Lakehouse Architecture

==========

Datalake + Datawarehouse

Lakehouse intends to provide you best of both worlds - Datalake & Datawarehouse.

Datalake benefits

==========

Inexpensive

Scalable

All kinds of data - structured, unstructured and semi structured.

open file formats - parquet

Challenges of datalake

==========

do not support ACID guarantees

not suitable for Reporting/BI workloads..

Datascience, Machine learning , Reporting, BI, Transactional

DataLake -> Machine Learning & Data Science

A subset of Data from DataLake used to go to Datawarehouse from where we used to

serve Reporting & BI usecases.

2 Tier architecture is called as Modern Datawarehouse architecture

1. Datalake - 100 TB (Machine learning, Data Science)

2. Datawarehouse - 20 TB (BI & Reporting)

Datalake -> Datawarehouse

Challenges with this 2 tier architecture

==========

(modern datawarehouse architecture)

1. we require 2 different systems

datalake and a datawarehouse

2. Data Duplication

3. Increased cost

4. Datalake -> Datawarehouse (ETL)

5. Stale data

Lakehouse Architecture

==========

Datalake

+

Datawarehouse

1. Inexpensive

2. All kinds of data

3. Open file Formats

4. Reduced data duplication

5. Reduced ETL activity

6. Should be able to handle all kind of workloads

Databricks Lakehouse Architecture

==========

Delta Engine - it adds the speed to the below things

Transactional Layer - Delta Lake (Delta Logs)

Storage Layer - Data Lake (S3, ADLS gen2, GCS)

Delta Engine - Session 1

==========

Data Skipping using Stats

orders table - 100 files

as part of your delta logs each file will contain some metadata.

file1

=====

column_name, min value, max value

order_id, 1, 100

customer_id, 1, 345

amount, 65, 10000

file2

=======

column_name, min value, max value

order_id, 101, 200

customer_id, 459, 900

amount, 20, 5000

select * from orders where order_id = 179

dbfs:/databricks-datasets/nyctaxi/tripdata/yellow/yellow_tripdata_2009-01.csv.gz

%sql

create database trip_db

504 mb in compressed format but in csv

trip_df =

spark.read.format("csv").option("header","true").option("inferSchema","true").load("d

bfs:/databricks-datasets/nyctaxi/tripdata/yellow/yellow_tripdata_2009-01.csv.gz")

display(trip_df)

trip_df.count()

14092413

we want to create 2 tables

one table should have data in parquet

other table should have data in delta

trip_df.repartition(20).write.format("delta").saveAsTable("trip_db.trips_delta")

Describe detail trip_db.trips_delta

/user/hive/warehouse/<databasename.db>/<tablename>

trip_df.repartition(20).write.format("parquet").saveAsTable("trip_db.trips_parquet")

%sql

Describe extended trip_db.trips_parquet

%sql

select min(fare_amt), max(fare_amt) from trip_db.trips_delta

%sql

select min(fare_amt), max(fare_amt) from trip_db.trips_parquet

%sql

select count(*) from trip_db.trips_delta

%sql

select count(*) from trip_db.trips_parquet

%sql

select * from trip_db.trips_delta where total_amt = 234

%sql

select * from trip_db.trips_parquet where total_amt = 234

%sql

select * from trip_db.trips_delta where total_amt > 232

%sql

select * from trip_db.trips_parquet where total_amt > 232

Delta Engine - Session 2

==========

Delta Cache

==========

Data Lake -> cache it on your worker machine local disk

its stored in a format which is really quick to retrieve.

2 ways to enable to delta cache..

1. use specific type of machines to get that... (Delta accelerated VMs)

2. set a property in case of normal cluster

spark.conf.get("spark.databricks.io.cache.enabled")

%sql

select sum(trip_distance),sum(total_amt) from trip_db.trips_delta group by

vendor_name

The system will track whenever the data changes in file or when the cluster restarts..

it will automatically evict the cache..

delta cache is applicable only on parquet based formats

spark based cache is applicable for all the file formats

delta cache is ideally a lot faster than spark based cache.

%sql

Cache

select * from trip_db.trips_delta

spark.conf.set("spark.databricks.io.cache.enabled","true")

Delta accelerated VMs with cache enable

Delta Engine - session 3

==========

challenges with 2 tier architecture - Modern Datawarehouse architecture.

Lakehouse architecture..

Delta Engine

DeltaLake - transaction logs

DataLake

Optimizations

==========

1. Data Skipping using stats

each data file will have metadata stored..

_delta_log folder

5 columns in our table

file1 - emp(1,5) , dept(100,150)

file2 - emp (100,150)

file3

file4

file5

2. Delta Cache

3. Small file problem

table - 10000 small files - 10 records each

4 big files - 25000 records each

Delta tables.. inserts, updates, deletes...

Inserts then you get more files...

1000 inserts...

1000 small files...

Created a database

loaded the csv file into the dataframe

we are creating a delta table with 500 files for each partition... 1500 files

%sql

create database taxidb;

%fs ls dbfs:/databricks-datasets/nyctaxi/tripdata/yellow

trip_df =

spark.read.format("csv").option("header","true").option("inferSchema","true").load("d

bfs:/databricks-datasets/nyctaxi/tripdata/yellow/yellow_tripdata_2009-01.csv.gz")

display(trip_df)

trip_df.repartition(500).write.format("delta").partitionBy("vendor_name").saveAsTable(

"taxidb.trips_delta")

%sql

DESCRIBE DETAIL taxidb.trips_delta

%fs head

dbfs:/user/hive/warehouse/taxidb.db/trips_delta/_delta_log/00000000000000000000.jso

n

%sql

select * from taxidb.trips_delta where total_amt = 20

%sql

OPTIMIZE taxidb.trips_delta

%sql

select * from taxidb.trips_delta where total_amt = 20

how to solve this small file problem

Compaction/bin-packing

==========

take multiple small files and club it into larger files

OPTIMIZE command can compact the delta files - upto 1 GB

we should perform optimize at periodic intervals...

but it is a resource intensive operation. Perform this in the non peak hours..

will OPTIMIZE help you in data skipping?

its not meant to give data skipping as optimization..

Delta Engine - session 4

==========

we learnt bin packing/ compaction - Small file Problem

OPTIMIZE command

NO data skipping is achieved using optimize.

Z-ordering is something which we can use along with OPTIMIZE to achieve data

skipping.

consider you have a employee table..

emp_id, emp_name, dob, salary....

500 different files...

file1 - 1,5,8 (1,8)

file2 - 2,4,9 (2,9)

file3 - 3,6,10 (3,10)

select * from employee where emp_id = 4

file1 - 1,2,3 (1,3) skipped

file2 - 4,5,6 (4,6) check this file

file3 - 8,9,10 (8,10) skipped

select * from employee where emp_id = 4

we can consider this like your clustered index in your database

Z-ordering is a technique to colocate related information in the same set of files.

this co-locality is used by databricks to achive data skipping and give you

performance benefits.

this will drastically reduce the amount of data that needs to be scanned.

trip_df =

spark.read.format("csv").option("header","true").option("inferSchema","true").load("d

bfs:/databricks-datasets/nyctaxi/tripdata/yellow/yellow_tripdata_2009-01.csv.gz")

run the below 2 times

==========

trip_df.repartition(200).write.mode("append").format("delta").saveAsTable("taxidb.trip

s_delta_new")

%sql

describe history taxidb.trips_delta_new

%sql

select * from taxidb.trips_delta_new where passenger_count = 4

400 files

file1 - 1,1,1,1,1,1,1,1,2,2,2,2,2

file2 - 2,2,2,2,2,3,3,3,3

%sql

optimize taxidb.trips_delta_new zorder by (passenger_count)

%sql

select * from taxidb.trips_delta_new where passenger_count = 4

the columns used in your filters, joins, groupby... you can zorder..

Data Skipping

==========

Data Skipping using stats...

Z ordering then we can definitely skip more data...

bin packing , z ordering - how to physically keep the data on the disk..

Partitioning and Bucketing

==========

country column can be a partition column...

20 different countries...

20 different folders will be created..

the cardinality of the partitioning column should be less...

Bucketing

==========

empid

16 buckets

1 - 1

2 - 2

3 - 3

19 - 3

25 - 9

Delta Engine - Session 5

==========

Optimize - bin packing/compaction

Z ordering - colocate the data based on Z ordered column

Vacuum

=======

employee table

file1 (version 0)

======

101, Sumit, 10000

102, Satish, 20000

103, Kapil, 30000

000000.json

============

add file1

file2 (version1)

======

104, Ram, 40000

000001.json

============

add file2

file3 (version2)

======

101, Sumit, 15000

102, Satish, 20000

103, Kapil, 30000

000001.json

============

add file3

remove file1

VACUUM command - It removes the data files

1. No longer referenced in the latest transaction logs

2. and older than a retention threshold (7 days)

the vacuum commands affects the time travel..

%sql

set spark.databricks.delta.retentionDurationCheck.enabled = false

%sql

VACUUM taxidb.trips_delta RETAIN 1 HOURS DRY RUN

%sql

VACUUM taxidb.trips_delta RETAIN 1 HOURS

we should perform VACUUM periodically

%sql

select * from taxidb.trips_delta version as of 0

Optimize

=========

auto optimization - optimized writes, auto compaction

Optimized writes (before writing to the disk)

==========

100 GB of data in S3

800 blocks

if we are creating a dataframe, 800 partitions

df.filter

800 partitions will be the output (800 tasks)

input 128 mb

output 5 mb

if we write to disk we will get 800 small files...

TBLPROPERIES (delta.autoOptimize.optimizeWrite = true)

TBLPROPERIES (delta.autoOptimize.autoCompact = true)

optimize write

==========

it creates bigger files before writing

auto compact

==========

once the files are written to the disk it will try to compact them and create bigger files.

auto compact only works when there are more than 50 small files...

t1 ->

t2 ->

t3 -> t801

t4 ->

. ->

.

.

t800

the approx file size it will try to create will be of ~128 mb

Delta Engine - Session 6

==========

Photon Query Engine - photon is a native vectorized engine developed in c++

this dramatically improves the query performance..

some parts of spark engine they rewrote in c++

takes benefits of the modern hardware to give the best performance..

what kind of queries will benefit from photon query engine...

photon engine is meant to do better with compute intensive queries..

queries which are short might not get significant performance gains..

Costly.. (software cost doubles up)

works with only parquet or delta tables...

delta cache

data skipping using stats

Optimize - bin packing

Z ordering - data skipping

VACUUM

partition/bucketing - data skipping

optimize write, auto compact

trip_df =

spark.read.format("csv").option("header","true").option("inferSchema","true").load("d

bfs:/databricks-datasets/nyctaxi/tripdata/yellow/yellow_tripdata_2009-01.csv.gz")

trip_df.repartition(20).write.format("delta").saveAsTable("trips_delta")

%sql

select sum(trip_distance), sum(total_amt) from trips_delta group by

vendor_name,payment_type