Spark 2.0 SQL Testing

Databricks published its preview version of Spark 2.0. The open source community has high expectation on it, so it got great numbers of downloading too. In the [formal blog](https://databricks.com/blog/2016/05/11/apache-spark-2-0-technical-preview-easier-faster-and-smarter.html), Easier/Faster/Smarter is the most innovation part of it. I would like to understand it as more SQL support, better performance, adaptive streaming computing and batch job unified programming model.

From practice perspective, these features are definitely most important parts to developers. It could be because Databricks hope this product more commercial. While what is the fact and comparing some commercial big data product, say TDH, how about its performance? I and my team did some deeper testing on it.

Testing Environment:

4 x86 Server, each server hardware config as following:

CPU：2X6 core, E5 2620 v2, 2.10GHz

Memory: 128G

Network bandwidth: 1G

Hard Drive Raid: 3X3T

Impression

During the test, we found it’s still a problem in Spark SQL’s stability. We spent 5 times more time than we expected. There is something unstable, especially in some continuous batch job running and we couldn’t finish it in a continuous testing.

To compare, we selected TDH 4.5 to test. We chose TPC-DS Benchmark by Databricks blog.

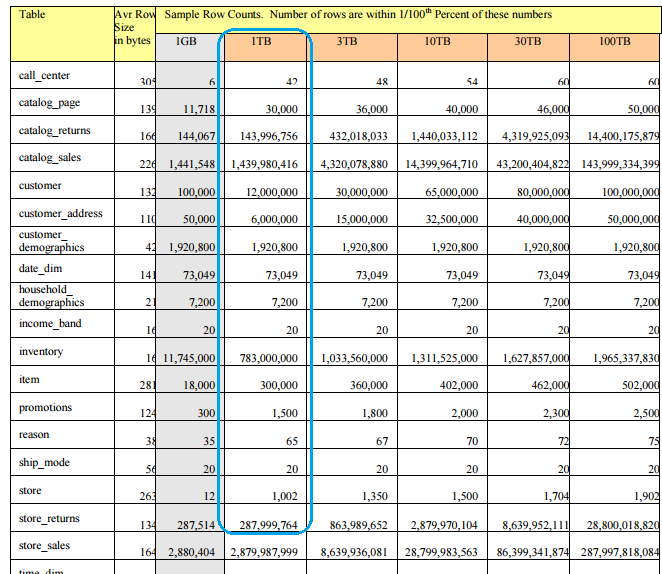
First, Spar 2.0 supports better TPC-DS than Spark 1.6 does. Spark 1.6 may run only about 70 queries while 2.0 can do 99 ones. In the jira of Spark-12540 we can see the SQL features that TPC-DS needed are being accomplished. From the Jira we can also see the ideas of implementing SQL compatibility. They are working in Catalyst to add adapters for different SQL features, and then convert it to the existing SQL execution plan with same syntax. This generally looks good while I think there could be some limitations such as it’s difficult to deal with different corner cases, and is not friendly to sub clause queries. And some SQLs could be using different adapter then the result could be unexpected.

During testing, we found for some non-TPC-DS SQL transactions, such as “<, >” or sub clause queries, Spark SQL doesn’t support it still. Therefore, for a complex production environment, it could be a reason not to select Spark.

Plus, Add/Modify/Delete transactional SQLs are not at the roadmap of Spark community. TDH started supporting standard Sql 99 from January 2015, including complete A/M/D supporting and passed TPC-DS testing completely. TDH also does support distributed transaction to make sure consistency during the data processing.

Spark 2.0 SQL seems still long way to go for production requirements.

TPC-DS Performance Comparison

****

1T TPC-DS data set was chosen as testing source, to simulate production environments. During testing, we opened Tungsten engine and whole-stage code generation to boost the best performance of Spark 2.0. We also chose Inceptor 4.5 for testing comparison and set inceptor.optimizer.mode=true. In general, Inceptor is advanced about 3 times than Spark 2.0 in performance. There is some good point in single SQL execution but in general it’s behaving not as expected.

From Tungsten Engine, Spark improved system stability, but some problem were there. Some common problems are memory leaking, block manager dead lock and overclocking memory are killed by Yarn. After Spark SQL long running, GC doesn’t look good.

This is a concern to our testing. As per TPS-DS spec, we need continuous test all SQL multiple times and then calculate each query’s performance, while Spark Sql 2.0 preview version can’t finish 1T data set testing. We have to do it from failure points many times to finish.

How’s Optimizer

We selected whole-stage code generation and vetorized parquet reader to compare.

Whole-stage code generation is to solve the performance caused by scala code calling stack too deep and too many virtual functions calling. It makes a few classes code to inline as a new class function to reduce overhead.

We can see this performance got 10 times improved when we run a simple dedicated SQL in the system. However, if the system is running other sqls parallel or the SQL got complicated the performance improvement is not obvious.

We selected the first 10 quries in TPC-DS to do vectorized parquet reader optimizer testing and comparison. We can see some improvement but not exceed 10%.

Conclusion

It seems we would like to stay on scala/java programming and leave spark SQL out of production still after Spark 2.0.