Project Group: R10A-05
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Description:

The purpose of this assessment is to understand Apache Spark in a Pythonic environment. The first question is to design and produce results from 3 different tasks, each using separate tables except for the on time performance tables. The second question is to perform tuning on each task in question 1 to optimize performance and decrease runtime.

TASK 1

Introduction:

Overall, the main dataset used for this question is the aircrafts_csv as well as flights_csv. The flight dataset is constantly modified and therefore the aircrafts dataset will be joined with different flight_csv at small, medium, large and massive size. All the tests are conducted multiple times and the result is obtained based on the average of each runtime.

Method:

1. Several PySpark functions are imported; for example, a regexp_extract is used to extract the number of each flight model, and an initcap is imported to capitalise the first letter of the manufacturer variable.
2. A pathway such as flight_path and aircrafts_path is created to read the dataset from DBFS.
3. The data is then sliced into 'tailnum', 'manufacturer' and 'model variable' in the aircrafts dataset; 'flight_num' and 'tail_number' in the flight dataset.
4. The column's name is renamed to appropriate and consistent names such as 'tailnum' to 'tail_num' and' flight_number' to 'flight_number' to remove the whitespace in the columns.
5. A new data frame, flightnum_clean and aircrafts_clean, is created, and it stores all the desired data values.
6. The two data frames are then left_outer join on variable 'tail_num' = 'tail_num', forming a combined data frame called model_flight.
7. A groupby().count() function is applied to determine the number of occurrences based on the variable tail_number in descending order.
8. After obtaining the result, a full_model data frame is created to store and join with the model_clean_df to find its corresponding manufacturer.
9. The 'regexp_extract' function is used to choose only the digit in the values, three numbers in this case, and store them into the new data frame top3.
10. Finally, we combined the two desired columns and used initcap function only to capitalise the first letter of the model, displaying all the values in the format of 'Cessna 123.'

Table of result

Task 1	Unoptimized	Optimized/sort-merge join	Optimized/broadcast
Small Dataframe	5.83 second runtime	3.49 second runtime	4.1 second runtime
Medium Dataframe	17.24 second runtime	9.55 second runtime	13.52 second runtime
Large Dataframe	1.71 minutes	54.77 second runtime	37.48 second runtime
Massive Dataframe	13.54 minutes	6.5 minutes	4.76 minutes

Project Group: R10A-05
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Note - Results are an average for 5 runs of each size dataframe.

****See appendix for all execution plans.**

Result:

For the data associated with task 1, finding the top3 Cessna model is relatively simple as there are only two datasets involved. It does not require any complex filtering method neither algorithm, but an inbuilt group by function is used, namely group by in spark As a result, the data displays that the top 3 model by Cessna company is always Cessna 172,210 and 550 respectively, various by the size of data.

Evaluation for data frame sort-merge.

The first approach displayed here is the most general approach with the ideology of select, filter, and result. A large data size is used for demonstration, showing that 14,285,416 data is found in the flights dataset, and 12 rows of output is found in the aircraft dataset. The data bricks system has an inner optimiser which would choose the more optimised approach for the task requested. The idea of a sort-merge join is applied; First, the data is shuffled and sorted internally, performing the join based on the same join key with the same worker, tail_number. As a result, the data that fits the criteria is reduced to 20,134, much smaller than before. Followed by a HashAggregate, and thus the rank of each model is calculated based on tail_number. Finally, the table of results is then joined back with the aircrafts table, adjusting the result in the desired form. In analysis, this method has highlighted the most straightforward way of obtaining the result; using a sort-merge join is an efficient way since it fits well in the case where you need a quick sort. However, the DAG tables have highlighted that this approach consists of many repetitive scanning, reading and filtering, increasing the runtime as shown in the result table. In conclusion, this approach has the lowest complexity but highest runtime, thus the result remains satisfying but needs improvement.

Optimisation 1 execution improvement:

The new execution plan is designed to reduce the number of unnecessary scanning and reading; this approach is made using the sqlContext.read.csv; the method allows the computer to select, filter, and clean the dataset while reading the data. This workflow is also known as pipelining, in which the job is proceeding together instead of one by one. As a result, there is a significant improvement in the processing runtime, which is around 51 per cent faster than the previous approach.

Optimisation 2 broadcast joining

Broadcast join is another optimization method; the idea is mentioned in the week six lecture in which one join table has a filter condition which reduces it to a small enough size to broadcast filtered tuples to all nodes, with subsequent local join per node. The dataset is used to fit well in broadcast joining since the filter condition on only selecting aircrafts made by Cessna company has reduced the dataset to a row output of twelve which is relatively small, joining with the much larger flights dataset that has around fourteen million rows . This is because internally, the large dataset will be partitioned over all n nodes, and broadcast S filtered will join with all nodes with R partitions. In practice, the broadcast join has no significant improvement in working with a small dataset, but it has a significant surge in working with the large and massive data set. Statistically, around 25.6 per cent on average faster in working with large and massive data.

****See appendix for all results.**

TASK 2

Task 2: Job Design Decision and Results

Two dataframes were created, one for the performance dataset, and the other for the airlines dataset. Some important Spark functions that were imported include format_string, to_date, year, and when functions. Using the format_string function to insert trailing zeros into the actual and scheduled departure times to make them consistent, the times were then split into hour and minute, with the first two objects in the string representing the hour, and the last two objects in the string representing the minute. This part of the operation was especially important when replacing all ‘00’ with ‘24’ which represented midnight. The to_date and year functions were then used to extract the year from the flight_date column, to filter out any flights that only occurred within a specific year

defined by the user. The when function was used to change all '00' in the hour column to '24' which represents midnight. This made it simpler to convert the hour into total minutes, to calculate the minutes delayed. The required part of this task utilized the aggregate function to calculate the minimum, maximum, average delayed minutes, and the total count of delays that occurred for flights that took place in the year 2000 (specified by user) and only United States airlines. Two separate joins were used in this task, the first one to join the performance dataset and the airlines dataset, using carrier_code as the join operator. The second join was to combine all aggregate function calculations into one comprehensive table that displayed the airline, max, min, and average minutes of delay, and the total count of delays for each airline.
Prior to tuning, the flow was to first create two data frames, then filter out information and finally perform calculations. When initially composing code for this task, many code logic decisions were not a main priority as the main focus to ensure proper results. Therefore, the structure of the code may seem elementary and unstructured, which may have affected performance, although the results were consistent in both the unoptimized and optimized versions of code which meant the unoptimized code produced proper results.
Also, RDD wasn't deployed in the unoptimized code, but the dataframe API was deployed. The reason behind this was to gauge how tuning methods could affect performance for the same type of API, for the sake of fairness (it is known that RDD which is the basis of dataset and dataframe always performs worse than dataframe, albeit with some exceptions - unstructured data types, low level transformations or or actions). Further, RDD does not pertain to a schema with columns and rows like dataframe which can effectively cause errors in data manipulation.
****See appendix/output file for all results.**

Task 2: Optimization/Tuning and Performance Evaluation

Code logic tuning played an important role in this task. Many calculations were performed, especially on the 'ontimeperformance_flights' dataframe. Many filters, groupby functions, and calculations were implemented: filtering for year = 2000; calculations on scheduled departure and actual departure; converting the hour to minutes; filtering delayed flights; and performing aggregate calculations to determine the min, max, average, and total count on delays. Structuring code to ensure data frames were called from disk to memory as few times as possible increased performance and decreased runtime.

In addition to code logic, several tuning methods were implemented to increase performance especially in larger datasets. Three code blocks were implemented. The first being the original data frame, the second using both broadcast joins and caching, and the third being an extension of the second but additionally using repartitioning and coalescing. The original data frame produced the worst run time due to its unoptimized structure and lack of tuning. The second method performed significantly better than the original data frame due to its optimized code structure, and the use of broadcast joins and caching. The caching of data frames that were used most frequently greatly increased the run time of the operations that were performed, namely the calculations on time and the aggregation functions (groupBy, aggregate min/max/avg/count). The reason for this is because the data frame was stored in the physical memory, which makes access quicker, rather than physical disk. However, the user must be aware of how much physical memory is available for caching. If the data frame is larger in size than the memory can handle, there may be an 'out of memory' exception thrown.

The third method produced mixed results, giving erratic results depending on the size of the data frame. Manually adjusting the partitions, which is a pipelining method, may help tune performance slightly from the second method, but the partitions are greatly trial and error if one is unsure how to use them. The assumption on why sometimes this method can sometimes perform worse is because Spark automatically makes the most optimal decision on the number of partitions, and therefore trying to manually repartition can sometimes have a detrimental effect on performance. Coalescing can at times increase the performance because it does not perform any reshuffling, while repartitioning reshuffles and causes a performance hit.

It is important to note that for smaller data sets, using less partitions can increase performance. However, it is important to know what function to use (repartition or coalesce). Coalesce does not reshuffle, but repartition reshuffles everything by trying to set equal partitions which can cause a drop in performance, sometimes performing

worse than method two (Spark automatically sets partition sizes). In the small dataset for method 3, using the coalesce function reduced the number of jobs to 4, which contributed to better performance.

Another interesting finding while tuning the large dataset, a combination of repartition and coalesce was used. After filtering the data in the large dataset, a repartition was implemented after a filter because filters can cause unequal partition sizes. After repartitioning the data into 16 repartitions, a coalesce to 1 partition was done prior to performing aggregate calculations. This resulted in quicker performance than implementing either a repartition or coalesce separately. Perhaps coalescing data to perform aggregate calculations increases performance rather than using multiple partitions, then putting them back together to get a result.

It can be reasonably concluded that the most efficient tuning method is using good code logic. The deployment of broadcast joins or caching, and in some cases using repartitioning or coalescing or a combination of both depending on data size can result in better performance. A definitive proof that tuning works is that the number of jobs decreased, originally from 10 jobs in the unoptimized, to 7 jobs in the optimized, and 4 jobs in the optimized with repartitioning/coalescing. Repartitioning and coalescing may increase performance, but overall, only slightly. Therefore, it is important to understand when to deploy these functions, and more importantly, how. Through this exercise, it is conclusive that the most important tuning methods are using good code logic and deploying functions where appropriate.

	Unoptimized	Optimized/Broadcast, Caching/No Repartition	Optimized/Repartition and Coalesce
Small Dataframe	10.4 second runtime	6.8 second runtime (3 partitions - Spark Set)	4.3 second runtime (2 partitions - Repartition) then (1 partition - Coalesce)
Medium Dataframe	42.3 second runtime	19.4 second runtime (8 partitions - Spark Optimized)	16.1 second runtime (4 partitions - Repartition) then (1 partition - Coalesce)
Large Dataframe	324 second runtime (5.4 minutes)	156 second runtime (2.6 minutes) (8 partitions - Spark Optimized)	138 second runtime (2.3 minutes) (6 partitions - Repartition) then (1 partition - Coalesce)
Massive Dataframe	2106 second runtime (35.1 minutes)	1062 second runtime (17.7 minutes) (23 partitions - Spark Optimized)	966 second runtime (16.1 minutes) (18 partitions - Repartition) then (1 partition - Coalesce)

Note - Results are an average for 5 runs of each size dataframe.

TASK 3

Task 3: Job Design Decision and Results

1. Firstly, several PySpark functions are needed to be imported thereby using the feature of these functions to analyse our dataset, and examples of the use of these functions will be shown in steps below.
2. We have created three data frames based on our interested columns, such as Aircrafts:(tail number, manufacturer, model), Flight: (carrier code, tail number), Airlines: (carrier code, name, country).

3. Also, the column's name is renamed to appropriate and consistent names such as 'tailnum' to 'tail_num' and' flight_number' to 'flight_number' to remove the whitespace in the columns. Name is renamed as airline_name.
4. Created a new data frame called data_3 by using the left outer join function to join the airlines dataset on the flights dataset based on the carrier code. Repeat the same steps, use the left outer join function to join the data_3 data set on the aircrafts dataset based on the tail number.
5. Discover lots of null values in column manufacturer and country, so filter the value which is not null, to get a clean dataframe.
6. Since we want the first three digits of the model number, but discover that there are lots of underline symbols between this model number, we have used the translate function to replace this underline symbol by null.
7. Then created a new_model column which contains the first digit numbers from the previous model column by using the regexp_extract function.
8. Also created a new data column called aircrafts_type by using concat function which concat with manufacturer column and new_model column into one column and use the lit function to leave a space between the manufacturer name and new model.
9. Then we created a new data frame called data_type which only selects the airline_name column and aircrafts_type column, and uses filter function to filter the arline_name which is not null again.
10. Then we order by airline_name in alphabetical order by using asc(), and we use the group by function group the airline_name and aircrafts_type, then using count to count number of aircrafts for each aircrafts_type, to use desc to sort the number of aircrafts in descending order.
11. Finally we need to use window functions to attain the rank of each row based on aircrafts_name and count, and subsequently filter our results to only keep the top 5 aircrafts models.
12. Hence, we use the collec_list function at the end by grouping the airline_name first then use agg.collect_list function to collect the top 5 aircraft_type models for each airline. Finally to use the show function to show our final output and this output is in the appendix page.

Result: The result of my output can be found in the appendix and output file document I have uploaded. The result should contain two column, one is airline_name and sort it in alphabetical order, and second column is aircrafts_type, and it will contain a list of aircrafts_types for each airline, in that list, it will store the top 5 aircrafts_types and sort by number of aircrafts_types in descending order, and in the output of the aircraft_types contains first leading 3 digits which I have extract them from the model column.

Optimization/Tuning and Performance Evaluation

The main changes made to optimise performance was:

1. BroadCast Join : Since the filtered airline and aircraft files are much smaller than the flight files, broadcast hash joins are used thereby fasting our running time.
2. Cache: Use the Cache method at the beginning of our three data frames, aircraft, airline and flight which is good for massive data sets. Since memory is faster than disk, reading data frames from the memory cache is very fast. This significantly faster data access speed improves the overall performance of my program.
3. Filter: Applying filters to only United State Airlines at the beginning of the program has two benefits. First of all, each operation will now reduce a column in the entire program, the country column, because it has achieved its purpose at the beginning of the process of filtering airlines. Second, since we filter out non United States airlines, the number of tuples that the program must operate on will be greatly reduced.

4. Reduce column: We could also reduce the number of unnecessary columns required when executing a join function. For example the country has already been chosen via user input, there is no need to keep the "*country*" column of the airlines table.
5. Efficient User Defined Function: it is the same as the basic principles of tasks 1 and 2, in which UDF is used to rank and retrieve the five most commonly used aircraft_types, thereby saving a lot of running time.

Performance Evaluation for unoptimized/optimized method for 4 different sizes of the dataset:

The difference in execution time with optimized one and unoptimized methods is not very much difference between small and medium datasets. Massive dataset takes 10 times as much as large and 42 times as much as medium, which is also shown in the performance output of Task 3 unoptimised. And for two optimized methods, broadcast join and cache, Although these two tuning methods' effect is less noticeable compared to the unoptimized one in small, medium and large data sets, there is still around 10 minutes decrease in execution time for a massive data set which is compared with unoptimized one, and broadcast join perform very well since it only have 10.45 minutes for the execution time which much less than other two.

Task 3	Unoptimized	Optimized/broadcast join	Optimized/cache
Small Dataframe	11.07 second runtime	10.45 second runtime	10.79 second runtime
Medium Dataframe	29.09 second runtime	28.14 second runtime	25.21 second runtime
Large Dataframe	2.73 minutes	2.67 minutes runtime	2.35 minutes runtime
Massive Dataframe	20.59 minutes	10.45 minutes	17.15 minutes

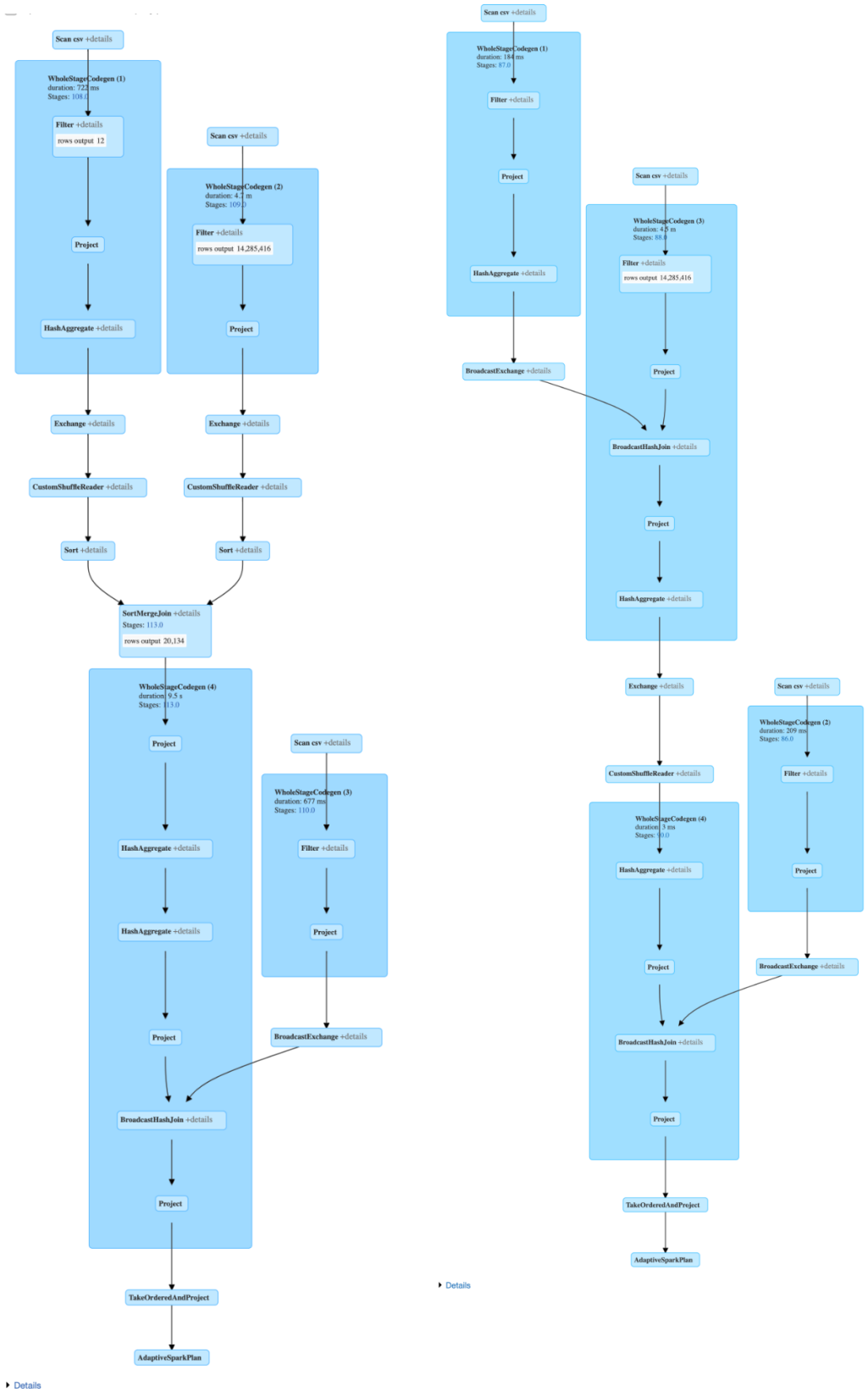
Note - Results are an average for 5 runs of each size dataframe.

****See appendix for all execution plans.**

Project Group: R10A-05
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Appendix: TASK 1

Task 1 Sort merge(left) vs broadcast(right) :



Project Group: R10A-05
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Result:

Small:

	initcap(Flights)
1	Cessna 172
2	Cessna 210
3	Cessna 550

Showing all 3 rows.

Massive:

	initcap(Flights)
1	Cessna 210
2	Cessna 172
3	Cessna 550

medium :

	initcap(Flights)
1	Cessna 210
2	Cessna 172
3	Cessna 550

Showing all 3 rows.

Large:

	initcap(Flights)
1	Cessna 172
2	Cessna 210
3	Cessna 550

Appendix: TASK 2

Task 2:

Results:

Small Dataframe:

▸ (4) Spark Jobs

```
Number of partitions:  6
Number of partitions:  1
```

	airline_name	num_delays	average_delay	min_delay	max_delay
1	Alaska Airlines Inc.	65	24.184615384615384	1	119
2	American Airlines Inc.	251	29.95219123505976	1	379
3	Continental Air Lines Inc.	127	21.874015748031496	1	216
4	Delta Air Lines Inc.	390	31.115384615384617	1	1438
5	Northwest Airlines Inc.	144	33.78472222222222	1	237
6	Southwest Airlines Co.	358	30.044692737430168	2	213
7	US Airways	236	27.944915254237287	1	496
8	United Airlines	375	32.16266666666667	1	380

Showing all 8 rows.

Medium Dataframe:

▸ (4) Spark Jobs

```
Number of partitions:  10
Number of partitions:  1
```

	airline_name	num_delays	average_delay	min_delay	max_delay
1	Alaska Airlines Inc.	594	38.13468013468013	1	1439
2	American Airlines Inc.	2729	28.360205203371198	1	439
3	Continental Air Lines Inc.	1257	28.105807478122514	1	485
4	Delta Air Lines Inc.	3698	21.421849648458625	1	1435
5	Northwest Airlines Inc.	1613	30.36267823930564	1	520
6	Southwest Airlines Co.	3632	27.46613436123348	1	330
7	US Airways	2678	28.435026138909635	1	1435
8	United Airlines	3629	32.68007715624139	1	448

Showing all 8 rows.

Project Group: R10A-05
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Large Dataframe:

▸ (4) Spark Jobs

```
Number of partitions:  20
Number of partitions:  1
```

	airline_name	num_delays	average_delay	min_delay	max_delay
1	Alaska Airlines Inc.	6016	35.16988031914894	1	1438
2	American Airlines Inc.	27102	28.524795218065087	1	1008
3	Continental Air Lines Inc.	12858	29.085394307046197	1	1433
4	Delta Air Lines Inc.	38073	23.338507603813728	1	1439
5	Northwest Airlines Inc.	15850	30.721577287066246	1	1438
6	Southwest Airlines Co.	36837	27.767217743030105	1	465
7	US Airways	26159	27.997018234641995	1	1439
8	United Airlines	36670	32.720152713389695	1	543

Showing all 8 rows.

Massive Dataframe:

▸ (4) Spark Jobs

```
Number of partitions:  30
Number of partitions:  1
```

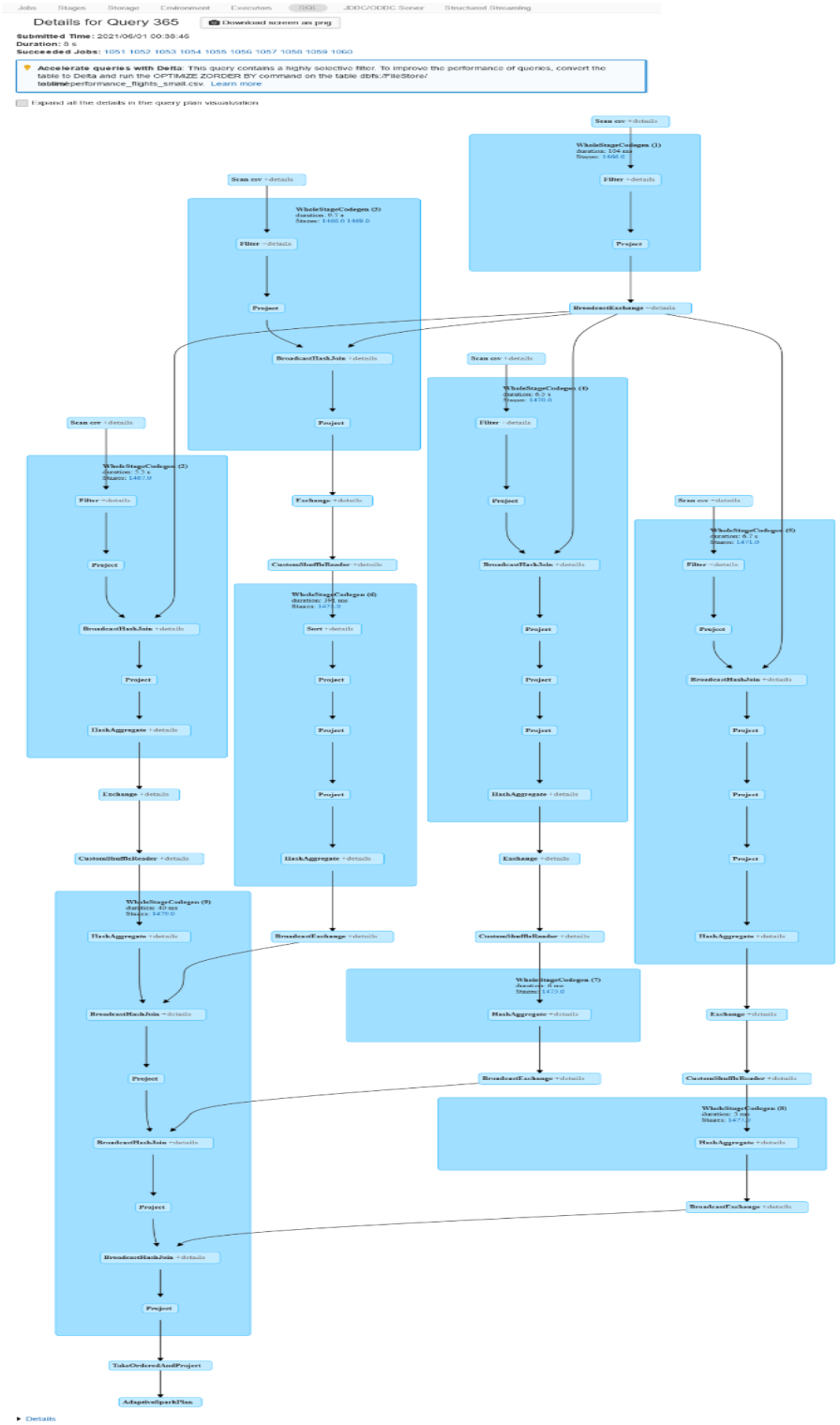
	airline_name	num_delays	average_delay	min_delay	max_delay
1	Alaska Airlines Inc.	47722	34.194627215959095	1	1439
2	American Airlines Inc.	217533	28.249585120418512	1	1434
3	Continental Air Lines Inc.	101757	28.374411588391954	1	1434
4	Delta Air Lines Inc.	302488	22.742634418555447	1	1439
5	Northwest Airlines Inc.	126458	30.47528033022822	1	1439
6	Southwest Airlines Co.	293705	27.92017160075586	1	571
7	US Airways	208097	28.218864279638822	1	1439
8	United Airlines	294144	32.725433801131416	1	684

Showing all 8 rows.

Project Group: R10A-05

SID_1: 490580833 SID_2: 480556587 SID_3:490422382

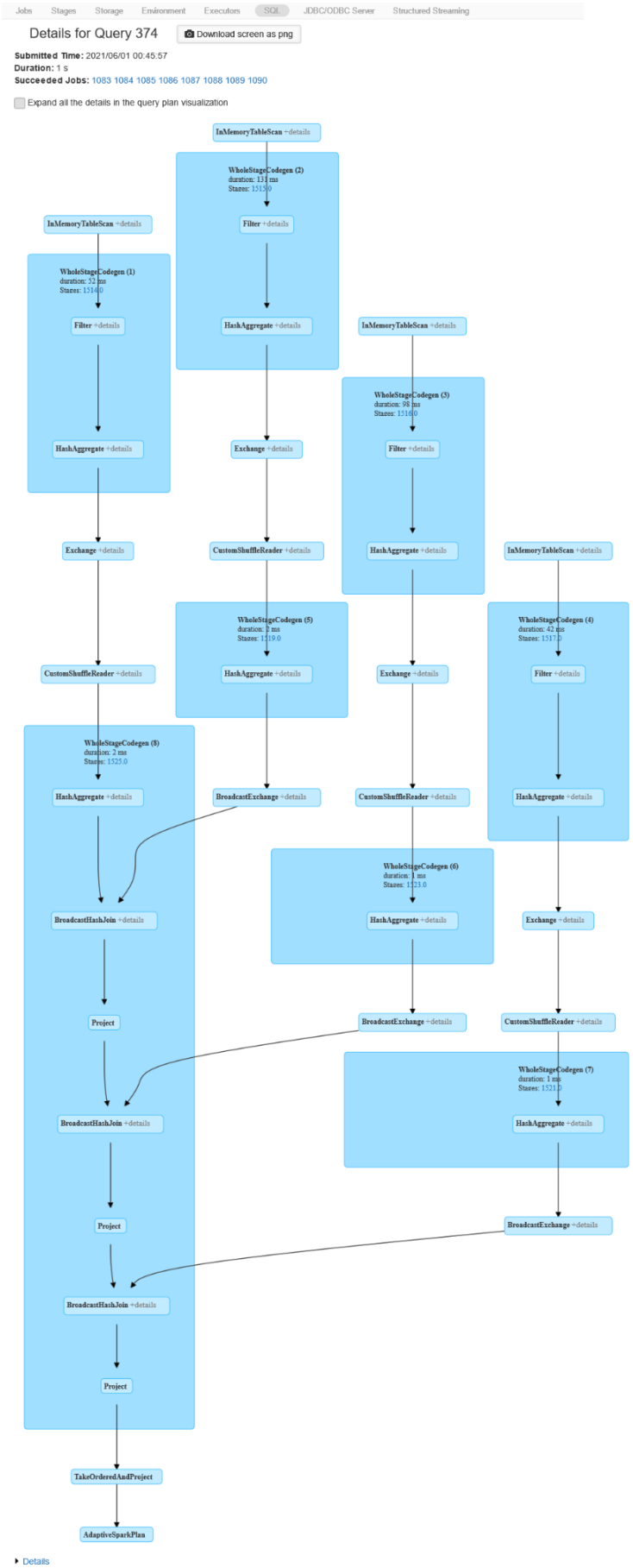
Unoptimized Data Frame: Workflow is similar across all data frame sizes.



Project Group: R10A-05

SID_1: 490580833 SID_2: 480556587 SID_3:490422382

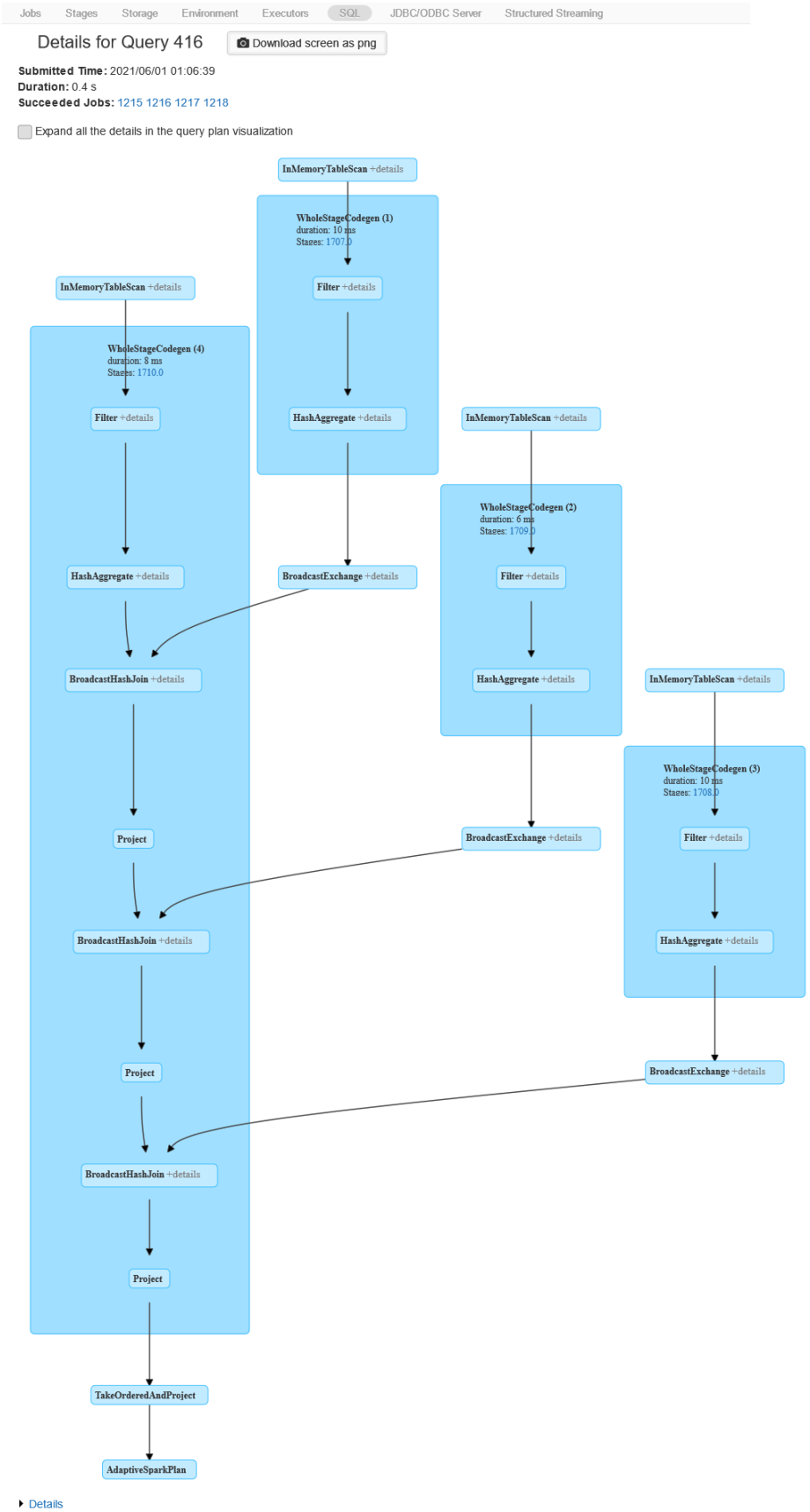
Optimized Data Frame (without repartition/coalesce): Workflow is similar across all data frame sizes.



Project Group: R10A-05

SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Optimized Data Frame (with repartition/coalesce): Workflow is similar across all data frame sizes.



Project Group: R10A-05
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Appendix: TASK 3

Task 3:

Results:

small dataset

airline_name	aircraft_type
AirTran	[BOEING 717, BOEING 737]
Alaska Airlines Inc.	[BOEING 737, MCDONNELL DOUGLAS 983, MCDONNELL DOUGLAS AIRCRAFT CO]
American Airlines Inc.	[MCDONNELL DOUGLAS 982, MCDONNELL DOUGLAS 983, AIRBUS INDUSTRIE 319, BOEING 757, BOEING 767]
American Eagle Airlines Inc.	[EMBRAER 145, EMBRAER 135, BOMBARDIER INC 600, SAAB-SCANIA 340]
Atlantic Southeast Airlines	[BOMBARDIER INC 600, EMBRAER 145, CANADAIR 600, EMBRAER 135, AEROSPATIALE/ALENIA 722]
Comair	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281]
Continental Air Lines Inc.	[BOEING 737, BOEING 757, BOEING 767, BOEING 777]
Delta Air Lines Inc.	[MCDONNELL DOUGLAS AIRCRAFT CO , BOEING 757, BOEING 737, BOEING 717, BOEING 767]
Frontier Airlines Inc.	[AIRBUS 319, AIRBUS 318, AIRBUS INDUSTRIE 319, AIRBUS 320]
Hawaiian Airlines Inc.	[BOEING 717, BOEING 767]
Independence Air	[BOMBARDIER INC 600]
JetBlue Airways	[AIRBUS 320, EMBRAER 190, AIRBUS INDUSTRIE 320]
JetSuite Air	[EMBRAER 145, EMBRAER 135]
Mesa Airlines Inc.	[BOMBARDIER INC 600, DEHAVILLAND 820, CANADAIR 600]
Northwest Airlines Inc.	[AIRBUS INDUSTRIE 320, BOEING 757, MCDONNELL DOUGLAS 951, DOUGLAS 931, AIRBUS 319]
PSA Airlines Inc.	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281]
Pinnacle Airlines Inc.	[BOMBARDIER INC 600]
Skywest Airlines Inc.	[BOMBARDIER INC 600, EMBRAER 120, CANADAIR 600]
Southwest Airlines Co.	[BOEING 737, BOEING 747, EMBRAER 120, CESSNA 337, BEECH]

medium dataset

airline_name	aircraft_type
AirTran	[BOEING 717, BOEING 737]
Alaska Airlines Inc.	[BOEING 737, MCDONNELL DOUGLAS 983, MCDONNELL DOUGLAS AIRCRAFT CO]
American Airlines Inc.	[MCDONNELL DOUGLAS 982, MCDONNELL DOUGLAS 983, AIRBUS INDUSTRIE 319, BOEING 757, BOEING 767]
American Eagle Airlines Inc.	[EMBRAER 145, EMBRAER 135, BOMBARDIER INC 600, SAAB-SCANIA 340]
Atlantic Southeast Airlines	[BOMBARDIER INC 600, EMBRAER 145, CANADAIR 600, EMBRAER 135, AEROSPATIALE/ALENIA 722]
Comair	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281]
Continental Air Lines Inc.	[BOEING 737, BOEING 757, BOEING 767, BOEING 777]
Delta Air Lines Inc.	[MCDONNELL DOUGLAS AIRCRAFT CO , BOEING 757, BOEING 737, BOEING 717, AIRBUS INDUSTRIE 320]
Frontier Airlines Inc.	[AIRBUS 319, AIRBUS 318, AIRBUS INDUSTRIE 319, AIRBUS 320]
Hawaiian Airlines Inc.	[BOEING 717, BOEING 767]
Independence Air	[BOMBARDIER INC 600]
JetBlue Airways	[AIRBUS 320, EMBRAER 190, AIRBUS INDUSTRIE 320]
JetSuite Air	[EMBRAER 145, EMBRAER 135]
Mesa Airlines Inc.	[BOMBARDIER INC 600, DEHAVILLAND 820, CANADAIR 600]
Northwest Airlines Inc.	[AIRBUS INDUSTRIE 320, BOEING 757, MCDONNELL DOUGLAS 951, DOUGLAS 931, AIRBUS INDUSTRIE 319]
PSA Airlines Inc.	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281]
Pinnacle Airlines Inc.	[BOMBARDIER INC 600]
Skywest Airlines Inc.	[BOMBARDIER INC 600, EMBRAER 120, CANADAIR 600]
Southwest Airlines Co.	[BOEING 737, BOEING 747, EMBRAER 120, CESSNA 337, BEECH]

large dataset

Project Group: R10A-05
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

airline_name	aircraft_type
AirTran	[BOEING 717, BOEING 737]
Alaska Airlines Inc.	[BOEING 737, MCDONNELL DOUGLAS 983, MCDONNELL DOUGLAS AIRCRAFT CO]
American Airlines Inc.	[MCDONNELL DOUGLAS 982, MCDONNELL DOUGLAS 983, AIRBUS INDUSTRIE 319, BOEING 757, BOEING 767]
American Eagle Airlines Inc.	[EMBRAER 145, EMBRAER 135, BOMBARDIER INC 600, SAAB-SCANIA 340]
Atlantic Southeast Airlines	[BOMBARDIER INC 600, EMBRAER 145, CANADAIR 600, EMBRAER 135, AEROSPATIALE/ALENIA 722]
Comair	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281]
Continental Air Lines Inc.	[BOEING 737, BOEING 757, BOEING 767, BOEING 777]
Delta Air Lines Inc.	[MCDONNELL DOUGLAS AIRCRAFT CO , BOEING 757, BOEING 737, BOEING 717, BOEING 767]
Frontier Airlines Inc.	[AIRBUS 319, AIRBUS 318, AIRBUS INDUSTRIE 319, AIRBUS 320]
Hawaiian Airlines Inc.	[BOEING 717, BOEING 767]
Independence Air	[BOMBARDIER INC 600]
JetBlue Airways	[AIRBUS 320, EMBRAER 190, AIRBUS INDUSTRIE 320]
JetSuite Air	[EMBRAER 145, EMBRAER 135]
Mesa Airlines Inc.	[BOMBARDIER INC 600, DEHAVILLAND 820, CANADAIR 600]
Northwest Airlines Inc.	[AIRBUS INDUSTRIE 320, BOEING 757, MCDONNELL DOUGLAS 951, DOUGLAS 931, AIRBUS 319]
PSA Airlines Inc.	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281]
Pinnacle Airlines Inc.	[BOMBARDIER INC 600]
Skywest Airlines Inc.	[BOMBARDIER INC 600, EMBRAER 120, CANADAIR 600]
Southwest Airlines Co.	[BOEING 737, BOEING 747, EMBRAER 120, CESSNA 337, BEECH]

massive dataset:

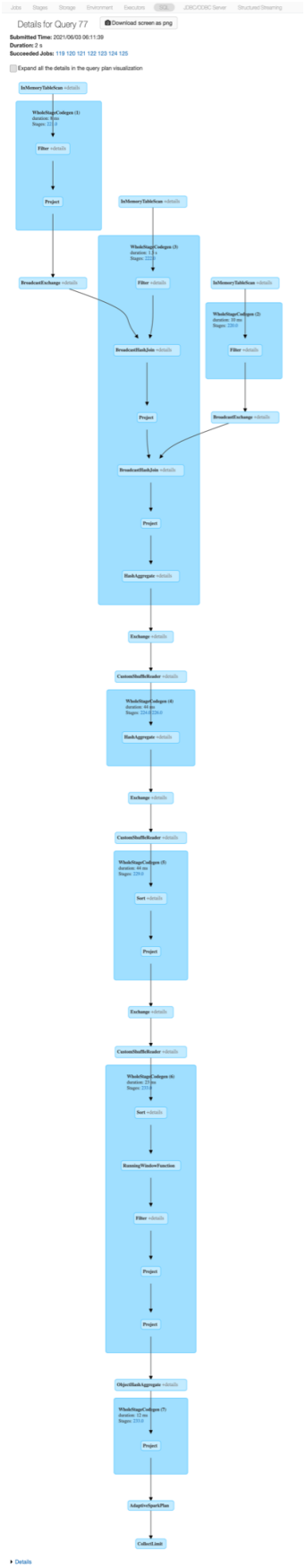
AirTran	[BOEING 717, BOEING 737]
Alaska Airlines Inc.	[BOEING 737, MCDONNELL DOUGLAS 983, MCDONNELL DOUGLAS AIRCRAFT CO]
American Airlines Inc.	[MCDONNELL DOUGLAS 982, MCDONNELL DOUGLAS 983, AIRBUS INDUSTRIE 319, BOEING 757, BOEING 767]
American Eagle Airlines Inc.	[EMBRAER 145, EMBRAER 135, BOMBARDIER INC 600, SAAB-SCANIA 340]
Atlantic Southeast Airlines	[BOMBARDIER INC 600, EMBRAER 145, CANADAIR 600, EMBRAER 135, AEROSPATIALE/ALENIA 722]
Comair	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281, EMBRAER 145]
Continental Air Lines Inc.	[BOEING 737, BOEING 757, BOEING 767, BOEING 777, EMBRAER 145]
Delta Air Lines Inc.	[MCDONNELL DOUGLAS AIRCRAFT CO , BOEING 757, BOEING 737, BOEING 717, AIRBUS INDUSTRIE 320]
Frontier Airlines Inc.	[AIRBUS 319, AIRBUS 318, AIRBUS INDUSTRIE 319, AIRBUS 320, BOEING 737]
Hawaiian Airlines Inc.	[BOEING 717, BOEING 767]
Independence Air	[BOMBARDIER INC 600]
JetBlue Airways	[AIRBUS 320, EMBRAER 190, AIRBUS INDUSTRIE 320]
JetSuite Air	[EMBRAER 145, EMBRAER 135]
Mesa Airlines Inc.	[BOMBARDIER INC 600, DEHAVILLAND 820, CANADAIR 600, BOEING OF CANADA LTD 810]
Northwest Airlines Inc.	[AIRBUS INDUSTRIE 320, BOEING 757, MCDONNELL DOUGLAS 951, DOUGLAS 931, AIRBUS 319]
PSA Airlines Inc.	[BOMBARDIER INC 600, CANADAIR 600, PIPER 281, EMBRAER 145]
Pinnacle Airlines Inc.	[BOMBARDIER INC 600]
Skywest Airlines Inc.	[BOMBARDIER INC 600, EMBRAER 120, CANADAIR 600, BOEING 737]
Southwest Airlines Co.	[BOEING 737, BOEING 747, EMBRAER 120, CESSNA 337, BEECH]
US Airways	[BOEING 737, AIRBUS INDUSTRIE 319, AIRBUS INDUSTRIE 320, AIRBUS INDUSTRIE 321, EMBRAER 190]
United Airlines	[BOEING 737, AIRBUS INDUSTRIE 320, BOEING 757, AIRBUS INDUSTRIE 319, BOEING 767]

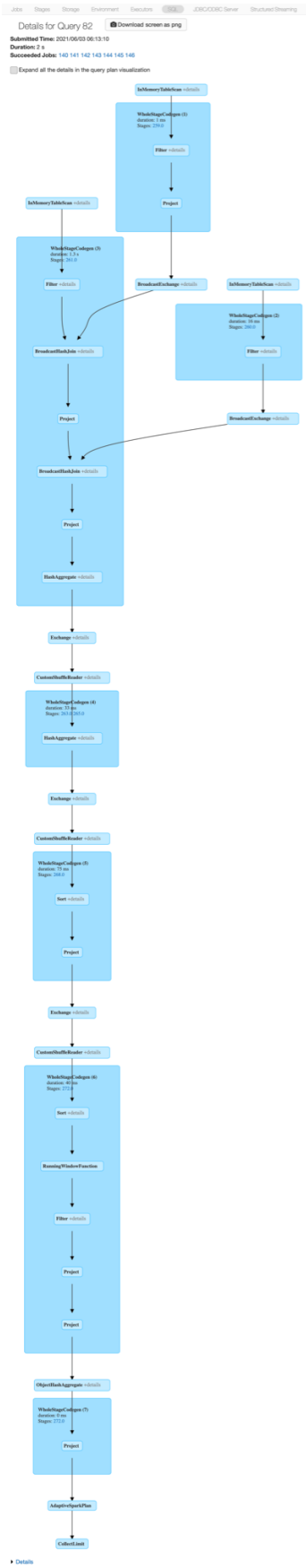
Project Group: R10A-05

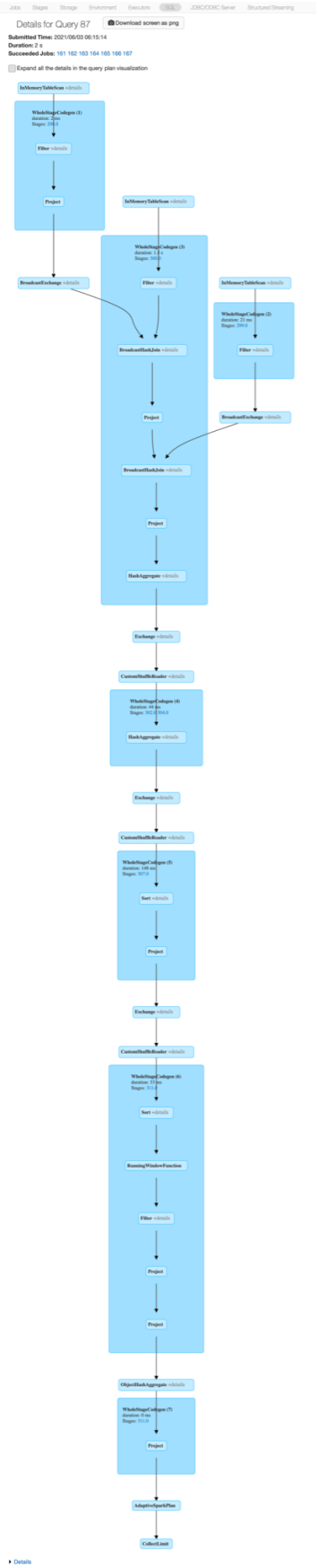
SID_1: 490580833 SID_2: 480556587 SID_3:490422382

Appendix: Task 3 execution plan

The Pre-Optimized Execution plan Optimized Execution plan(Broadcast) Optimized Execution plan(Cache)







Appendix

HDFS location of output files results/task1 or task2 or task3