System Investigation Part 4: Apache Spark Paper

Ross Friscia
Data Science
University of Washington
Seattle, Washington, USA
rafrisci@uw.edu

ABSTRACT
In this paper, I discuss Apache Spark and how it handles the processing and querying of data. The subjects of user interface, indexes, consistency, scaling, and security are discussed. I also explore the querying and storage of AirBnB data utilizing Spark along with Azure Databricks.

CCS CONCEPTS

- **Information systems → Data management systems**; *Database management system engines;*

KEYWORDS
datasets, data lakehouse, scaling, Spark SQL, RPC,

ACM Reference format:
Ross Friscia. 2022. System Investigation Part 4: Apache Spark Paper. In *Proceedings of DATA 514: Data Management for Data Science. Seattle, WA, USA, 4 pages.*

1 INTRODUCTION

For this project I selected Apache Spark as my system. From the website, “Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters” [2]. Spark allows users to use Python, SQL, Scala, Java, or R to perform all its capabilities. Those capabilities include: intaking static or streamed data, data analysis, data transformation, data science, and machine learning. Users interact with Spark through one of its four libraries: Spark SQL and DataFrames, Spark Streaming, MLib, and GraphX. Each library allows users to perform one of the different capabilities mentioned earlier.

The real power of Spark is how it utilizes distributed systems to complete tasks. Instead of relying on one machine to complete the tasks asked of the system, a distributed system receives a task, gives pieces of the task to different machines that belong to the distributed system, these machines (called nodes) complete the task, and the completed task is put together for the user who requested it. Distributed systems rely on a system of many cheap machines instead of the traditional method of one very expensive machine handling tasks. Another benefit of this is the machines are easy to replace and it is easier to upgrade your system. Also, a machine may fail in a distributed system and another machine will pick up the work and complete the task without the user seeing any of it. [4]

The number of worker nodes set to perform tasks in Spark are determined at the cluster and job level. Spark clusters are set up with a minimum and maximum number of workers. The cluster starts with the minimum number of workers but can scale up to the maximum number of workers depending on the job it is handling. Spark will handle this scaling automatically, which can run tasks faster than constant-sized clusters or systems and reduce overall costs by not running at maximum capacity all the time. Autoscaling can be ignored, and the user can set the number of cores at the job level if so desired.

2 SYSTEM SUMMARY

2.1 Data Model

The data used for my system design is the AirBnB dataset provided by the class [1]. An example of the data model I would use can be found in the E/R diagram below. There were too many attributes to cleanly include each in the diagram, so property_info and host_info for the listings entity as well as reviewer_info for review entity represents multiple attributes. Property_info includes property_name, city, state, neighborhood, property_type, room_type, amenities, bedrooms, beds, and price, host_info includes host_id, host_name, host_response_rate, and host_acceptance_rate, and reviewer_info includes reviewer_id and reviewer_name.

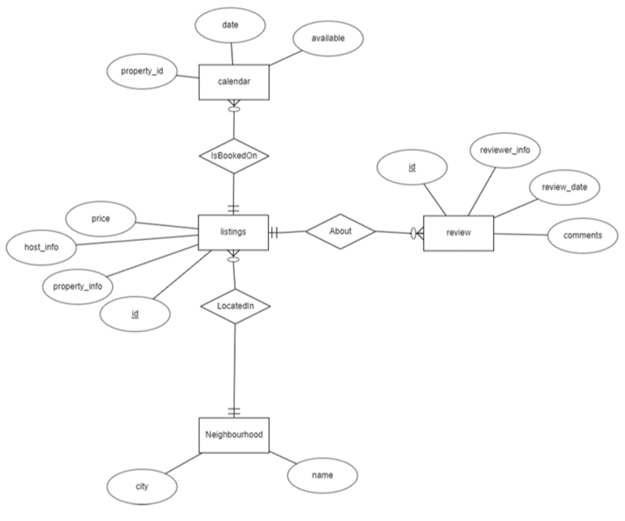
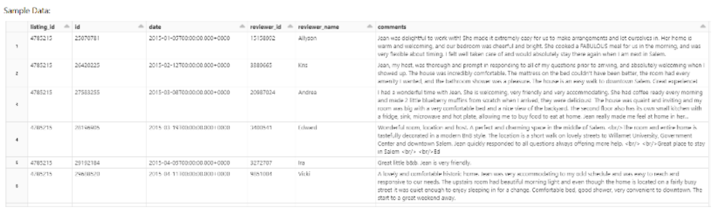


Figure 1: E/R Diagram of Implemented Data Model

The below image is a sample of the review table loaded into my system. This was done in Azure Databricks when determining the best data model for the queries I would want to run on the data.


Sample Data:

	listing_id	id	date	reviewer_id	reviewer_name	comments
1	4785215	25070781	2015-01-05T00:00:00.000+0000	15158902	Allyson	Jean was delightful to work with! She made it extremely easy for us to make arrangements and let ourselves in. Her home is warm and welcoming, and our bedroom was cheerful and bright. She cooked a FABULOUS meal for us in the morning, and was very flexible about timing. I felt well taken care of and would absolutely stay there again when I am next in Salem.
2	4785215	26420225	2015-02-12T00:00:00.000+0000	3389665	Kris	Jean, my host, was thorough and prompt in responding to all of my questions prior to arriving, and absolutely welcoming when I showed up. The house was incredibly comfortable. The mattress on the bed couldn't have been better, the room had every amenity I wanted, and the bathroom shower was a pleasure. The house is an easy walk to downtown Salem. Great experience!
3	4785215	27583255	2015-03-08T00:00:00.000+0000	20987024	Andrea	I had a wonderful time with Jean. She is welcoming, very friendly and very accommodating. She had coffee ready every morning and made 2 little blueberry muffins from scratch when I arrived, they were delicious! The house was quaint and inviting and my room was big with a very comfortable bed and a nice view of the backyard. The second floor also has its own small kitchen with a fridge, sink, microwave and hot plate, allowing me to buy food to eat at home. Jean really made me feel at home in her...
4	4785215	28196905	2015-03-19T00:00:00.000+0000	3400541	Edward	Wonderful room, location and host. A perfect and charming space in the middle of Salem.
 The room and entire home is tastefully decorated in a modern BnB style. The location is a short walk on lovely streets to Willamet University, Government Center and downtown Salem. Jean quickly responded to all questions always offering more help.

Great place to stay in Salem

Ed
5	4785215	29192184	2015-04-05T00:00:00.000+0000	3272707	Ira	Great little b&b. Jean is very friendly.
6	4785215	29680520	2015-04-11T00:00:00.000+0000	9851004	Vicki	A lovely and comfortable historic home. Jean was very accommodating to my odd schedule and was easy to reach and responsive to our needs. The upstairs room had beautiful morning light and even though the home is located on a fairly busy street it was quiet enough to enjoy sleeping in for a change. Comfortable bed, good shower, very convenient to downtown. The start to a great weekend away.

Figure 2: Sample of Implemented Review Table

2.2 User Interface

For this project I explored deploying a Spark system on Azure Databricks, a managed platform that is used to transform large quantities of data and to explore data. This exploration was done using the Salem, OR AirBnB data. While Spark has its own UI called Web UI to monitor and inspect Spark job executions, Databricks provides a UI for notebooks, stored data, Spark clusters, and job logs. Databricks allows users to point and click in its UI instead of code solutions too. If users would prefer to code tasks, then they can use one of the programming languages mentioned earlier.

The Spark Web UI also comes with security features with can limit users who can view and modify applications, support TLS/SSL connections, event logs for applications, and HTTP Security Headers. [2]

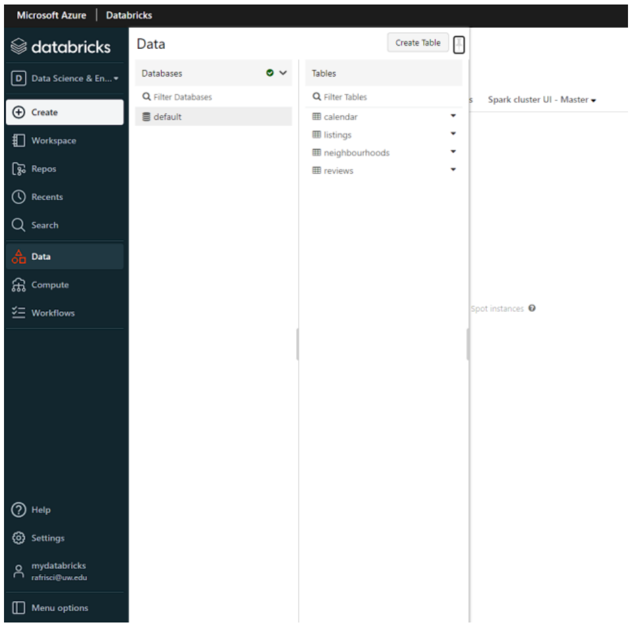
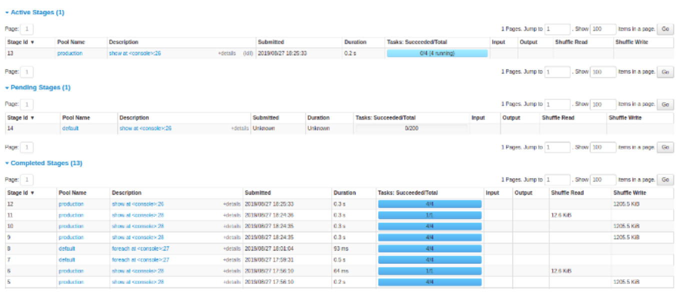


Figure 3: Databricks UI Screenshot


▾ Active Stages (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
13	production	show at <console>:26 +details (kill)	2019/08/27 18:25:33	0.2 s	0/4 (4 running)				

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

▾ Pending Stages (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	default	show at <console>:26 +details	Unknown	Unknown	0/200				

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

▾ Completed Stages (13)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
12	production	show at <console>:26 +details	2019/08/27 18:25:33	0.3 s	4/4				1205.5 KiB
11	production	show at <console>:28 +details	2019/08/27 18:24:36	0.3 s	1/1			12.6 KiB	
10	production	show at <console>:28 +details	2019/08/27 18:24:35	0.3 s	4/4				1205.5 KiB
9	production	show at <console>:28 +details	2019/08/27 18:24:35	0.3 s	4/4				1205.5 KiB
8	default	foreach at <console>:27 +details	2019/08/27 18:01:04	93 ms	4/4				
7	default	foreach at <console>:27 +details	2019/08/27 17:59:31	0.5 s	4/4				
6	production	show at <console>:28 +details	2019/08/27 17:56:10	64 ms	1/1			12.6 KiB	
5	production	show at <console>:28 +details	2019/08/27 17:56:10	0.2 s	4/4				1205.5 KiB

Figure 4: Spark UI Example Provided in Documentation [2]

2.3 Indexes

In the past, Spark did not consider indexes important and was not a part of the system. This is because Spark was built more with data engineering in mind than data management. Spark is a distributed system, and it is difficult to index the data when it is distributed across many machines. However, there is a recent development that may change this. Hyperspace introduces the ability to index with Apache Spark. However, it is only recommended to use if "queries contain filters on predicates with high selectivity" or "queries contain a join that requires heavy shuffles" [5]. This is because those are the circumstances the creators have determined indexing will increase the speed of workloads or queries. This development is new, and my current data load does not fulfill either of the recommended conditions to use it, so I did not implement indexing.

2.4 Consistency

Spark does not comply with atomicity, consistency, isolation, and durability (ACID) standards as it is mainly meant to process static and streaming data into a storage system. However, this changes if using Spark with Databricks. Their partnership with Delta Lake makes it so "each write is an isolated transaction that is recorded in an ordered transaction log" [3]. So, while Spark does not satisfy ACID transactions on its own, in combination with other products it can.

2.5 Scaling

As mentioned previously, Spark scales according to the tasks it needs to complete by increasing the number of workers if the task is more intensive. It also performs tasks using Resilient Distributed Datasets (RDDs) which are fault-tolerant and connected by Directed Acyclic Graphs (DAGs). These systems ensure that the distributed system know the tasks that need to be done and that the individual machines are working. However, the decision of sharding or replication is up to the system that Spark stores the data in. In my implementation, the data is stored in Databricks which allows me to utilize Delta clones to replicate the stored data.

2.6 Security

Mentioned earlier, Spark Web UI security does a good job of managing access of users and monitoring application usage but there is additional security between Spark processes. Remote Procedure Calls (RPCs) are what allow Spark processes to communicate with each other. This is done by Spark applications providing a shared secret to a daemon (driver that starts the executors). This provides a proof of identity to the daemon allowing the application to provide or gain access to the necessary data. The generation and distribution of shared secrets depend on deployment, but all deployments support them. [6]

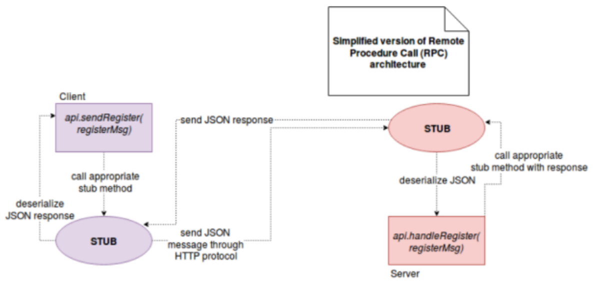

Simplified version of Remote Procedure Call (RPC) architecture
Client
api.sendRegister(registerMsg)
send JSON response
STUB
call appropriate stub method with response
call appropriate stub method
deserialize JSON
deserialize JSON response
STUB
send JSON message through HTTP protocol
api.handleRegister(registerMsg)
Server

Figure 5: RPC Architecture Example [6]

3 SYSTEM INSTALLATION

The AirBnB data for my Spark system would be loaded into Azure Databricks. As shown in the Databricks UI screenshot as well as the E/R diagram the data is split up into four different tables. This would be accomplished by expanding the notebooks used to load the Salem data to load the data of the other cities. There was some data cleaning that had to be performed to the Salem file that would need to be utilized for the other data as well.

For one, when loading the listings data, the option multiLine had to be enabled in spark.read.format to read commas in the description field without causing the rest of the description text to go into the next row as the files were all comma separated. I also had to add some Python code to introduce backslashes in front of any parentheses in the description text so the data would load properly. I also changed the neighhborhood_group column in neighbourhoods to have the city name in the data. Finally, there were some additional columns that were unnecessary that I removed.

Other considerations that would need to be made when loading the rest of the data would be the handling of schema and appending to a table rather than writing a new table. The notebooks that created the four tables were great for that purpose, but the additional data being loaded into those tables will need to be appended to the table not overwriting it. A benefit of doing this using Spark is that I can evaluate the data being loaded in temporary tables before pushing it to the production table, ensuring that only properly cleaned data makes it to the production tables. As for the schema, I manually created it for the four tables rather than utilizing the inferSchema option in the spark.read.format as there were some issues with it handling some integer fields as floats. This issue sounds small, but it would impact some queries as well as how the data in that column could be interpreted.

One variant of a system design I considered was separating the host_info into its own table and having host_id as a foreign key in the listings table. While I initially thought that this was the optimal way of organizing the data model, I realized that my selected queries in the next section did not necessitate such a structure, that it would create an additional join in my queries slowing them down, and it would require extra work to separate the information out of the listings table.

4 SYSTEM DESIGN

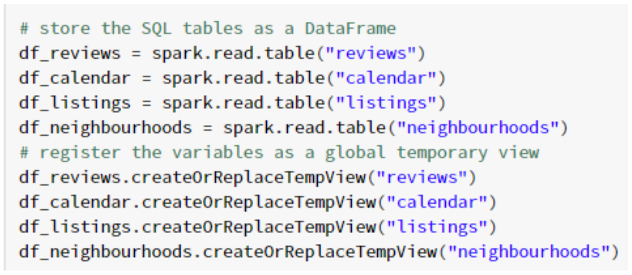
As mentioned previously, my data would be stored in Azure Databricks and across four tables. I also selected the following queries to answer:

1 Are there any neighbourhoods in any of the cities that don't have any listings?

2 (Booking trend) For each city, how many reviews are received for December of each year?

3 (AirBnB search) Display list of stays in Portland, OR with details: name, neighbourhood, room type, how many guests it accommodates, property type and amenities, per night's cost and is available for the next two days in descending order of rating.

My data model would follow the E/R diagram in figure 1. I would not create any indexes because the functionality is not aligned with the data my installation is processing. I would also use Spark SQL to query the data as that is the best Spark library to perform such tasks. After initializing a Spark session, using spark.read.table to set the tables as a DataFrame in Python, and registering the DataFrame as a global temporary view using createGlobalTempView, I would be able to use the spark.sql function to write SQL queries in.

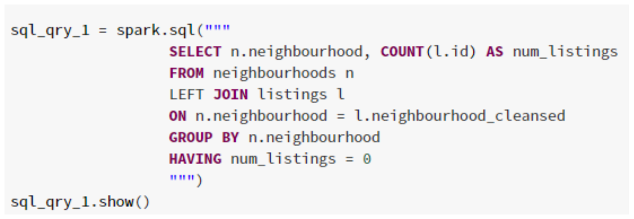


```
# store the SQL tables as a DataFrame
df_reviews = spark.read.table("reviews")
df_calendar = spark.read.table("calendar")
df_listings = spark.read.table("listings")
df_neighbourhoods = spark.read.table("neighbourhoods")
# register the variables as a global temporary view
df_reviews.createOrReplaceTempView("reviews")
df_calendar.createOrReplaceTempView("calendar")
df_listings.createOrReplaceTempView("listings")
df_neighbourhoods.createOrReplaceTempView("neighbourhoods")
```

Figure 6: Preparing the Data For Querying

4.1 Query 1: Neighborhoods Without Listings

For this query the solution involves both the neighbourhoods table as well as the listings table. I would left join the listings table to the neighbourhood table, select the neighbourhood column and aggregate listing IDs in a count, and then filter the query for any instance where the count of listing ID's is zero.



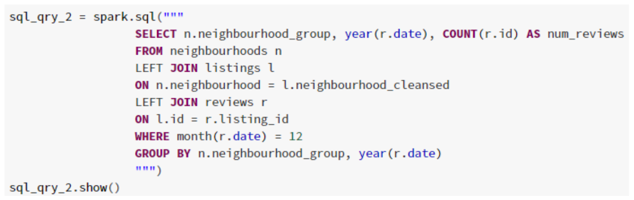
```
sql_qry_1 = spark.sql("""
                    SELECT n.neighbourhood, COUNT(l.id) AS num_listings
                    FROM neighbourhoods n
                    LEFT JOIN listings l
                    ON n.neighbourhood = l.neighbourhood_cleansed
                    GROUP BY n.neighbourhood
                    HAVING num_listings = 0
                    """)
sql_qry_1.show()
```

Figure 7: Query 1 Solution

Utilizing the alternate system design separating host information into its own table would do nothing to change the querying or performance of this solution.

4.2 Query 2: Booking Trends

Query 2 will be using the neighbourhoods table again, but this time utilizing neighbourhood_group for the cities. It will left join to listings again, and listings will left join to reviews on the listing IDs. I would select the neighbourhood_group, year of the review date, and aggregate the review IDs in a count. I would also filter the month of the review date so it equals 12.



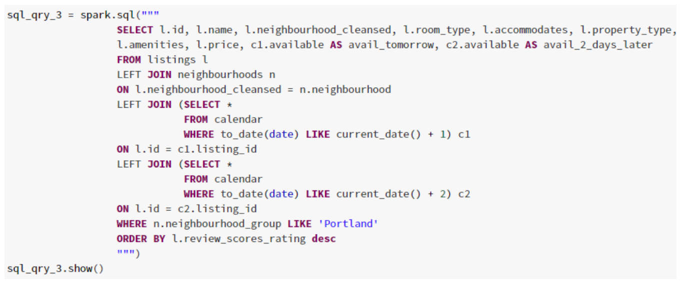
```
sql_qry_2 = spark.sql("""
                    SELECT n.neighbourhood_group, year(r.date), COUNT(r.id) AS num_reviews
                    FROM neighbourhoods n
                    LEFT JOIN listings l
                    ON n.neighbourhood = l.neighbourhood_cleansed
                    LEFT JOIN reviews r
                    ON l.id = r.listing_id
                    WHERE month(r.date) = 12
                    GROUP BY n.neighbourhood_group, year(r.date)
                    """)
sql_qry_2.show()
```

Figure 8: Query 2 Solution

Once again, utilizing the alternate system design would not change the performance or query of this solution.

4.3 Query 3: AirBnB Search

Query 3 would involve the listings, neighbourhoods, and calendar tables. The listings table would provide all the columns except the availability ones. The calendar table would need to be joined with the listings table twice, once for the availability tomorrow and again for the next day. I will also filter the neighbourhood_group column to be equal to Portland, and order by review_scores_rating descending.



```
sql_qry_3 = spark.sql("""
                  SELECT l.id, l.name, l.neighbourhood_cleansed, l.room_type, l.accommodates, l.property_type,
                  l.amenities, l.price, c1.available AS avail_tomorrow, c2.available AS avail_2_days_later
                  FROM listings l
                  LEFT JOIN neighbourhoods n
                  ON l.neighbourhood_cleansed = n.neighbourhood
                  LEFT JOIN (SELECT *
                             FROM calendar
                             WHERE to_date(date) LIKE current_date() + 1) c1
                  ON l.id = c1.listing_id
                  LEFT JOIN (SELECT *
                             FROM calendar
                             WHERE to_date(date) LIKE current_date() + 2) c2
                  ON l.id = c2.listing_id
                  WHERE n.neighbourhood_group LIKE 'Portland'
                  ORDER BY l.review_scores_rating desc
                  """)
sql_qry_3.show()
```

Figure 9: Query 3 Solution

This query would also not be affected by utilizing the alternate structure either. However, another structure that may benefit this query would be creating a calendar table for each year and making each day a different column with the availability as the data in the column. I believe this structure would improve the speed and ease of the query, but it would be a negative overall for the data structure. Creating a new table for each year would mean having to update the query every year to the new table, creating a clutter of tables in the database, and creating difficulties for load new data and querying across years.

5 Conclusion

Apache Spark is an excellent data processing system. The use of its libraries makes querying the data loaded little more work than a typical SQL query. This, along with its distributed system explains why the system is so widely used. Spark can handle streaming or static data and answer a variety of questions. Utilized with Azure Databricks, this provided an excellent option in processing and querying the AirBnB data.

ACKNOWLEDGMENTS

I would like to thank Professor Kristin Tufte as well as the TA's Srivalya Elluru and Punya Shetty, for this course, providing project ideas, and answering my questions in a timely manner.

REFERENCES

[1] AirBnB. Get the Data. Retrieved June 5, 2022 from http://insideairbnb.com/get-the-data/
[2] Apache Spark. Apache spark™ - unified engine for large-scale data analytics. Retrieved June 5, 2022 from https://spark.apache.org/
[3] Databricks. 2022. Data Lakehouse Architecture and AI Company. (May 2022). Retrieved June 5, 2022 from https://databricks.com/
[4] Distributed Systems. What are distributed systems? an introduction. Retrieved June 5, 2022 from https://www.splunk.com/en_us/data-insider/what-are-distributed-systems.html
[5] J. Juluczni. Hyperspace indexes for Apache Spark - Azure Synapse Analytics. Retrieved June 5, 2022 from https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-performance-hyperspace
[6] R.PC Spark. RPC in Apache Spark. Retrieved June 5, 2022 from https://www.waitingforcode.com/apache-spark/rpc-apache-spark/read