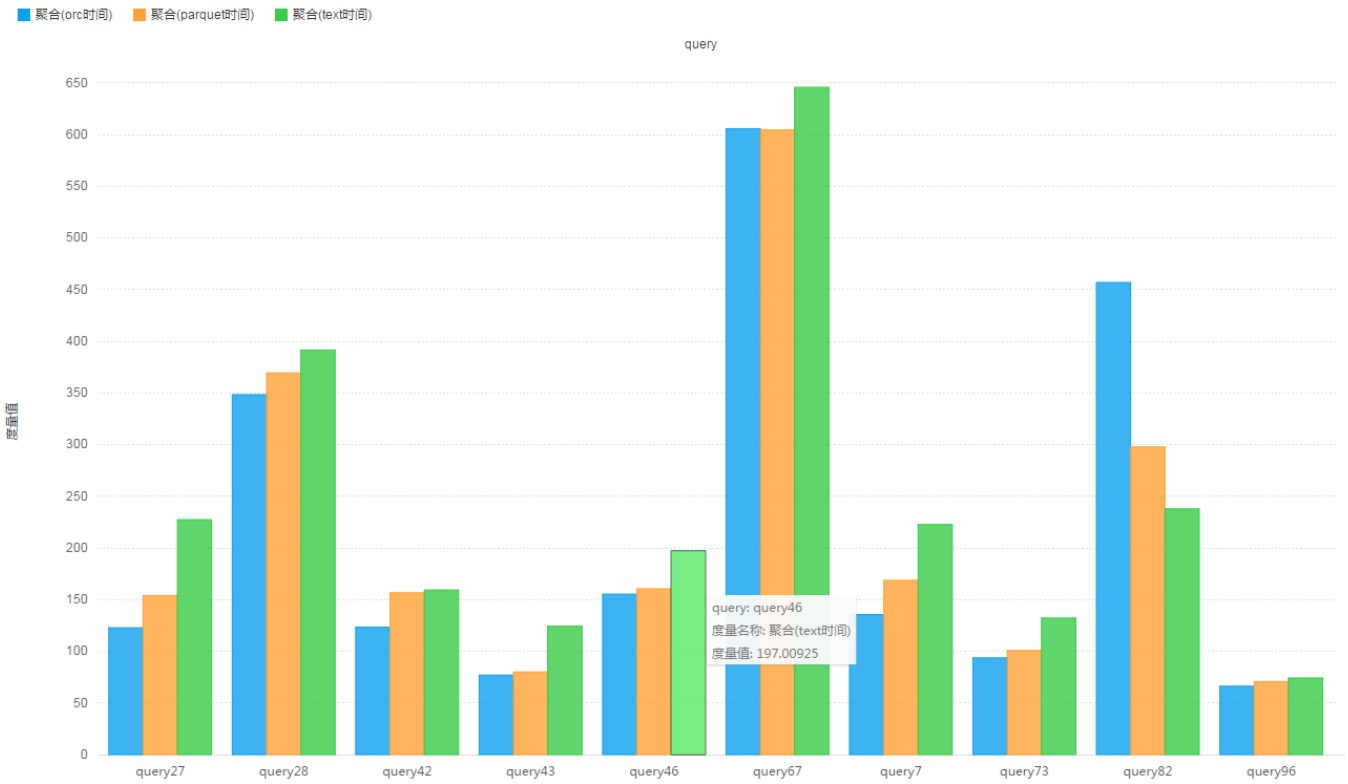
- 原始Text格式，未压缩 : 38.1 G
- ORC格式，默认压缩（ZLIB）,一共1800+个分区 : 11.5 G
- PARQUET格式，默认压缩（Snappy?），一共1800+个分区 ： 14.8 G

测试1(执行全部50个查询)结果：



说明：原始Text格式相差并不大，甚至有的更优？！可能原因是在多个表join的情况下，每一个查询需要多达10+个MR任务处理，而这些查询文件存储格式之间的查询主要存在于前几个读取原始表记录的job，而后面的任务相差并不大，进而导致查询时间的差异减小，从后面单个宽表的测试结果可以看到，Text和其它两种列式存储格式的性能上的差异还是挺大的。

测试2(执行其中10条查询)结果：



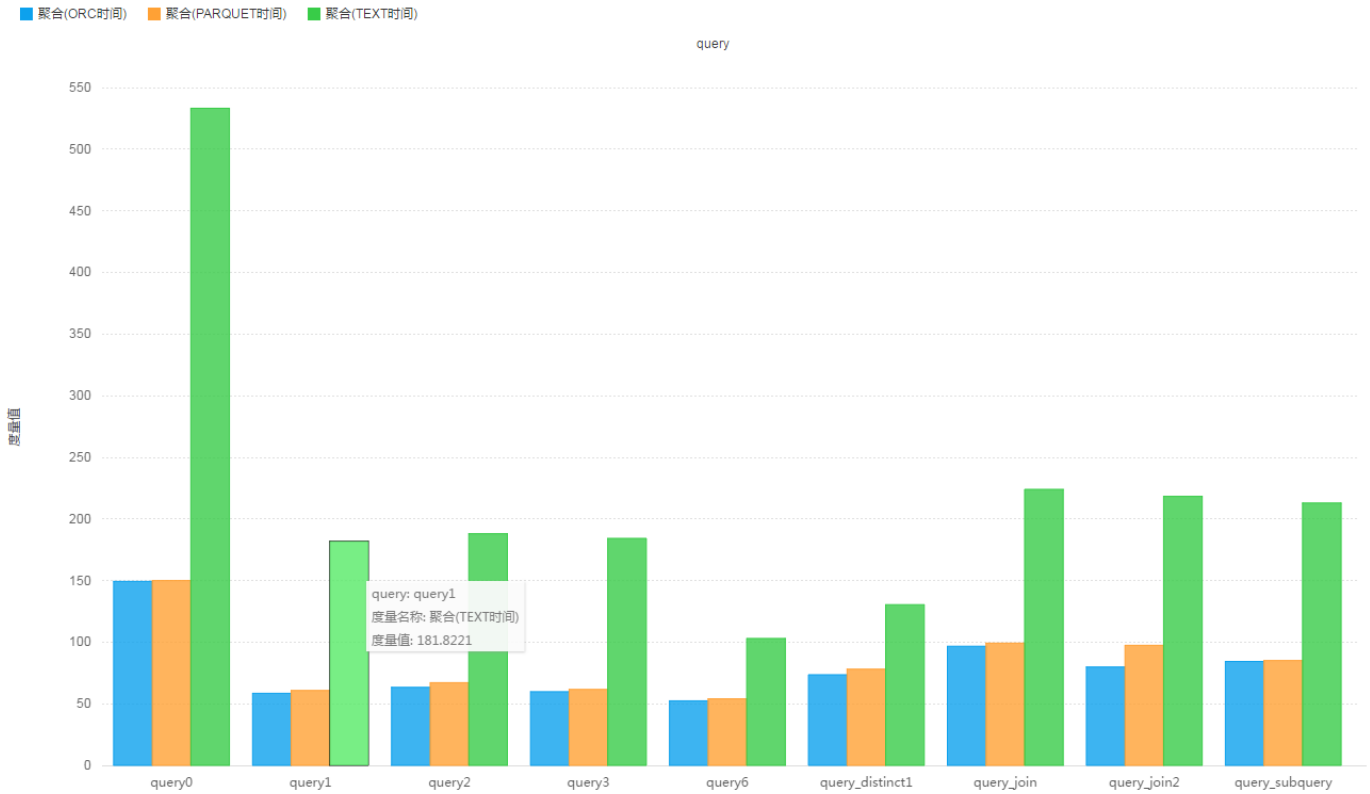
说明：和测试1类似，相差不大，对于query82，ORC格式差很多？！

场景二

该场景中只涉及到了一个宽表，并且没有任何分区字段，store*sales*wide_table表记录数：263,704,266，表大小为：

- 原始Text格式，未压缩： 149.0 G
- ORC格式，默认压缩： 10.6 G 比store_sales表还小？
- PARQUET格式，默认压缩： 12.5 G 比store_sales表还小？

查询测试结果：

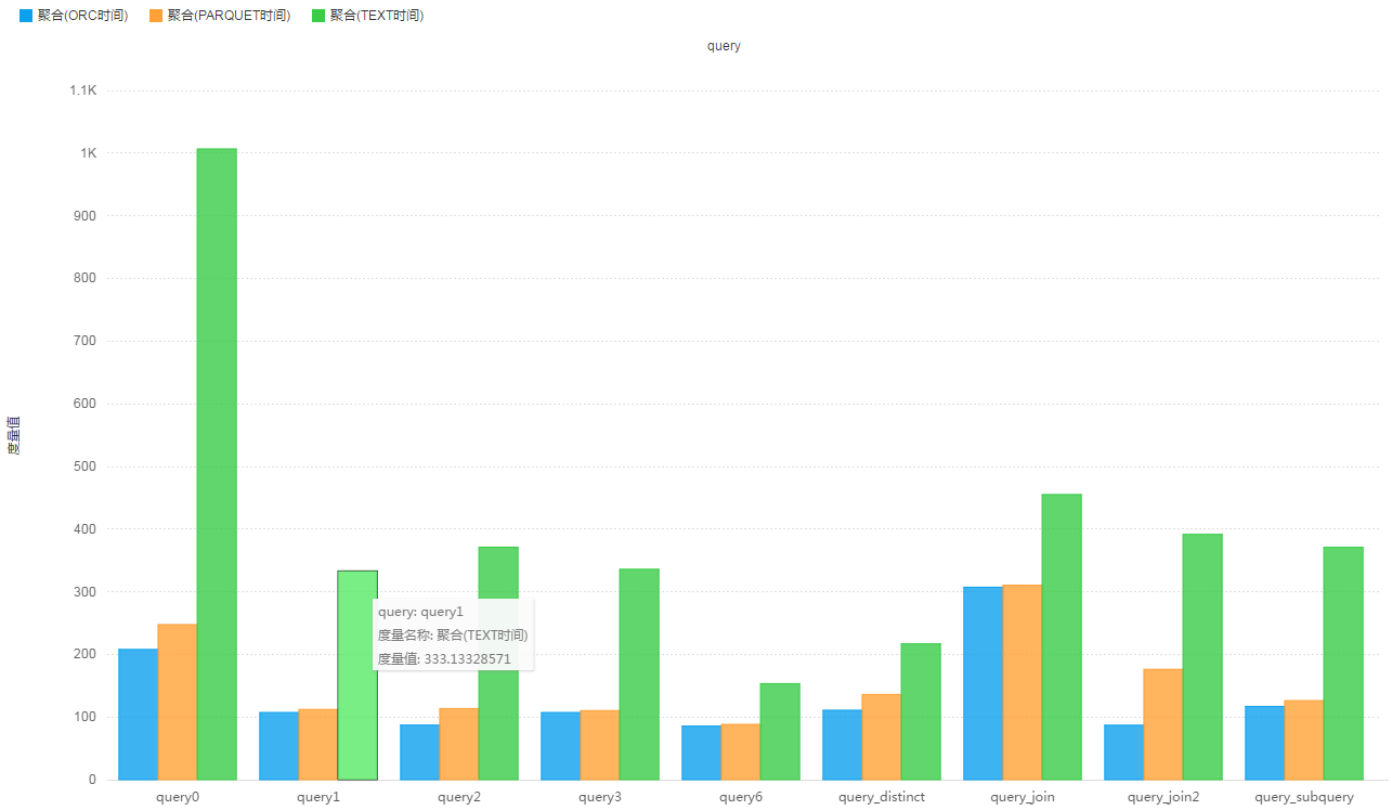


场景三

该场景中只涉及一个嵌套的宽表，没有任何分区字段，store*sales*wide_table*one*nested表记录数：263,704,266，表大小为：

- 原始Text格式，未压缩： 245.3 G
- ORC格式，默认压缩： 10.9 G 比store_sales表还小？
- PARQUET格式，默认压缩： 29.8 G

查询测试结果：



场景四

该场景中只涉及一个多层嵌套的宽表，没有任何分区字段，store*sales*wide_table*more*nested表记录数：263,704,266，表大小为：

- 原始Text格式，未压缩： 222.7 G
- ORC格式，默认压缩： 10.9 G 比store_sales表还小？
- PARQUET格式，默认压缩： 23.1 G 比一层嵌套表store*sales*wide_table*one*nested要小？

查询测试结果：

六、总结

本文使用HIVE对三种不同的文件存储格式——Text、ORC和Parquet进行了对比测试，从测试结果来看，星状模型对于数据分析场景并不是很合适，多个表的join会大大拖慢查询速度，并且不能很好的利用列式存储带来的性能提升，在使用宽表的情况下，ORC文件格式在存储空间上要远优于Text格式，较之于PARQUET格式有一倍的存储空间提升，在导数据（insert into table select 这样的方式）方面ORC格式也要优于PARQUET，在最终的查询性能上可以看到，无论是无嵌套的扁平式宽表，或是一层嵌套表，还是多层嵌套的宽表，两者的查询性能相差不多，较之于Text格式有2到3倍左右的提升，在query_join2这个查询中，使用宽表和另外一个维度表执行join查询的时候，ORC要优于PARQUET格式。

另外，通过对比场景二和场景三的测试结果，可以发现扁平式的表结构要比嵌套式结构的查询性能有所提升，所以如果选择使用大宽表，则设计宽表的时候尽可能的将表设计的扁平化，减少嵌套数据。

通过这三种文件存储格式的测试对比，ORC文件存储格式无论是在空间存储、导数据速度还是查询速度上表现的都较好一些，并且ORC可以一定程度上支持ACID操作，社区的发展目前也是Hive中比较提倡使用的一种列式存储格式，另外，本次测试主要针对的是Hive引擎，所以不排除存在Hive与ORC的敏感度比PARQUET要高的可能性。