An overview of Amazon Athena

and how it performs against Amazon Redshift

Julien Simon, Principal Technical Evangelist, AWS
julsimon@amazon.fr
@julsimon

Never trust the first image on Google!

"*Amazon Athena is a professional wrestler*" 8-|

On second thought, that's quite relevant!

amazon web services

Amazon Athena is a professional data wrestler!

- New service announced at re:Invent 2016
- Run interactive SQL queries on S3 data
  - No need to load or aggregate data: ‘schema-on-read’
  - S3 data is never modified
  - Cross-region buckets are supported
- No infrastructure to create, manage or scale
- Availability: us-east-1, us-west-2
- Pricing: $5 per Terabyte scanned
  - Scanned data rounded off to the nearest 10MB
  - Stored data: normal S3 pricing applies

AWS re:Invent 2016: Introducing Athena (BDA303) https://www.youtube.com/watch?v=DxAuj_Ky5aw

Athena queries

- Service based on Presto (already available in Amazon EMR)
- Table creation: Apache Hive DDL
  - CREATE EXTERNAL_TABLE only
  - CREATE TABLE AS SELECT is not supported
- ANSI SQL operators and functions: what Presto supports
- Unsupported operations
  - User-defined functions (UDF or UDAFs)
  - Stored procedures
  - Any transaction found in Hive or Presto

https://prestodb.io
http://docs.aws.amazon.com/athena/latest/ug/known-limitations.html

Data formats supported by Athena

- Unstructured
  - Apache logs, with customizable regular expression
- Semi-structured
  - Comma-separated values (CSV)
  - Tab-separated values (TSV)
  - Text File with custom delimiters
  - JSON
- Structured
  - Apache Parquet
  - Apache ORC (Optimized Row Columnar)
- Compression formats: Snappy, Zlib, GZIP (no LZO)
  - Less I/O → better performance and cost optimization

https://parquet.apache.org/

https://orc.apache.o

Data partitioning

- Partitioning reduces the amount of scanned data
  - Better performance
  - Cost optimization
- Data may be already partitioned in S3
  - CREATE EXTERNAL TABLE *table_name*(…) PARTITIONED BY (...)
  - MSCK REPAIR TABLE *table_name*
- Data can also be partitioned at table creation time
  - CREATE EXTERNAL TABLE *table_name*(…)
  - ALTER TABLE table_name ADD PARTITION …

Running queries on Athena

- AWS Console (quite cool, actually)
  - Wizard for schema definition and table creation
  - Saved queries
  - Query history
  - Multiple queries in parallel
- JDBC driver
  - SQL Workbench/J
  - Java application

http://docs.aws.amazon.com/athena/latest/ug/connect-with-jdbc.html

Setting up SQL Workbench/J

- Download & install JDBC driver
- Create a new connexion
  - Driver: *com.amazonaws.athena.jdbc.AthenaDriver*
  - URL: *jdbc:awsathena://athena.us-east-1.amazonaws.com:443/*
  - Username: your AWS Access Key
  - Password: your AWS Secret Key
  - Add an extended property: *s3_staging_dir*
    - S3 bucket for output data, e.g. *s3://jsimon-athena-output/*
    - Make sure this S3 bucket is in the same region as Athena
- You're all set!

Athena vs Redshift

- Redshift
  - Start 4-node cluster: dc1.large, 160GB SSD, low I/O, $0.25/hr
  - Start 4-node cluster: dc1.8xlarge, 2.56TB SSD, v.high I/O, $4.80/hr
  - Create table
  - Load data from S3 (COPY operation)
  - Run some queries
- Athena
  - Create table
  - Run the same queries

Athena vs Redshift: start your engines!

- Athena
  - Initialization : < 5s (table creation)
  - Cost: $0.0025 for a full scan (12GB + a few thousand S3 requests)
  - Unlimited storage in S3
- Redshift (dc1.large)
  - Initialization: 6mn (create cluster) + 38mn (data load)
  - $1/hr ($0.36 with 3-yr, 100% upfront RIs)
  - Maximum storage: 640GB (about 2TB with compression)
- Redshift (dc1.8xlarge)
  - Initialization: 6mn (create cluster) + 4mn (data load)
  - $19.20/hr ($6 with 3-yr, 100% upfront RIs)
  - Maximum storage: 10TB (about 30TB with compression)

Athena vs Redshift: data set

Caveat: this isn't a huge data set and it doesn't have any joins

- 1 table
- 1 billion lines of "e-commerce sales" (43GB)
- CSV format, 10 columns
- 1000 files in S3, compressed to 12GB (bzip2)

```
Lastname, Firstname,Gender,State,Age,DayOfYear,Hour,Minutes,Items,Basket
YESTRAMSKI,KEELEY,F,Missouri,36,35,12,21,2,167
MAYOU,SCOTTIE,M,Arkansas,85,258,11,21,9,106
PFARR,SIDNEY,M,Indiana,59,146,22,21,3,163
RENZONI,ALLEN,M,Montana,31,227,13,49,10,106
CUMMINS,NICKY,M,Tennessee,50,362,1,33,1,115
THIMMESCH,BRIAN,M,Washington,29,302,20,41,2,95
```

Athena vs Redshift: table creation

```
CREATE EXTERNAL TABLE athenatest.sales (
  lastname STRING,
  firstname STRING,
  gender STRING,
  state STRING,
  age INT,
  day INT,
  hour INT,
  minutes INT,
  items INT,
  basket INT
)
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = ',',
  'field.delim' = ','
) LOCATION 's3://jsimon-redshift-demo-us/data/'
```

```
CREATE TABLE sales(
lastname VARCHAR(32) NOT NULL,
firstname VARCHAR(32) NOT NULL,
gender VARCHAR(1) NOT NULL,
state VARCHAR(32) NOT NULL,
age INT NOT NULL,
day INT NOT NULL,
hour INT NOT NULL,
minutes INT NOT NULL,
items INT NOT NULL,
basket INT NOT NULL)
DISTKEY(state)
COMPOUND SORTKEY (lastname,firstname);

COPY sales FROM 's3://jsimon-redshift-demo-us/data/'
REGION 'us-east-1' CREDENTIALS … DELIMITER ',' bzip2
COMPUPDATE ON;
```

Athena vs Redshift: SQL queries

Identical queries on both systems

```
; Q1: count sales (1 full scan)
SELECT count(*) FROM sales
; Q2: average basket per gender (1 full scan)
SELECT gender, avg(basket) FROM sales GROUP BY gender;
; Q3: 5-day intervals when women spend most
SELECT floor(day/5.00)*5, sum(basket) AS spend FROM sales WHERE gender='F' GROUP
BY floor(day/5.00)*5 ORDER BY spend DESC LIMIT 10;
; Q4: top 10 states where women spend most in December (1 full scan)
SELECT state, sum(basket) AS spend FROM sales WHERE gender='F' AND day>=334 GROUP
BY state ORDER BY spend DESC LIMIT 10;
; Q5: list the top 10000 female customers in the top 10 states (2 full scans)
SELECT lastname, firstname, spend FROM (
     SELECT lastname, firstname, sum(basket) AS spend FROM sales WHERE gender='F'
     AND state IN(
          SELECT state FROM sales WHERE day>=334 GROUP BY state
          ORDER BY sum(basket) DESC LIMIT 10
     ) AND day >=334 GROUP BY lastname,firstname
) WHERE spend >=500 ORDER BY spend DESC LIMIT 10000;
```

Athena vs Redshift: SQL queries

YMMV, standard disclaimer applies ☺

; Q1: count sales

Athena: 15-17s, Redshift: 2-3s, Redshift 8xl: <1s

; Q2: average basket per gender

Athena: 20-22s, Redshift: 15-17s, Redshift 8xl: 4-5s

; Q3: 5-day intervals when women spend most

Athena: 20-22s, Redshift: 20-22s, Redshift 8xl: 4-5s

; Q4: top 10 states where women spend most in December

Athena: 22-25s, Redshift: 10-12s, Redshift 8xl: 2-3s

(courtesy of the '*state*' distribution key)

; Q5: list the top 10000 female customers in the top 10 states

Athena: 38-40s, Redshift: 34-36s, Redshift 8xl: 7-9s

So?

- For this data set, Athena query performance is in the same ballpark as a vanilla Redshift cluster of 4 dc1.large nodes
- Athena saves you the long init time (cluster creation + data load)
- And probably a lot of money as well
  - Several orders of magnitude cheaper if you run a single query!
  - Similar to Lambda vs EC2

- So… Athena looks great IMHO ☺
- I can see it being used for much more than ad-hoc queries
- Redshift still rules when you need the best performance possible

EMR, Redshift or Athena?

- EMR
  - Scale-out data crunching
  - Custom code running complex transformations on unstructured data
  - Rich Apache Hadoop ecosystem, at the cost of complexity
- Redshift
  - Petabyte-scale enterprise data warehouse
  - ETL, complex SQL queries and joins on long-lived, structured data
  - Many techniques for performance optimization
- Athena
  - Answering questions in minutes, with zero infrastructure plumbing
  - Ad-hoc SQL queries, with probably a few or no joins
  - Emphasis on simplicity, not on raw performance

Athena in a nutshell

- Run ad-hoc SQL queries on S3 data in minutes
- No infrastructure
- Multiple input formats supported
- Slower than Redshift on 8xl nodes, but pretty fast!
- A simpler, very cost-efficient alternative to EMR and Redshift for ad-hoc analysis

Thank you!

Julien Simon, Principal Technical Evangelist, AWS
julsimon@amazon.fr
@julsimon

amazon web services