Compare Databricks to Apache Spark on AWS (Final Report)

Yiming Liu
liuy379@uw.edu

1. Introduction
Apache Spark has experienced substantial growth over the last decade and it has become the essential data processing and AI engine in many companies because of its speed and ease of use. Databricks, founded by the team that originally created Apache Spark, has delivered products built on top of Apache Spark that are more optimized and simpler to use than Apache Spark. For example, the Databricks Runtime is a data processing engine built on highly optimized version of Apache Spark and it provides up to 50x performance gains. Interestingly, in 2014, the year after Databricks was founded, the team contributed to more than 75% of the code added to open source Apache Spark, so it has truly been the core engineering team behind Apache Spark. The goal of this project is to compare performance and ease of use of Databricks and Spark on AWS. Overall, Databricks outperforms AWS Spark in terms of both performance and ease of use. However, if we consider the cost of Databricks, choosing between these two platforms depends on actual scenarios.

2. Evaluated Systems
The systems I work on in the project are Spark on AWS and Databricks. For Databricks, it is a unified data analytics platform that offers features like collaborative notebooks, optimized machine learning environments, and completely managed ML lifecycle. I have mainly focused on the Spark engine Databricks offers (Databricks Runtime) for performance testing and collaborative notebooks for ease of use testing. The Databricks Runtime implements the open Apache Spark with a highly optimized execution engine, which provides significant performance gains compared to standard open source Apache Spark found on cloud platforms. The collaborative notebooks are perfect for quick exploratory data analysis or collaborative data science works.

3. Problem Statement and Method
The main problem statement is that is Databricks faster than Spark on AWS? To evaluate performances, I have tested and recorded the runtime of same data reading, aggregation, join, and machine learning actions on these two platforms. Another problem statement is that Databricks is more user friendly than Spark on cloud platforms as this is one of the missions of Databricks. To evaluate ease of use, I have noted down my experience and thoughts while using Databricks.

4. Results
4.1 Performance Testing
Databricks claims it has 50x performance gains upon the open source Apache Spark, and I have performed some basic data operations and manipulations (data reading, groupyby, join) to test this. The data I use for this part of the testing has size of 3 GB and it is publicly available on Kaggle. The baseline instance setup I used was 2 m5.xlarge workers, and I also tried 4 m5.xlarge workers and 8 m5.xlarge workers in all the test cases to test the effect of speed up.

4.1.1 Data Reading
The first thing I tested was reading data from S3 into Databricks notebook and AWS ipython notebook. The data is stored as csv format with no partition. However, I have also tested Spark's reading speed over different types of files (partitioned csv, partitioned parquet), and I will discuss this in later sections. *Spark.read.csv* is used here to read the data, and parameter *inferSchema=true* is set to make sure the reading action is actually performed. As shown in **Figure 1**, Databricks outperforms AWS Spark in all cases. Specifically, the Databricks' reading time on 8 workers is 60% faster than AWS Spark's.

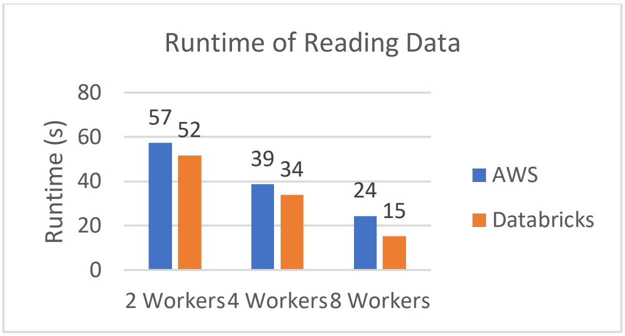


Figure 1

4.1.2 Aggregation and Join

Then I tested how Databricks and AWS Spark perform differently on aggregation and join. The data used in this part are all in RDD dataframe format with a fixed partition size of 5. Simple aggregation on one of the Id in the dataset was performed and *count()* was used in order to force Spark to take actions. **Figure 2** and **Figure 3** are the runtime comparisons of aggregation and join on the two platforms. As it is shown in the figures, Databricks still outperforms AWS Spark in both aggregation and join operations. The speed up by increasing number of workers is linear on both platforms, and quite similar to previous data reading testing, Databricks is 66% faster on joining RDD dataframes on 8 workers than AWS Spark, and we can observe decreasing returns to speed-up as well. Due to potential cost of launching clusters with more than 8 workers and the scope of this project, 8 workers (32 virtual cores in total) are the maximum threshold in this project. However, given the fact that substantially more workers are needed in many actual practices, it is very likely that the performance can be further boosted with more workers. Also, because the implementation of Apache Spark in Databricks Runtime is on top of an optimized execution engine and it is not open sourced, it is quite hard to elaborate on these performance improvements.

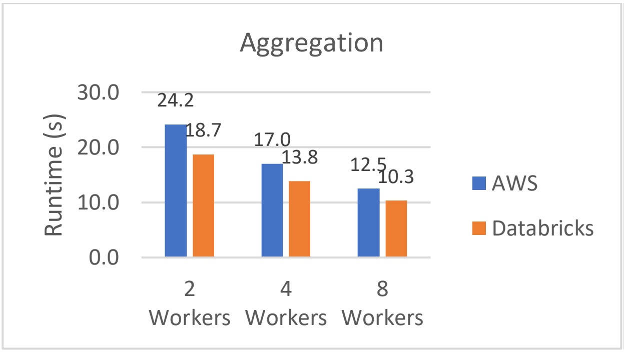


Figure 2

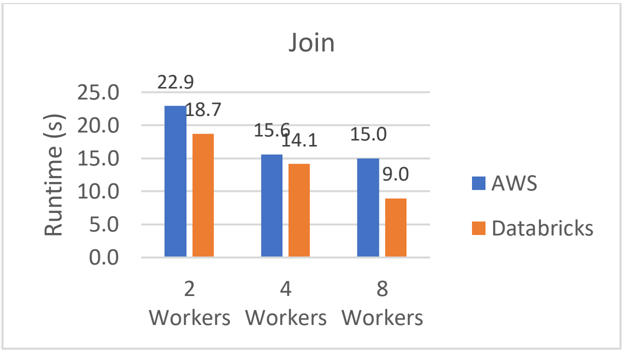


Figure 3

While I was testing the performance of aggregation and join, I found that operations like groupby and join can benefit from setting the appropriate number of partitions. So I took a detour from comparing Databricks and AWS Spark, and tried to perform the above aggregation and join actions with different numbers of partitions. I used *df.repartition(n)* to set number of partitions, and it can also be expressed as *df.repartition(n, column)* where the column is usually the column that we aggregate data on or the joining key. In the dataset I used, it is a *card_id* column so this is what *ID Partition* in the legend refers to. **Figure 4** and **Figure 5** are the results. We see that having more partitions generally result in faster execution. Interestingly, we can see that the runtime is reduced from 14s to 11s when both number of workers and number of partitions increase. This makes sense as papalism is better optimized when more partitions can be processed at the same time. Additionally, I thought aggregation and join can benefit from aggregating and joining on the column that the data is partitioned on, but the results do not indicate that. However, I still think the column partition can make a difference here if the data size is large enough. I will dig more on this in future works.

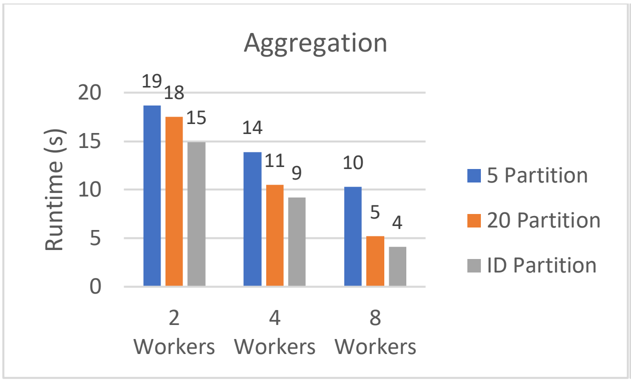


Figure 4

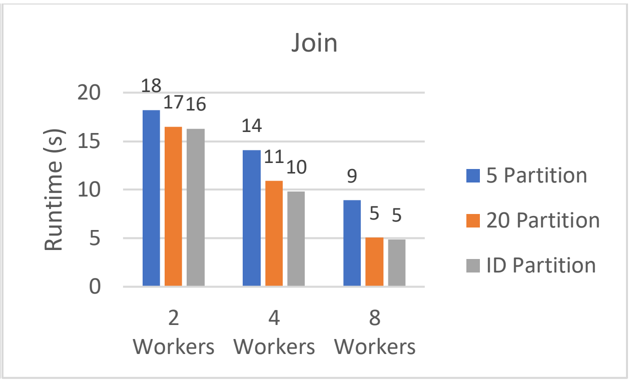


Figure 5

4.1.3 Machine Learning

Next, I explored how machine learning (ML) training process is like on these two platforms. Although there are many large datasets out there for machine learning, the size of dataset usually shrinks quite a lot after aggregation. So in order to test Databricks' and AWS Spark's ability to handle ML model training on large datasets, I decide to use the MovieLens 20M dataset (700 MB) and leverage the recommendation API in MLlib to build a recommender system. Basically, given a user-item-rating matrix, the algorithm will mathematically decompose the matrix into two submatrices (user feature matrix and item feature matrix). The recommendation API in MLlib implement alternating least squares to perform matrix approximation during training process. Since the entire 700 MB data source will be mathematically decomposed during the training process, it is easy for us to test the performance differences between these two platforms and also observe scale up effect. For the hyperparameters, *ranks* is set to 3, *iteration* is set to 20, and *lambda_* is set to 0.1. Since evaluating performance of models is not in the scope of this project, I didn't use large iteration number for perfect convergence. The result is shown in **Figure 6** and the effect of scale up is evident in the chart. As the training process is basically matrix multiplication, errors calculation, and then weight update, reduction in the dimensions of the data source substantially lowers the training time. One thing to note is that the training time of full data on 4 workers is nearly identical to that of 50% data on 2 workers. This means the tradeoff between data size and number of workers is quite linear at least at this stage. Looking at the comparison between Databricks and AWS Spark, we found that the performances are similar at small scale of data. However, as the scale of data increases, Databricks starts outperforming AWS Spark again. Especially on full data with 2 workers, runtime of Databricks is 72% shorter than AWS Spark. It is not done yet, but I plan to look into the Spark UI and see how Spark papalizes and distributes the approximation process to executors.

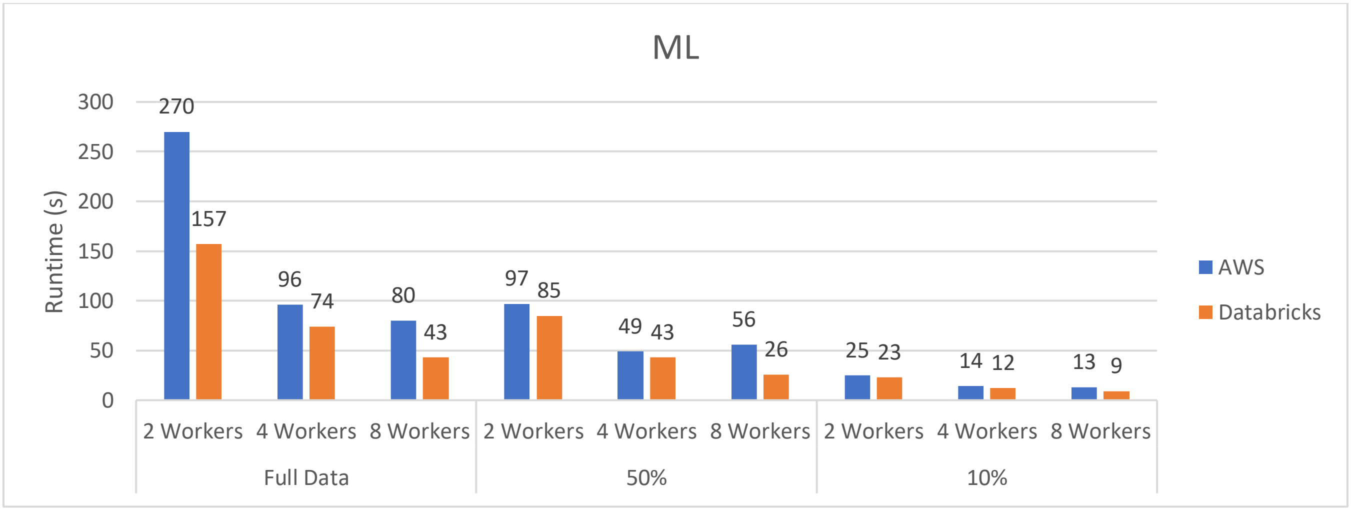
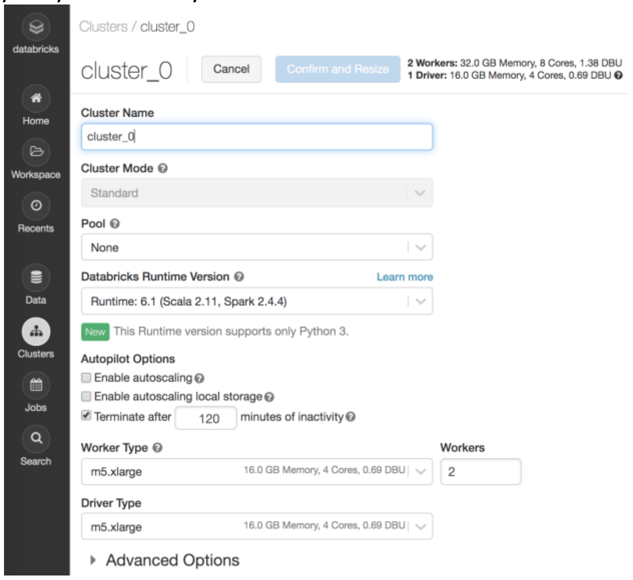


Figure 6

4.2 Ease of Use Testing

4.2.1 Working with Clusters

Creating EMR clusters in AWS Spark includes several steps such as selecting integrated services, selecting worker types, configuring security groups, and linking to key pems. The whole process is not complicated but for first time users of AWS, it might be vague and not intuitive enough. Compared to setting up Spark clusters in AWS, creating, editing, and monitoring clusters are incredibly easy in Databricks. First of all, to create a cluster, users can just go to the cluster tab, select the version of Databricks Runtime, select worker type and how many workers you want then launch. Users can also tell Databricks to terminate the cluster after how many minutes of inactivity, which is super nice design to avoid wasting credits on clusters that forget to terminate. Also, within the Ui of each cluster, users can see all the notebooks that are attached to this cluster, all the libraries that are installed on the cluster, event log, driver logs, metrics, and Spark UI. Everything has a clean and straightforward look which is very easy on users' eyes.



2 Workers: 32.0 GB Memory, 8 Cores, 1.38 DBU
1 Driver: 16.0 GB Memory, 4 Cores, 0.69 DBU

Cluster Name: cluster_0
Cluster Mode: Standard
Pool: None
Databricks Runtime Version: Runtime: 6.1 (Scala 2.11, Spark 2.4.4)
This Runtime version supports only Python 3.
Autopilot Options
- [ ] Enable autoscaling
- [ ] Enable autoscaling local storage
- [✓] Terminate after 120 minutes of inactivity

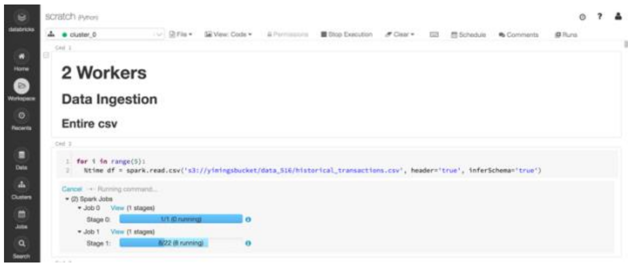
Worker Type: m5.xlarge 16.0 GB Memory, 4 Cores, 0.69 DBU
Workers: 2
Driver Type: m5.xlarge 16.0 GB Memory, 4 Cores, 0.69 DBU

Figure 7

To edit the cluster (change worker type, increase number of workers), users only need to click the *Edit* tab, make the modifications, and then confirm. Based on my experience using Databricks in the past week, the UI is intuitive and very easy to use.

4.2.2 Working with Notebooks

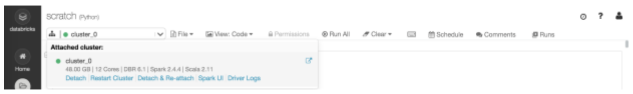
Setting up iPython notebooks in AWS Spark is quite a process. After the cluster is initialized and running, one needs to ssh into the instance, install iPython, write a shell script for launching notebook, and configure local ports to initiate and access the notebook. After finishing coding in a notebook, the environment does not autosave for you and users need to download the notebook and upload it again for any future work. It is definitely not the most human center environment for Spark users. In Databricks, after cluster is launched and running, users can create notebooks attached to the cluster and start doing data analytics. This is way easier than setting up Jupyter Notebook running Spark on AWS. **Figure 8** is a screenshot of the notebook UI. As we can see, it has a simple design and looks very similar to Jupyter Notebook. One thing to mention is that notebooks on Databricks autosave on cloud, so it is super convenient to share work. Specifically, the permission tab allows users to grant privileges to other users for viewing or editing notebooks. Let's say you did some exploratory analysis and want to quickly share with your boss. You can copy and paste the URL of the notebook and send it to him through slack or email, and then he can just open it in browser. If he wants to make a change, the notebook will take care of version control for you. Other than the permission tab, other tabs on top of the notebook are self-evident. Users can detach, restart, and check the storage of the cluster as shown in **Figure 9**. User can clone the notebook to back up the notebook or export it as IPython notebook as shown in **Figure 10**. One thing to note in **Figure 8** is that Databricks provides in-line progress bar of executions. Users can get overview of progresses of their programs by checking the blue progress bar. And if more information is needed by the user, he can just click on *View* to open the full Spark UI, which shows all the details about jobs, stages, and memories etc. I think the blue progress bars are super satisfying for people to look at. Additionally, when it comes to data analytics, communicating results with visualization is crucial. And luckily, Databricks integrates with Tableau and allows dashboard building directly by any plot within the notebook. Plots can even be directly generated by SQL queries, which makes it very easy to edit or maintain dashboard.



```
for i in range(5):
  %time df = spark.read.csv('s3://yiningsbucket/data_516/historical_transactions.csv', header='true', inferSchema='true')
```

```
Cancel  Running command...
▾ (2) Spark Jobs
  ▾ Job 0   View (1 stages)
      Stage 0:  1/1 (0 running)
  ▾ Job 1   View (1 stages)
      Stage 1:  8/22 (8 running)
```

Figure 8



Attached cluster:
cluster_0
48.00 GB | 12 Cores | DBR 6.1 | Spark 2.4.4 | Scala 2.11
Detach | Restart Cluster | Detach & Re-attach | Spark UI | Driver Logs

Figure 9

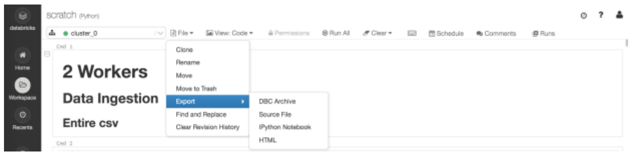


Figure 10

4.2.3 Working with Job Scheduler

As a unified data analytics platform, of course Databricks offers job scheduling feature. Users can directly add scripts to job scheduler for scheduled run. Since Databricks supports different Runtime Version, users writing scripts running on different runtime versions efficiently. A xgboost training on regular runtime version can happen right after a ResNet training on ML GPU runtime version, which can just be a snapshot of an entire data science pipeline.

4.2.4 Cost

Cost of using AWS Spark mainly depends on the worker type, and AWS also provide cost explorer for user to review their spending. However, cost of using Databricks might be the only downside of this platform. 2 m5.xlarge workers cost $0.55 per hour not including AWS charges. In practice, we usually need much more workers to handle large volume of data so users can potentially spend considerable amount of money using this platform.

5. Summary

In conclusion, Databricks runs faster than AWS Spark in all the performance test. For data reading, aggregation and joining, Databricks is on average 30% faster than AWS and we observed significant runtime difference (Databricks being ~50% faster) in training machine learning models between the two platforms. In terms of ease of use, Databricks is much more user friendly than AWS Spark by offering easy setups, integrated notebooks, and easy collaborations. In terms of cost, AWS Spark is on the better side as users only pay based on worker type and size, but Databricks puts quite a lot more cost on users and the cost is highly correlated with the worker type as well. In practice, people usually use more power worker types and more amount of workers as well, so the cost can actually be substantially higher when using Databricks.