Performance Test
Hive/Spark (Parquet) vs. Azure Databricks Delta

Compare performance of Hive/Spark tables
(with underlying Parquet file format)
with
Azure Databricks Delta tables

Prepared by
DM Jumpstart Engineering Team
(askdmjfordmtools@microsoft.com)

Disclaimer
The High-Level Architecture, Migration Dispositions and guidelines in this document is developed in consultation and collaboration with Microsoft Corporation technical architects. Because Microsoft must respond to changing market conditions, this document should not be interpreted as an invitation to contract or a commitment on the part of Microsoft.

Microsoft has provided generic high-level guidance in this document with the understanding that MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE INFORMATION CONTAINED HEREIN.

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.
Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.
© 2019 Microsoft. All rights reserved.

Note: The detail provided in this document has been harvested as part of a customer engagement sponsored through the Azure Data Services Jumpstart Program.

What is Databricks Delta?

Delta is an optimized Spark table that brings data reliability and performance optimizations to cloud data lakes. Making your data lake faster and more reliable – accelerating innovation.

Databricks Runtime *is the most optimized version of...* 

A Delta Table *is the most optimized version of...* A Spark/Hive Table

https://docs.azuredatabricks.net/delta/delta-intro.html

Databricks Delta - Unified Data Management

Delta is an optimized Spark table, **unifying both streaming and batch data**, that stores data in the open-source Parquet format in blob storage.

- **Improves performance** by managing metadata about your table
  - Automated file management to make engineering simple (compaction)
  - Creates and maintains indexes to speed up queries 10x – 100x

- **Improves data reliability & simplifies pipelines** by adding data management capabilities
  - Transactional guarantees (ACID)
  - Schema enforcement ensures data integrity
  - UPSERTs/Deletes allow for changing data

What is Databricks Delta?

- The core abstraction of Databricks Delta is an optimized Spark table that:
  - Stores data as Parquet files in DBFS
  - Maintains a transaction log that efficiently tracks changes to the table
  - Data can be read and written stored in delta format using Spark SQL batch and streaming APIs

- With the addition of the transaction log, as well as other enhancements, Databricks Delta offers some significant benefits:
  - Acid Transactions – a big one for consistency
  - Multiple writers can simultaneously modify a dataset and see consistent views
  - Writers can modify a dataset without interfering with jobs reading the dataset
  - Faster Read Access – automatic file management organizes data into large files that can be read efficiently
  - Statistics that enable speeding up reads by 10-100x and data skipping avoids reading irrelevant information
  - This is not available in Apache Spark, only in Databricks

Databricks Delta – Query Performance

- DATA INDEXING
  - Delta creates and maintains indexes of the ingested data
  - This speed up the querying dramatically
- DATA SKIPPING
  - Delta maintains file statistics so that data subsets relevant to the query are used instead of entire tables — this partition pruning avoids processing data that is not relevant to the query
  - Multi-dimensional clustering (using Z-ordering algorithm) is used to enable this
  - This technique is particularly helpful in the case of complex queries
- COMPACTION
  - Often, especially in the case of streaming data, a large number of small files are created as data is ingested
  - Storing and accessing these small files can be processing-intensive, slow and inefficient from a storage utilization perspective
  - Delta manages file sizes (i.e. compacts or combines multiple small files into more efficient larger ones) to speed up query performance
- DATA CACHING
  - Accessing data from storage repeatedly can slow query performance
  - Delta automatically caches highly accessed data to speed access for queries improving performance by an order of magnitude

Performance Test

- Cluster Size – 8 Nodes
  - Each node is Standard_DS3_v2 – with 4 cores, 14 GB
  - No autoscaling for the cluster
- Size of parquet files – ~32 GB (uncompressed ~340 GB)
- Statistics – Rows: 635,586,654 – Columns: 84
- Databricks Runtime – 4.3 beta (includes Apache Spark 2.3.1, Scala 2.11)

Please note, although this data volume is not huge enough to represent the big data scenario however it will give an idea of the performance differences and how you can carry out this exercise in your environment with your big data.

Performance Test

Hive External Table – Parquet

Delta Table

Please note, the given time difference between Hive/Spark table (with underlying Parquet) and Delta table doesn't look huge, however it would be much better with bigger dataset, especially when you leverage delta optimization (discussed later).

Performance Test

Hive External Table – Parquet

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)
FROM EDW.pos_hive_big
WHERE BusDt BETWEEN '2018-07-10' AND '2018-07-25'
GROUP BY BusDt
ORDER BY BusDt
```

(1) Spark Jobs

BusDt	sum(GrossSalesLclAmt)
2018-07-10	
2018-07-11	
2018-07-12	
2018-07-13	
2018-07-14	
2018-07-15	
2018-07-16	
2018-07-17	

Sanitized

Command took 32.66 seconds -- by a :om at 8/9/2018 9:26:14 PM on arali_hive2

Delta Table

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)
FROM EDW.pos_delta_big
WHERE BusDt BETWEEN '2018-07-10' AND '2018-07-25'
GROUP BY BusDt
ORDER BY BusDt
```

(2) Spark Jobs

BusDt	sum(GrossSalesLclAmt)
2018-07-10	
2018-07-11	
2018-07-12	
2018-07-13	
2018-07-14	
2018-07-15	
2018-07-16	
2018-07-17	

Sanitized

Command took 31.32 seconds -- by a om at 8/9/2018 9:27:31 PM on arali_hive2

Please note, the given time difference between Hive/Spark table (with underlying Parquet) and Delta table doesn't look huge, however it would be much better with bigger dataset, especially when you leverage delta optimization (discussed later).

Performance Test

Hive External Table – Parquet

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)
FROM EDW.pos_hive_big
WHERE CntryCd = 'CA'
GROUP BY BusDt
ORDER BY BusDt
```

(1) Spark Jobs

BusDt	sum(GrossSalesLclAmt)
2018-07-01	
2018-07-02	
2018-07-03	
2018-07-04	
2018-07-05	
2018-07-06	
2018-07-07	
2018-07-08	

Sanitized

Command took 1.25 minutes -- by a :om at 8/9/2018 10:26:38 PM on arali_hive2

Delta Table

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)
FROM EDW.pos_delta_big
WHERE CntryCd = 'CA'
GROUP BY BusDt
ORDER BY BusDt
```

(2) Spark Jobs

BusDt	sum(GrossSalesLclAmt)
2018-07-01	
2018-07-02	
2018-07-03	
2018-07-04	
2018-07-05	
2018-07-06	
2018-07-07	
2018-07-08	

Sanitized

Command took 1.16 minutes -- by : :om at 8/9/2018 10:29:33 PM on arali_hive2

Please note, the given time difference between Hive/Spark table (with underlying Parquet) and Delta table doesn't look huge, however it would be much better with bigger dataset, especially when you leverage delta optimization (discussed later).

Performance Test – Optimization with Delta

Databricks Delta supports the ability to optimize the layout of data stored in DBFS with these two algorithms:

- Compaction (bin-packing) – Improves the speed of read queries from a table by coalescing small files into larger ones from large number of small files to small number of larger files.
- ZOrdering – Colocates related information in the same set of files. This co-locality is automatically used by Databricks Delta data-skipping algorithms to dramatically reduce the amount of data that needs to be read.

Best Practice - If you have a large amount of data, we suggest that you limit it to new data for optimization, using partition filters (WHERE clause) when possible for ZOrdering.

https://docs.azuredatabricks.net/delta/optimizations.html

Performance Test

Hive External Table – Parquet

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)
FROM EDW.pos_hive_big
WHERE CntryCd = 'CA'
GROUP BY BusDt
ORDER BY BusDt
```

(1) Spark Jobs

BusDt	sum(GrossSalesLclAmt)
2018-07-01	
2018-07-02	
2018-07-03	
2018-07-04	
2018-07-05	
2018-07-06	
2018-07-07	
2018-07-08	

Sanitized

Command took 1.25 minutes -- by a... om at 8/9/2018 10:26:38 PM on arali_hive2

Delta Table

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)
FROM EDW.pos_delta_big
WHERE CntryCd = 'CA'
GROUP BY BusDt
ORDER BY BusDt
```

(2) Spark Jobs

BusDt	sum(GrossSalesLclAmt)
2018-07-01	
2018-07-02	
2018-07-03	
2018-07-04	
2018-07-05	
2018-07-06	
2018-07-07	
2018-07-08	

Sanitized

Command took 18.43 seconds -- by ar... m at 8/9/2018 11:13:16 PM on arali_hive2

Please note, performance of your query would vary here based on amount of data requested by your query.
You can find more information here: https://docs.azuredatabricks.net/delta/optimizations.html

Benefits of using Databricks Delta

- New data/partition gets reflected automatically to all the clusters – no need for repairing table or deleting/adding partitions to the table
- No need to refresh table metadata in local cache with REFRESH table command
- Optimization for performance and cost
- Transactional consistent –
  - When rebuilding a partition fails, data remains consistent.
  - Multiple users can modify the dataset simultaneously and see the consistent view
- A single delta table can be target for both batch and stream data ingestion – no need to have separate pipelines for streaming and batch

Benefits of using Databricks Delta – Cont'd

- You can write MERGE statement merge (insert/update) data into Databricks Delta table
- Databricks Delta table also simplifies operational overhead in comparison with Hive External table as explained next

Using Hive External tables – connected to external metadata store (Azure SQL Database)

Data Engineering – Worspace/Cluster	Business Group – Workspace/Cluster
1 df2.write.format("parquet").partitionBy("BusDt").save("/mnt/workspaces/Personal/arali/pos/parquet_hive_small") ▸ (1) Spark Jobs Command took 6.46 minutes -- by : 8/8/2018 3:31:26 PM on arali_hive1	
1 %sql 2 CREATE TABLE EDW.pos_hive_small 3 USING parquet 4 OPTIONS (path "/mnt/workspaces/Personal/arali/pos/parquet_hive_small") ▸ (1) Spark Jobs OK Command took 3.32 seconds -- by ar :om at 8/8/2018 3:38:53 PM on arali_hive1	
1 %sql 2 SELECT count(*) from EDW.pos_hive_small ▸ (1) Spark Jobs count(1) 0 Command took 1.18 seconds -- by ar om at 8/9/2018 11:20:27 AM on arali_hive1	1 %sql 2 SELECT count(*) from EDW.pos_hive_small ▸ (1) Spark Jobs count(1) 0 Command took 1.20 seconds -- by a m at 8/9/2018 11:22:07 AM on arali_hive2
1 %sql 2 MSCK REPAIR TABLE EDW.pos_hive_small OK Command took 0.85 seconds -- by ar om at 8/9/2018 11:22:35 AM on arali_hive1	

Data Engineering – Worspace/Cluster

```
1 %sql
2 SELECT count(*) from EDW.pos_hive_small
```

▸ (1) Spark Jobs

count(1)
22701090

Command took 12.88 seconds -- by ar om at 8/9/2018 11:23:01 AM on arali_hive1

```
1 df2.write.format("parquet").partitionBy("BusDt").mode("Append").save("/mnt/workspaces/Personal/arali/pos/parquet_hive_small")
```

▸ (1) Spark Jobs

Command took 1.91 minutes -- by /9/2018 11:25:30 AM on arali_hive1

```
1 %sql
2 SELECT count(*) from EDW.pos_hive_small
```

▸ (1) Spark Jobs

count(1)
22701090

Command took 8.22 seconds -- by ar om at 8/9/2018 11:29:46 AM on arali_hive1

```
1 %sql
2 MSCK REPAIR TABLE EDW.pos_hive_small
```

OK

Command took 0.96 seconds -- by a com at 8/9/2018 11:32:50 AM on arali_hive1

```
1 %sql
2 SELECT count(*) from EDW.pos_hive_small
```

▸ (1) Spark Jobs

count(1)
43312969

Command took 4.42 seconds -- by a com at 8/9/2018 11:33:30 AM on arali_hive1

Business Group – Workspace/Cluster

```
1 %sql
2 SELECT count(*) from EDW.pos_hive_small
```

▸ (1) Spark Jobs

count(1)
22701090

Command took 13.02 seconds -- by a om at 8/9/2018 11:23:40 AM on arali_hive2

```
1 %sql
2 SELECT count(*) from EDW.pos_hive_small
```

▸ (1) Spark Jobs

count(1)
22701090

Command took 5.08 seconds -- by ar om at 8/9/2018 11:30:25 AM on arali_hive2

```
1 %sql
2 SELECT count(*) from EDW.pos_hive_small
```

▸ (1) Spark Jobs

count(1)
43312969

Command took 7.21 seconds -- by a om at 8/9/2018 11:34:01 AM on arali_hive2

Data Changes – REFRESH TABLE

```
1 %sql
2 SELECT BusDt, count(*) from EDW.pos_hive_small_wopartition GROUP BY BusDt
```

▸ (1) Spark Jobs

⊞Error in SQL statement: SparkException: Job aborted due to stage failure: Task 0 in stage 35.0 failed 4 times, most recent failure: Lost task 0.3 in stage 35.0 (TID 644, 10.139.64.6, executor 1): java.io.File
dbfs:/mnt/workspaces/Personal/arali/pos/parquet_hive_small_wopartition/part-00049-tid-3570897718831618244-b8ec4699-c44b-45b0-8202-6056a4be89ab-670-c000.snappy.parquet
It is possible the underlying files have been updated. You can explicitly invalidate the cache in Spark by running 'REFRESH TABLE tableName' command in SQL or by recreating the Dataset/DataFrame involved.
at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.org$apache$spark$sql$execution$datasources$FileScanRDD$$anon$$readFile(FileScanRDD.scala:211)
at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.org$apache$spark$sql$execution$datasources$FileScanRDD$$anon$$createNextIterator(FileScanRDD.scala:380)
at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1$$anonfun$prepareNextFile$1.apply(FileScanRDD.scala:298)
at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1$$anonfun$prepareNextFile$1.apply(FileScanRDD.scala:294)

```
1 %sql
2 REFRESH TABLE EDW.pos_hive_small_wopartition
```

OK

Command took 0.89 seconds -- by ar[redacted].com at 8/9/2018 11:37:56 AM on arali_hive2

Using Databricks Delta – connected to external metadata store (Azure SQL Database)

Data Engineering – Worspace/Cluster

```
df2.write.format("delta").partitionBy("BusDt").save("/mnt/workspaces/Personal/arali/pos/parquet_delta_sma
```

▸ (2) Spark Jobs

Command took 7.32 minutes -- by arali@starbucks.com at 8/8/2018 10:27:16 AM on arali_delta

```
%sql
DROP TABLE IF EXISTS pos_delta;
CREATE TABLE pos_delta
USING DELTA
LOCATION "/mnt/workspaces/Personal/arali/pos/parquet_delta_small"
```

OK

Command took 0.61 seconds -- by arali@starbucks.com at 8/8/2018 10:37:11 AM on arali_delta

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)  FROM pos_delta GROUP BY BusDt
```

▸ (6) Spark Jobs

BusDt	sum(GrossSalesLclAmt)	sum(NetRevLclAmt
2018-07-20	Sanitized	

Command took 12.90 seconds -- : 8/8/2018 10:40:59 AM on arali_delta

Business Group – Workspace/Cluster

```
%sql
SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)  FROM pos_delta GROUP BY BusDt
```

▸ (6) Spark Jobs

BusDt	sum(GrossSalesLclAmt)	sum(NetRevLclA
2018-07-20	Sanitized	

Command took 19.03 seconds -- by : 2018 10:42:17 AM on arali_delta

Data Engineering – Workspace/Cluster

```
1 df2.write.format("delta").mode("Append").save("/mnt/workspaces/Personal/arali/pos/parquet_del
```

▸ (2) Spark Jobs

Command took 2.17 minutes -- by arali@starbucks.com at 8/8/2018 10:45:43 AM on arali_delta

```
1 %sql
2 SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)  FROM pos_delta GROUP BY BusDt
```

▸ (6) Spark Jobs

BusDt	sum(GrossSalesLclAmt)	su
2018-07-20		
2018-07-21		

Sanitized

Command took 11.99 seconds -- by : 8/2018 10:48:33 AM on arali_delta

Business Group – Workspace/Cluster

```
1 %sql
2 SELECT BusDt, SUM(GrossSalesLclAmt), SUM(NetRevLclAmt)  FROM pos_delta GROUP BY BusDt
```

▸ (7) Spark Jobs

BusDt	sum(GrossSalesLclAmt)	sum(
2018-07-20		
2018-07-21		

Sanitized

Command took 18.46 seconds -- m at 8/8/2018 10:49:17 AM on arali_delta

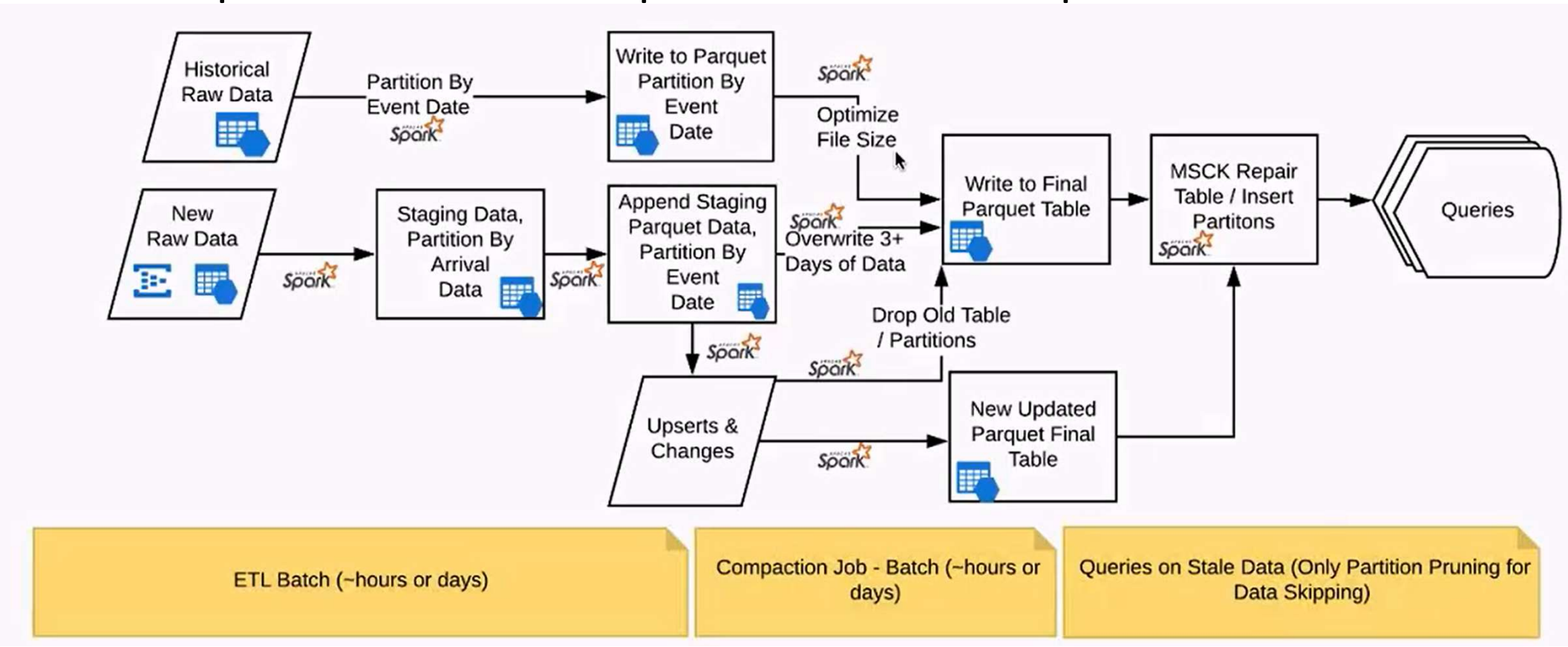
Feedback and suggestions

If you have feedback or suggestions for improving this data migration asset, please contact the Data Migration Jumpstart Team (askdmjfordmtools@microsoft.com). Thanks for your support!

Note: For additional information about migrating various source databases to Azure, see the Azure Database Migration Guide.

Appendix

Data Pipeline – with Spark/Hive/Parquet



Data Pipeline – simplified with Delta