Cloud Platform Comparison

AWS Athena, Google BigQuery, Snowflake, Databricks

STANFORD GRADUATE SCHOOL OF BUSINESS

Outline

- Recall
- Updated Benchmark Results
- Introduction to Databricks
- Databricks Demo
- Query Comparison
- Follow-up Features

Recall

- ~ 2.5 TB GDELT dataset located in Amazon S3
- 8 queries involving table joins, recasting data types, aggregating, string matching, Common Table Expressions
- Snowflake outperformed AWS Athena and Google BigQuery in terms of query speed and cost
- Snowflake data is partitioned and columnarly-compressed by default

Updated Benchmark Results

Query Speed

Databricks (Spark.SQL) > Snowflake > Databricks (Native SQL) > BQ Native > Athena Column > BQ External > Athena Row

Data Scanned

Snowflake < Athena Column < BQ Native < BQ External = Athena Row

Cost

Databricks (Spark.SQL) < Snowflake < Databricks (Native SQL) < Athena Column < BQ Native < BQ External = Athena Row

Introduction to Databricks

Overview

- Partnerships with AWS and Microsoft Azure
- An analytics platform that runs Spark on AWS machines
- Can choose machine type and use spot pricing
- Two infrastructure types: cluster or serverless pool
- Two modes: interactive (notebooks, for data analytics) and non-interactive (scripts and jobs for data engineering)

Introduction to Databricks

Standard Cluster	**Serverless Pool**
Autoscaling available	Autoscaling available
Needs manual configuration	Auto-configuration for performance, fault isolation and high concurrency
Can choose different driver/worker instance type	Same driver/worker instance types
Supports Python, R, SQL, Scala	Supports Python, SQL
Auto-terminate available	Auto-terminate not available

Introduction to Databricks

Overview

- To allow auto-scaling: provide the min and max number of instances and cluster/pool scales based on workload
- Cluster supports PySpark, SparkSQL, native SQL, native Python, Scala, R; some R/Python packages are pre-installed, others can be installed by user
- Serverless pool supports PySpark, SparkSQL, native SQL and native Python only
- Native Python ML models will not be distributed. SparkML should be used instead for better performance.

Remarks on Benchmarking Results

- SparkSQL runs faster than native SQL, but the latter outputs query results in the notebook and lets you visualize results based on sample without running an additional Spark job.

- Some type of caching might occur: same query gets faster after successive runs in the same notebook; not true if you change notebooks

- The cluster/pool scales based on workload: a max of 4 workers does not mean all 4 are leveraged for each query. (Contrast this to Snowflake, where a 4-node cluster uses all 4 nodes.)

Introduction to Databricks

Data In

- Reads from S3: no data out charges, fast read time
- Data load: load from S3 > create Dataframe (exists only in the notebook) > write as table (persistent)
- Error-handling: can store bad files and records and reasons from the exception logs to a specified filepath for later investigation

Introduction to Databricks

Data Out

- Saving query output:
  - Native SQL: use create table as to save output to database in Parquet format
- Use PySpark to write output to database or another destination (S3/local) in any format (Parquet/csv)
- Database is stored in the Databricks File System (DBFS) tied to your account. DBFS is independent of your notebook and cluster, hence data persists across different notebook/cluster instatiations.
- Can interact with DBFS through the Databricks CLI.

Introduction to Databricks

Pricing

- Per-second billing
- Pay for underlying machinery + Databricks cost
- AWS costs billed directly to AWS account
- Databricks on Azure is currently on preview for 50% of AWS cost.

Introduction to Databricks

Pricing

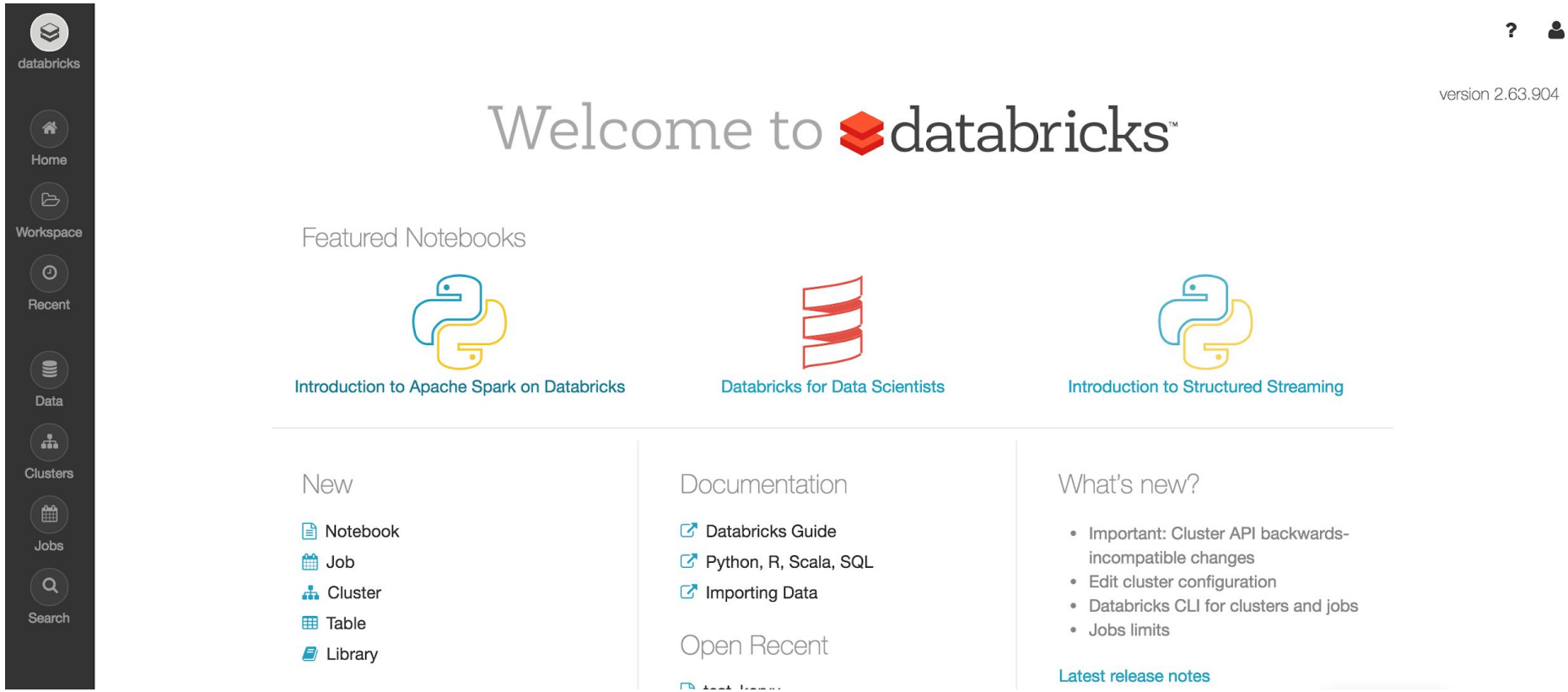
	Non-Interactive	Interactive
Databricks AWS	$0.20 per hour per Databricks Unit (DBU) + AWS cost	$0.40 per hour per Databricks Unit (DBU) + AWS cost
Operational Security Package	+ $0.15 per Databricks Unit (DBU)	

Introduction to Databricks

Takeaways

- Cheaper and faster than Snowflake
- Not a data warehousing tool
- Everything done through scripting
- Key feature is elasticity + auto-scaling
- Notebooks support different languages
- Can execute entire analytics workflow (ETL + modelling) in the same notebook

Databricks Demo



Query Comparison

Q1: Select all records where theme includes ‘terror’

	DB Spark SQL	SF	DB Native SQL	BQ Native	Athena Columnar	BQ External	Athena Row
Runtime (min)	0.38	35.22	0.59	59.9	71.98	74.60	82.02
Data Scanned (GB)		924.4	0.12	2380	2370	2380	2380
Cost ($)	0.03	4.70	0.04	11.90	11.85	11.90	11.90

* Options with smallest run time, data scanned and cost highlighted.

Query Comparison

Q2: Select all events where the confidence score >= 100

	DB Spark SQL	SF	DB Native SQL	BQ Native	Athena Columnar	BQ External	Athena Row
Runtime (min)	0.99	1.32	3.10	5.10	7.88	74.60	10.36
Data Scanned (GB)		14.9	0.21	80.6	36.1	158.0	157.8
Cost ($)	0.07	0.18	0.22	0.40	0.18	0.79	0.79

Query Comparison

Q4: All mentions where actor 1, actor 2, or action is located in US and event occurred in 2017; using a ‘where in’ clause

	DB Spark SQL	SF	DB Native SQL	BQ Native	Athena Columnar	BQ External	Athena Row
Runtime (min)	4.68	1.12	5.19	10.05	11.60	12.93	13.50
Data Scanned (GB)		14.3	17.61	97.6	57.5	158.0	157.8
Cost ($)	0.32	0.15	0.35	0.49	0.29	0.79	0.79

Query Comparison

Q8: Number of rows per combination of 27 column & join of three tables (Exhausted Resources Error query)

	DB Spark SQL	SF	DB Native SQL	BQ Native	Athena Columnar	BQ External	Athena Row
Runtime (min)	2.06	3.88	2.09	4.00	Failed	7.05	Failed
Data Scanned (GB)		4.2	2.2	20.3	-	158.0	-
Cost ($)	0.14	0.52	0.14	0.10	-	0.79	-

- See here for more details

Query Comparison

8 Spark SQL queries run on Databricks using different pool sizes

	Query Time (min)		Query Cost ($)	
Query	**4 nodes**	**20 nodes**	**4 nodes**	**20 nodes**
1	0.38	0.44	0.03	0.13
2	0.99	0.59	0.07	0.17
4	4.68	2.06	0.32	0.59
8	2.09	0.84	0.14	0.24

Remarks

- On the 4-node cluster, queries run right after data load jobs took longer than queries run as the first jobs on a “fresh” cluster. This could be related to “depleted” EC2 CPU Credit Balance.
- On the 20-node cluster, a “fresh” cluster yielded slower query times than a “working” one. This could be because previous tasks force the cluster to scale up, and subsequent queries leveraged a larger pool of resources before it could scale down again.

Follow-Up Features

Permissioning

Google BigQuery	Snowflake	Databricks
• User- or role-based • Available for objects and actions • Securable at database level • Table and column-level permissioning proxied via securable views	• User- or role-based • Available for objects and actions • Actions: cluster management, SQL • Objects: cluster, database, table • Securable at table level • Column-level permissioning proxied via securable views	• User-based access control • Available for objects and actions • Actions: cluster management, SQL • Objects: cluster, database, table, notebook • Data securable at table level • Notebook securable at cell level • Column-level permissioning proxied via securable views

Follow-Up Features

Security

Google BigQuery	Snowflake	Databricks
• Data is encrypted • MFA available	• Data is encrypted • MFA available	• Data not encrypted • MFA available

Ability to tag users

Google BigQuery	Snowflake	Databricks
• Audit logs to track down actions and acting user • Can add labels to datasets, tables, and views to trace cos • Can set cost quota for projects and individual user	• Can be achieved by spinning up separate warehouses for each user's compute efforts • This will add cost since pricing is based on warehouse-second	• Notebooks are user-based • Since cluster is linked to AWS, you can track instance costs on AWS dashboard

Follow-Up Features

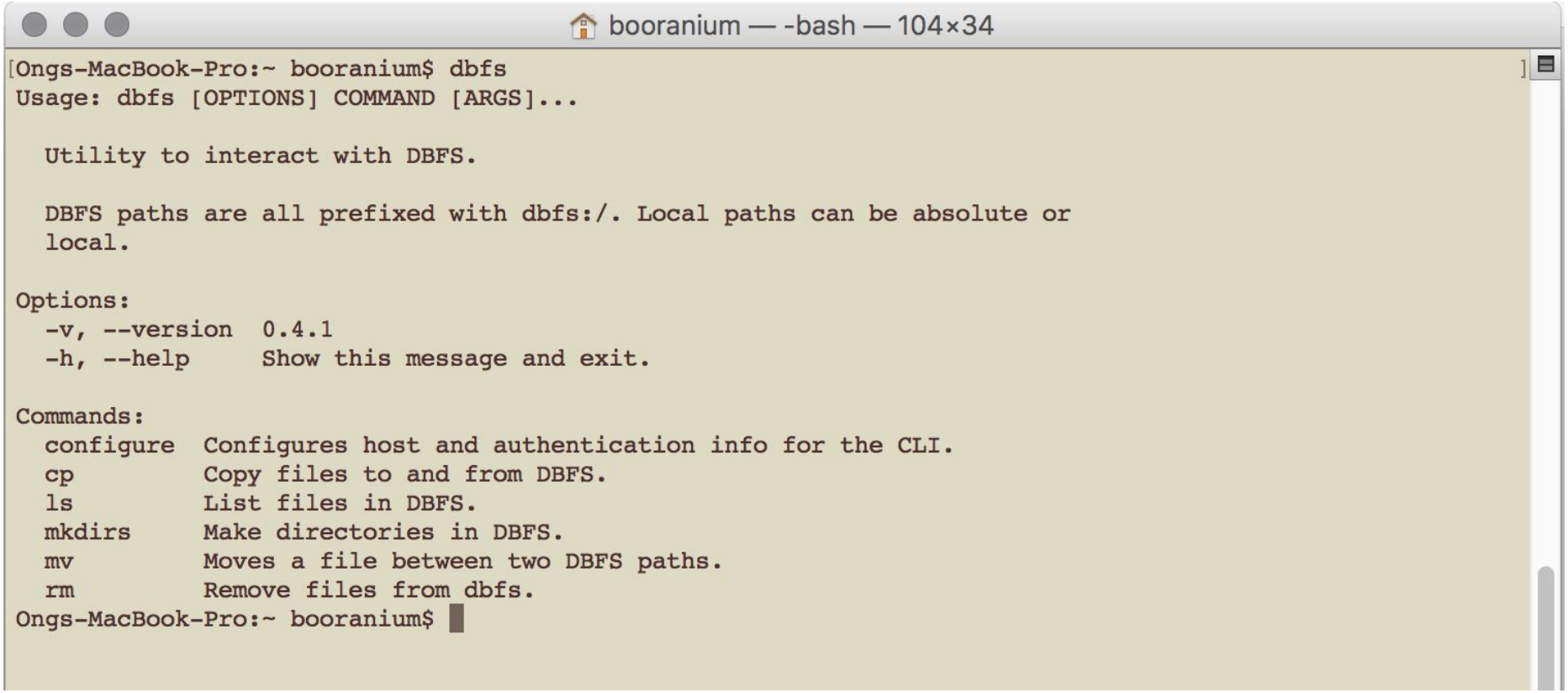
Single Sign On

Google BigQuery	Snowflake	Databricks
• Already in place	• Supported	• Supported

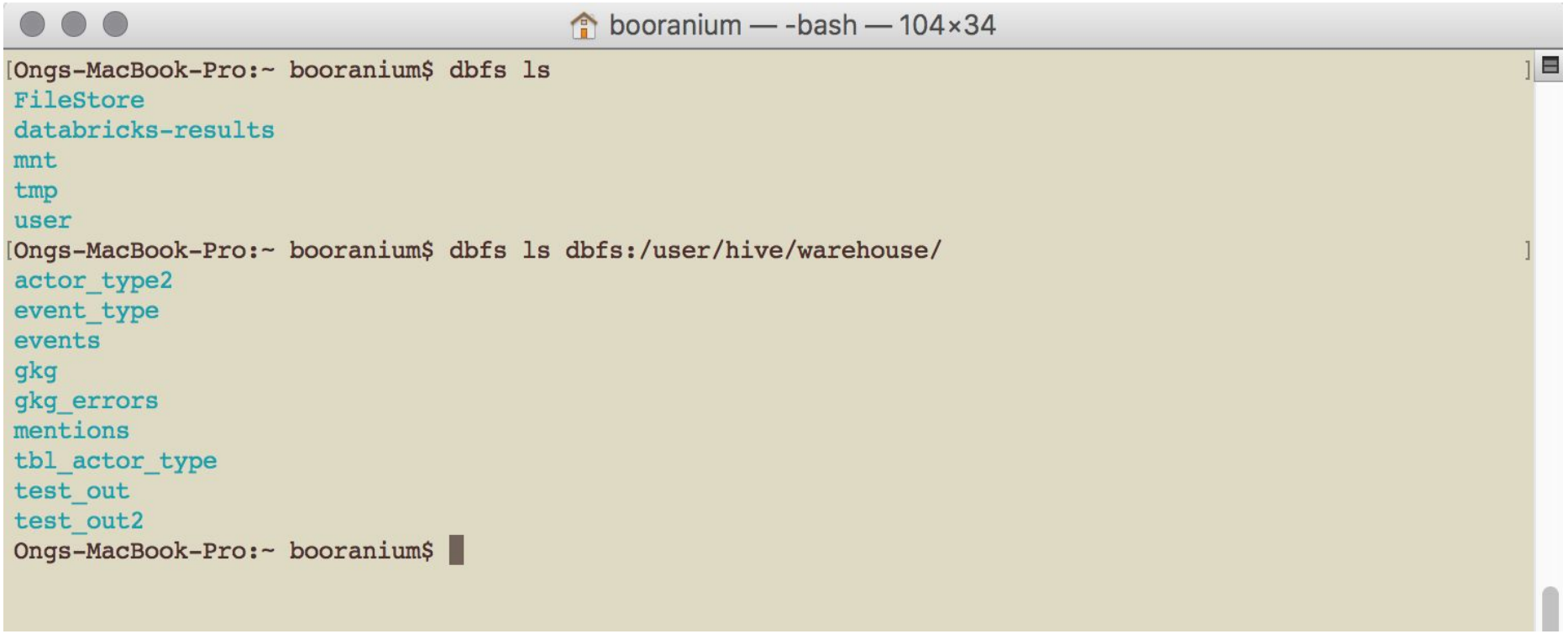
“Easy-Environment” Options

Google BigQuery	Snowflake	Databricks
• Nice web UI with drop menu options • CLI and client libraries available	• Admin can set relevant privileges for warehouse • Option to set warehouse to auto-suspend after certain amount of time and auto-resume when query is run	• Admin can configure cluster and data access settings • Provide pre-configured notebooks • Option to auto-suspend cluster (but no serverless pool) after period of inactivity

DBFS CLI


booranium — -bash — 104×34
Ongs-MacBook-Pro:~ booranium$ dbfs
Usage: dbfs [OPTIONS] COMMAND [ARGS]...
Utility to interact with DBFS.
DBFS paths are all prefixed with dbfs:/. Local paths can be absolute or local.
Options:
-v, --version 0.4.1
-h, --help Show this message and exit.
Commands:
configure Configures host and authentication info for the CLI.
cp Copy files to and from DBFS.
ls List files in DBFS.
mkdirs Make directories in DBFS.
mv Moves a file between two DBFS paths.
rm Remove files from dbfs.
Ongs-MacBook-Pro:~ booranium$

DBFS CLI


booranium — -bash — 104×34
Ongs-MacBook-Pro:~ booranium$ dbfs ls
FileStore
databricks-results
mnt
tmp
user
Ongs-MacBook-Pro:~ booranium$ dbfs ls dbfs:/user/hive/warehouse/
actor_type2
event_type
events
gkg
gkg_errors
mentions
tbl_actor_type
test_out
test_out2
Ongs-MacBook-Pro:~ booranium$

Intro to Amazon Aurora

- An relational DB managed by Amazon RDS (fully managed: updates, security patches, data replication, etc)
- Supports both MySQL (5x faster) and Postgres (3x faster)
- Auto-scaling up to 64 TB
- “Database cluster”: multiple DB instances
- Data stored in replicas across multiple availability zones for increased availability + fault tolerance
- Requires setting up a VPC with subnets in at least two availability zones

Change lives. Change organizations. Change the world.