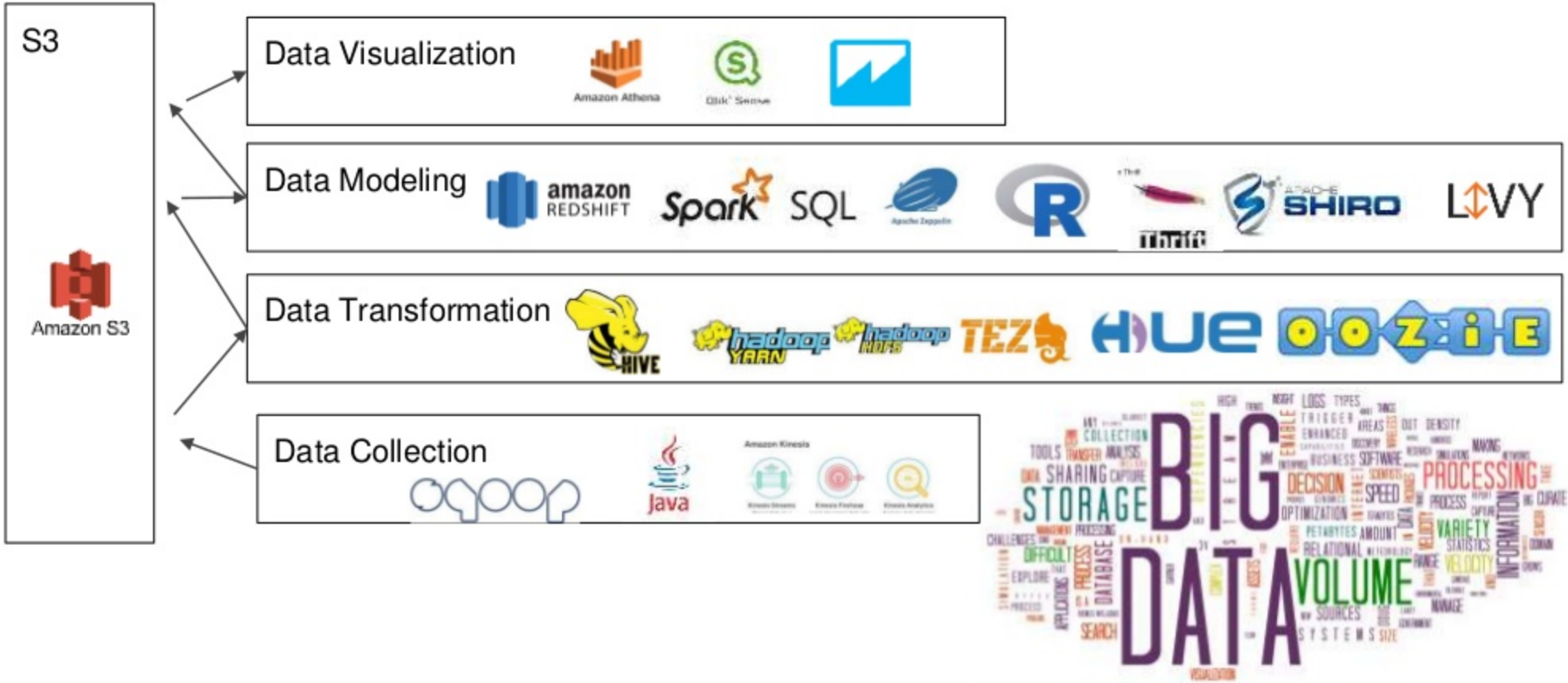
AWS Big Data Demystified #2 Athena, Spectrum, EMR, Hive

Omid Vahdaty, Big Data Ninja

TODAY'S BIG DATA APPLICATION STACK PaaS and DC...

Big Data Generic Architecture | Summary



Agenda for Today...

Amazon Athena

- According to complexity order:
  - Athena
  - Redshift Spectrum
  - EMR
  - Hive
  - Performance TIPS

amazon EMR

amazon REDSHIFT

Big Data Jargon

- SQL | Schema | Database | Table | DDL
- Ad Hoc query / power Query / Concurrency
- PaaS
- External Table
- Metastore (GLUE)
- Compression
- Partition + bucketing
- SerDe (json,regex,parquet etc)
- Data Format Parquet, ORC AVRO,
- Hadoop file system [HDFS]
- Yarn
- Tez engine
- S3

Introduction AWS Athena SQL

AWS Big Data Demystified

Omid Vahdaty, Big Data Ninja

Amazon Athena

Athena Demo

- Features
- Console: quick Demo - convert to columnar
- Bugs (annoying compiler errors)

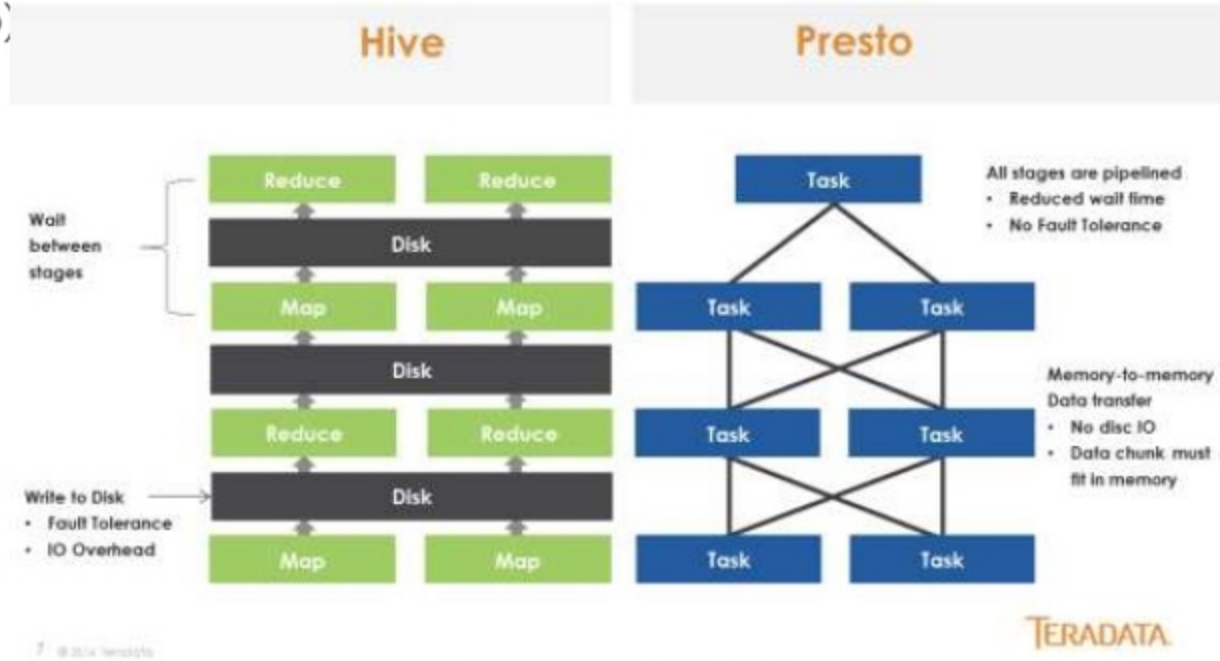
Athena & Hive Convert to Colunar example

https://amazon-aws-big-data-demystified.ninja/2018/05/15/converting-tpch-data-from-row-based-to-columnar-via-hive-or-sparksql-and-run-ad-hoc-queries-via-athena-on-columnar-data/

Behind the scenes

Amazon Athena

- Uses presto for queries
  - Runs in memory
  - (google documentation presto)
- Uses hive for
  - ddl function,
  - complex data types
  - Save temp results to disk
- Relies heavily on parquet
  - Compression
  - Meta data for aggregations



Concurrency

Amazon Athena

- **5 concurrent** queries per AWS account
- Can be increased by **support ticket.**

Billing

Amazon Athena

- **Canceled queries**, are not billed, even if they scan data for an hour!
- **Billing is on compressed data**, not uncompressed, good for end user.
- **5$ per TB**;

Connection

Amazon Athena

- **Web GUI**
- **JDBC**, but has wrapper to other languages.
- http://docs.aws.amazon.com/athena/latest/ug/connect-with-jdbc.html

- Quicksight, SQL workbench etc.

Serde

Amazon Athena

- **<u>Serde are pre installed</u>**
- All formats are supported in compression
  - Parquet - snappy is default, you can change it (decompress is fast)
  - Orc - zlib compression
  - Apache web server - server logs - **RegexSerDe**

Parquet vs Text

- Parquet
  - Colunar
  - Schema segregation into footer
- Text gzip = not colunar. But compressed.

Converting to columnar

- Hive
- spark/ SparkSQL

Partition

Amazon Athena

- Why?
  - Reduces costs
  - Increase performance
- How?
  - Format: Dt=2018-07-23
  - More format available:
    - /2018/07/23
    - And more

TIP: Ignore quotes in csv

Amazon Athena

```
CREATE EXTERNAL TABLE IF NOT EXISTS  walla_mail_mime_inventory (

 bucket string, Key string, VersionId string, IsLatest string, IsDeleteMarker string, Size bigint,
LastModifiedDate string, ETag string, StorageClass string, IsMultipartUploaded string, ReplicationStatus string,
EncryptionStatus string

)ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

   "separatorChar" = ",",

   "quoteChar"     = "\""

)

LOCATION 's3://bucket/';
```

Athena - Summary

- **Use Athena when u get started**
  - Ad hoc query
  - Cheap
  - Forces you to work external all the way.
- **If you fully understand how to work with Athena → you understand big data@aws**
  - It will be very much the same in hive
  - It will be very much the same in sparkSQL

Introduction to AWS Redshift Spectrum

AWS Big Data Demystified

Omid Vahdaty, Big Data Ninja

Spectrum Demo

amazon
REDSHIFT

Console - Reshift Features

Benchmark overview

Console: Demo on large data set

- Use the table created on athena
- Run a query

Working Spectrum example

amazon
REDSHIFT

https://amazon-aws-big-data-demystified.ninja/2018/05/15/want-to-select-data-on-redshift-spectrum-which-was-created-at-athena/

External Table/Schama

amazon
REDSHIFT

- **External Schema**
- http://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_EXTERNAL_SCHEMA.html
- **External table:** http://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_EXTERNAL_TABLE.html
- **Getting started:**
- http://docs.aws.amazon.com/redshift/latest/dg/c-getting-started-using-spectrum-create-role.html
- http://docs.aws.amazon.com/redshift/latest/dg/c-getting-started-using-spectrum-create-external-table.html

Getting started

- **Create schema, Make sure spectrum is available in your region**
  - create external schema spectrum
  - from data catalog
  - database 'spectrumdb'
  - iam_role 'arn:aws:iam::506754145427:role/mySpectrumRole'
  - create external database if not exists;
- **Supported data type**: http://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_EXTERNAL_TABLE.html
- **Bucket and cluster must be in same region**

partitions?

amazon
REDSHIFT

- **Manually add each partitions?** http://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_EXTERNAL_TABLE.html
- **Tip - use hive and Athena msck repair cmd.**

AWS Redshift Spectrum Tips

- **more cluster nodes = more spectrum slices = more performance**
- **smaller nodes = more concurrency**
- Be Sure to understand **local vs External**
- Make sure you understand the data use case
- Redshift Spectrum **doesn't support nested data types,** such as STRUCT, ARRAY, and MAP. –> need to customize solution for this via HIVE.
- data type conversions: string → varchar, double → double precision → done.
- **DO**
  - **GROUP BY clauses**
  - **Comparison conditions and pattern-matching conditions, such as LIKE.**
  - **Aggregate functions, such as COUNT, SUM, AVG, MIN, and MAX.**
  - **String functions.**
  - **to bring back a huge amount of data from S3 into Amazon Redshift to transform and process.**
- **DO NOT: DISTINCT and ORDER BY.**
  - **doesn't support DATE as a regular data type or the DATE transform function.**
  - Be careful about putting large tables that frequently join together in Amazon S3
  - **CTAS is not supported** from External table to External Table. i.e you **can not write to external table - only read.**

Redshift Spectrum Summary

- Spectrum →
  - requires redshift cluster
  - External Table **READ ONLY! (no write)**
- Work with spectrum →
  - if you have a huge hd hoc query (aggregations)
  - If want to remove some data from redshift data to s3, and later on analize it.

Introduction to EMR

AWS Big Data Demystified

Omid Vahdaty, Big Data ninja

amazon
EMR

EMR recap

amazon
EMR

- Hadoop Architecture
  - Master
  - Core
  - Task
  - HDFS
  - Yarn (container)
  - Engine: MR, TEZ
- Scale out/scale up
- Hadoop anti pattern - Join.
- AWS GLUE -
  - shared meta store…
  - And more, but not the topic for today.

EMR DEMO

amazon
EMR

- Console - how to create custom cluster
  - Show all tech options
  - GLUE (prestor, spark, hive)
  - Cofig
    - Maximize resource allocation
    - Dynamic resource allocation
    - Config Path
  - Bootstrap / step
  - Uniform instance/ fleet instances
  - Custom AMI
  - Roles
  - Show security
  - Cli to create cluster

EMR

amazon
EMR

- Tips on creating cheap cluster + performances
  - Auto scaling - based on
    - yarn available memory
    - Container Pending Ratio
  - Spots - bidding strategy
  - New instance group
  - Tasks instance with auto scaling!
  - Same size task as data node

EMR summary

- Use custom cluster
  - Get to know: maximize resource allocation
  - Experiment with all the open source options (hue,zeppelin,oozie, ganglia)
- User Glue to share meta store
- Use task nodes (even without autoscaling, u can kill it with no impact)
- When you are ready
  - Spot instances
  - Auto scaling

Introduction to Hive

AWS Big Data Demystified

Omid Vahdaty, Big data ninja

Hive Agenda

- Console
  - Hive over Hue
  - Hive over CLI
  - Hive over JDBC
- Create external table location S3 text
- Data types
- Serde
- Create external table location S3 parquet
- Json
- External table
- Convert to columnar with paritions - aws example
- Insert overwrite + dynamic partition

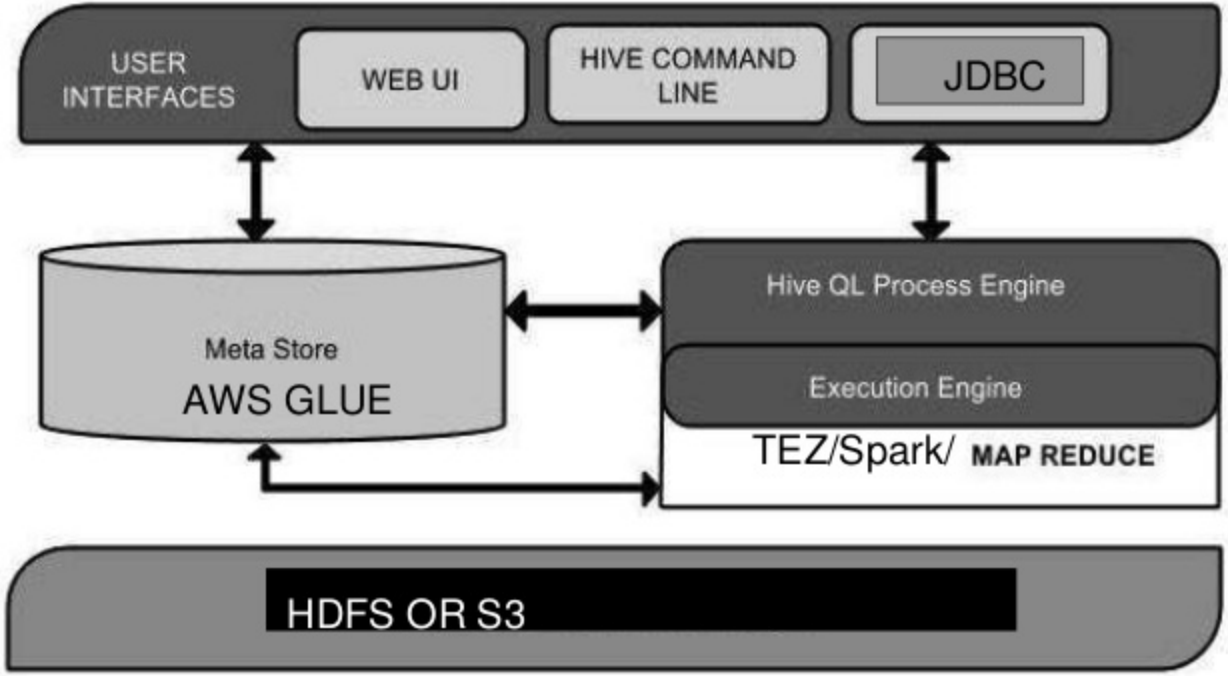
Hive is not...

- Not A design for OnLine Transaction Processing (OLTP)
- Not A language for real-time queries and row-level updates

Hive is...

- It stores schema in a database and processed data into **HDFS**.
- It is designed for **OLAP**.
- It provides SQL type language for querying called **HiveQL** or HQL.
- Configuring **Metastore** means specifying to Hive where the database is stored

Hive Architecture



Data Types

- **Column Types**
  a. int/big int
  b. Strings: char/varchar
  c. Timestamp, dates
  d. Decimals
  e. Union : a set of of several data types
- **Literals**
  a. Floating point, decimal point, null
- **Complex Types**
  a. Arrays, struct, maps!

Supported file formats

- **TEXTFILE** (**CSV**, **JSON**)
- **SEQUENCEFILE** (Sequence files act as a container to store the small files.)
  - Uncompressed key/value records.
  - Record compressed key/value records – only 'values' are compressed here
  - Block compressed key/value records – both keys and values are collected in 'blocks' separately and compressed. The size of the 'block' is configurable.
- **ORC** (recommend for hive, local tables, acid transactions such as delete/update)
- **Parquert**(recommended for spark and External Table)

SerDe

- SerDe is short for **Serializer/Deserializer.** Hive uses the SerDe interface for IO. The interface handles both serialization and deserialization and also interpreting the results of serialization as individual fields for processing.
- A SerDe allows Hive to read in data from a table, and write it back out to HDFS in any custom format. Anyone can write their own SerDe for their own data formats.
- Supported:
- Avro (Hive 0.9.1 and later), http://avro4s-ui.landoop.com
- ORC (Hive 0.11 and later)
- RegEx
- Thrift
- Parquet (Hive 0.13 and later)
- CSV (Hive 0.14 and later)
- JsonSerDe (Hive 0.12 and later in hcatalog-core)
- For Hive releases prior to 0.12, Amazon provides a JSON SerDe available at s3://elasticmapreduce/samples/hive-ads/libs/jsonserde.jar.

File format summary

- CSV/ json ---> text
- Smaller than block size → sequence file
- Analytics: ORC/Parquet columnar based
- Avro → Row based. Used for intensive write use cases

Create table as parquet - local table

```
CREATE TABLE parquet_test (
 id int,
 str string,
 mp MAP<STRING,STRING>,
 lst ARRAY<STRING>,
 struct STRUCT<A:STRING,B:STRING>)
PARTITIONED BY (part string)
STORED AS PARQUET;
```

External Table

Hive tables can be created as EXTERNAL or INTERNAL. This is a choice that affects how data is loaded, controlled, and managed.

Use **EXTERNAL** tables when:

1. The data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files.
2. Data needs to remain in the underlying location even after a **DROP TABLE**.

Convert to Columnar

http://docs.aws.amazon.com/athena/latest/ug/convert-to-columnar.html

- Write to S3 bucket - if you make a **typo** in the bucket name, the data will be written. Just not in S3… :)
- **Create External** table on **source** data from **s3 bucket;**
- You need to manage the partitions via **msck**, identify partitions that were manually added to the distributed file system
- **create target table on s3 as parquet**
- **insert** data from **source to destination**.
- **Query data on s3, as parquet , directly from Hive.**
- **Think files** ==> not one file at a time.

Json SerDe Example

https://github.com/rcongiu/Hive-JSON-Serde

```
CREATE TABLE json_test1 (
    one boolean,
    three array<string>,
    two double,
    four string )
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS TEXTFILE;
```

https://github.com/quux00/hive-json-schema

How to add a serde to EMR hive?

```
ADD JAR /home/hadoop/json-serde-1.3.8-SNAPSHOT-jar-with-dependencies.jar
```

http://stackoverflow.com/questions/26644351/cannot-validate-serde-org-openx-data-jsonserde-jsonserde

Hive Schema From Json

https://github.com/quux00/hive-json-schema

How to add a serde to EMR hive?

ADD JAR /home/hadoop/json-serde-1.3.8-SNAPSHOT-jar-with-dependencies.jar

How get schema from json file:

```
java -jar target/json-hive-schema-1.0-jar-with-dependencies.jar file.json my_table_name
```

http://stackoverflow.com/questions/26644351/cannot-validate-serde-org-openx-data-jsonserde-jsonserde

Example (schema from json)

https://amazon-aws-big-data-demystified.ninja/2018/05/17/getting-a-sql-schema-from-json/

Nested jsons

https://aws.amazon.com/blogs/big-data/create-tables-in-amazon-athena-from-nested-json-and-mappings-using-jsonserde/

- Consider AVRO (works better on nested columns than parquet)

Lateral View (explode) && inline

- https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView
- http://stackoverflow.com/questions/25088488/use-inlinearraystruct-struct-in-hive
- http://stackoverflow.com/questions/11373543/explode-the-array-of-struct-in-hive
- Laterl view of multiple arrays: http://stackoverflow.com/questions/20667473/hive-explode-lateral-view-multiple-arrays

Dynamic partition

https://cwiki.apache.org/confluence/display/Hive/DynamicPartitions

set hive.exec.dynamic.partition.mode=nonstrict

```
INSERT OVERWRITE TABLE t  partition (dt)

SELECT

source_ip ,

  dns,

  dt

FROM bbl_dns WHERE dt=current_date order by source_ip;
```

Why Use ORC ?

1. ORC has performance optimizations
2. **ORC has transaction : delete/update**
3. ORC has bucketing (index...)
4. ORC suppose to be faster than Parquet
5. Parquet might be better if you have highly nested data, because it stores its elements as a tree like **Google Dremel** does (See here).
6. Apache ORC might be better if your file-structure is flattened.
7. when Hive queries ORC tables **GC is called about 10 times less frequently**. Might be nothing for many projects, but might be crucial for others.

Why Use Parquet

<u>WHY NOT ORC: Couple of considerations for Parquet over ORC in Spark are:</u>

1) Easily creation of Dataframes in spark. No need to specify schemas.

2) Worked on highly nested data.

3) works well with spark.Spark and Parquet is good combination

4. Also, ORC compression is sometimes a bit random, while Parquet compression is much more consistent. It looks like when ORC table has many number columns - it doesn't compress as well. It affects both zlib and snappy compression

Why Use ORC/Parquet

Confusing parts:

https://stackoverflow.com/questions/32373460/parquet-vs-orc-vs-orc-with-snappy

- **Hive** has a vectorized ORC reader but no vectorized parquet reader.
- **Spark** has a vectorized parquet reader and no vectorized ORC reader.
- **Spark performs best with parquet, hive performs best with ORC.**

Hive Summary

- Understand the following concepts
  - Colunar
  - External table
  - Json parsing
  - ACID tables
  - ORC/Parquet
  - Lateral View + explode
- **Bottom line demystified**
  - **Work with external tables all the way!**
  - Use **parquet** (for future work with spark…)
  - ACID → use ORC + Hive
  - Use **Serde** to parse raw data
  - Use **dynamic partitions** when possible (carefully)
  - Use hive to convert data to what u need - **insert OVERWRITE**

Big Data SQL performance Tips @ AWS | Lesson Learned

AWS Big Data Demystified

Omid Vahdaty, Big Data Ninja

Generally speaking be familiarized with

1. When to normalize/denormalize
2. Columnar VS Row based
3. Storage types: AVRO/Parquet/ORC
4. Compression
5. Complex data types
6. When to partition? How much?
7. Processing Int is faster than string by an order of X10
8. What is the faster DB in the world? [**Hint: what is your use case?**]
9. Network latency from Client to Server?
10. Encryption at rest / in motion = performance impact?

Tips for external / local tables when to use what/which

Use local table when

1. when using only analytics DB such as redshift , and you don't need a data lake
2. when data is small and temporary insert takes time
3. When performance is everything (be prepared to pay 5 to 10 times more on external)
4. when you need to insert temporary results of your query - and there is no option to write to an external table (hive supports write to an external table, but athena doest)

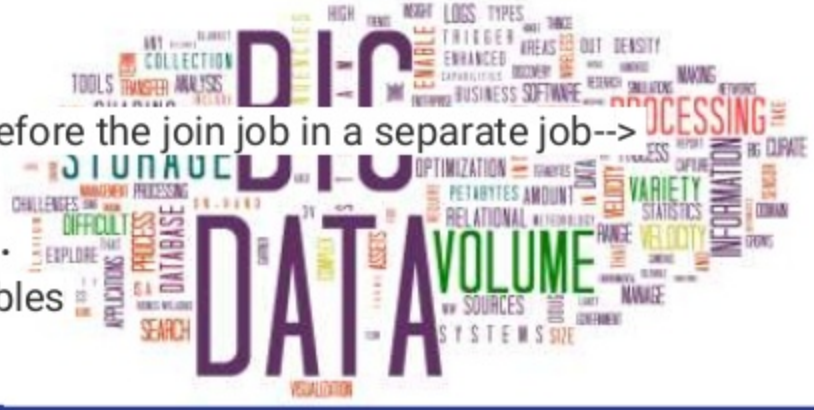
Use External table when

1. cost is an issue - use transient clusters
2. Your data is already on S3.
3. when you need several DB's for several use cases, and you want to avoid insert/ETL's
4. when you want to decouple the compute power and storage power :
5. i.e athena & spectrum - infinite compute , infinite storage. pay on what you use only.

	Redshift	Redshift spectrum	Hive	Athena
Cost	High	low	medium	low
Performance	top 10 in the world.	fast enough...	slow...	fast enough...
Syntax	Postgres	postgres	Hive	Presto
Data types advantages	no arrays	no arrays	complex data types	complex data types
Storage type	Colunar	Colunar	Columnar, and Row	Columnar, and Row
Usecase	Joins , traditional DBMS, analytics: Joins, AGG , order by	Aggregations ONLY,	transformation , advanced parsing, transient clusters.	ad hoc querying, not for big data.
Anti pattern	temporary cluster	Joins	Joins, quick and dirty, simplicity	Joins / Big Data / Inserts

Performance Tips for modeling

1. choose **correct partition** . [dt? win time?]
2. big data anti pattern - usage of join... use **flat table** whenever possible. easier to calculate.
3. **Static Raw data** (one time job as data enters) = Precalculate what you need on a daily basis = storage is cheap...
   a. lookup tables - convert to int when possible, or even boolean if exist. dont use "like"
   b. datepart of wintime = can you pre calculate into fact table?
   c. minimize likes
   d. string to int/boolean/bit when possible.
   e. case = can you pre calculate into fact table?
   f. coalesce = can you pre calculate into fact table?
   g. calculate group by of indexes (bluekai/gaid) values before the join job in a separate job--> reduce running time of join.
4. **Dynamic data** (recurring daily job) **compute is expensive...**
   a. filter data by the same time interval across all fact tables
   b. filter rows not needed across all fact tables



If you must join...

1. notice the **order of the tables** - join - from small to big.
2. **filter** as much as possible
3. use **only columns** you must.
4. Use **explain** to understand the query you are writing
5. use explain to minimize raws (**small table X small table** = maybe equals big table)
6. copy **small tables to all data nodes** (redshift/hive)
7. use **hints** if possible.
8. Divide the job to **smaller atomic steps**

Tips to avoid join

1. use **flat tables with many columns** - storage is cheap
2. use **complex data types** such as arrays, and nested arrays.

Hive tuning tips

1. Avoid order by if possible…
2. Minimize reduces.
3. Config suggested:
   a. set hive.exec.parallel=true;
   b. set hive.exec.parallel.thread.number=24;
   c. set hive.tez.container.size=4092; (check this one carefully)
   d. set hive.exec.parallel=true;
   e. set hive.support.concurrency=true;
   f. set hive.exec.compress.output=true;
      set hive.exec.compress.intermediate=true;
      set mapred.compress.map.output=true;
      set hive.execution.engine=mr;

Redshift Tips

- https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-techniques-for-amazon-redshift/
- **Distribution style**
- **Sort key** column identified, which acts like an index in other databases,
- **COMPOUND sort keys**
- **Caching - disk VS ram**

Performance Summary

- Partitions…
- External table VS Local table
- Flat tables + complex data types VS Join
- Compression
- Columnar → Parquet

Lecture summary - starting with big data?

- Start with athena
- Have already redshift? Consider spectrum
- Use EMR Hive to transform data from any structured semistructured data to parquet
- Fully nested? consider AVRO

Complex Q&A from the audience - post lecture notes

- When to use redshift? And when to use EMR (spark SQL, hive, presto)
  - https://amazon-aws-big-data-demystified.ninja/2018/06/03/when-should-we-emr-and-when-to-use-redshift/
- Cost reduction on Athena:
  - https://amazon-aws-big-data-demystified.ninja/2018/06/03/cost-reduction-on-athena/

Stay in touch...

- Omid Vahdaty
- +972-54-2384178
- https://amazon-aws-big-data-demystified.ninja/
- Join our meetup, FB group and youtube channel
  - https://www.meetup.com/AWS-Big-Data-Demystified/
  - https://www.facebook.com/groups/amazon.aws.big.data.demystified/
  - https://www.youtube.com/channel/UCzeGqhZIWU-hIDczWa8GtgQ?view_as=subscriber