THE UNIVERSITY OF SYDNEY

University of Sydney

DATA3404

Data Science Platforms

Group Assignment

Team members:
Charles Hyland 450411920
Yiran Jing 460244129
Jazlyn Lin 470345744

Semester 1, 2019

Contents

Job Design Documentation

Task 1: Top 3 Cessna Models

Aircrafts:(tail number, manufacturer, and model), *Flights*: (tail number).
Before:

1. Join two data files using the **join** function with tail number as the join key.
2. Apply **FilterFunction** to only return aircrafts with the manufacturer equaled to "CESSNA", and then project the "model" column only.
3. Apply a **flatMap** function with **CountFlightPerModel** object to produces one for each instance. Additionally, append "Cessna" to each instance and retrieve the first three digits of each instance to fit the final output format.
4. Then, group the data by the different Cessna models using the **groupBy** function, and for each group use the **sum** function to count all the instances of the same Cessna model.
5. Rank Cessna models in descending order using the **sortPartition** function and return the top 3 Cessna model by the **first** function.

After:

1. Apply **FilterFunction** to only return aircrafts with the manufacturer equaled to "CESSNA", and then use **project** function to project the tail number and model columns.
2. Join the two data files using **join** with **broadcast hash** and then filter for CESSNA mode. We then project the model column only.
3. Add **ReadFields** for the **CountFlightPerModel** object to specify the model field is used to compute a result value. After that, apply **flatMap** with the **CountFlightPerModel** object. We repeat the following steps from before.

Task 2 Average Departure Delay

Aircrafts: (tail number), *Flights* (carrier code, flight date, tail number, scheduled departure, actual departure), *Airlines*: (carrier code, name, country), year variable for user specified year.
Before:

1. First, we join the three dataset to get a merged dataset. We then project only five columns (flight date, scheduled departure, actual departure, name, country).
2. Then we filter the airline dataset by using the **FilterFunction** to contain only US Airlines, and then filter the specified year using 'flight date' field of Flights file. We then **project** three columns (scheduled deparature, actual departure, name)
3. Filter out non-delayed flights if actual departure time is not later than scheduled time, and filter out cancelled flights by catching ParseExceptions within the **FilterFunction**
4. Apply a **flatMap** with the **TimeDifferenceMapper** object to calculate the actual delay time for each delay departure flight.
5. Apply the **flatMap** function with the **NumMapper** object to produce one for each instance, then, group the data by the different US airlines using **groupBy** function. Then for each group, use **sum** function to count all the instances of the same US airlines to construct the *joinresultNum* dataset.
6. Then, group the *joinresult* dataset byUS airlines using the **groupBy** function, and use the **sum** function to get the total length of delay time for each US airline. After that, join this dataset with *joinresultNum* to get *joinresultNumSum* dataset.
7. Group *joinresult* data by US airlines using **groupBy** function, and use **min** to get the min length of delay time for each US airline. After that, join this dataset with *joinresultNumSum* to get *joinresultNumSumMin* dataset.

8. Group the *joinresult* data by US airlines using the **groupBy** function. Then use **max** to get the max length of delay time for each US airline. After that, join this dataset with *joinresultNumSumMin* to get *joinresultNumSumMinMax* dataset.
9. Apply the **flatMap** function with **AvgMapper** object to get the average delay time for each US airline. Then, rank US airlines in alphabetical order by the **sortPartition** function.

After:

1. At the beginning, we filter the airline dataset by the **FilterFunction** to contain only US Airlines, and then **project** only two fields, the carrier code and name columns.
2. Filter the specified year using the 'flight date' field of Flights file, and then remove this field.
3. Filter out non-delayed flights if actual departure time is not later than scheduled time, and filter out the cancelled flight by catch ParseException within **FilterFunction**
4. After that, apply a **flatMap** with **TimeDifferenceMapper** object to calculate the actual delay time for each delay departure flight.
5. Rank the US airlines in alphabetical order by **sortPartition** function before join.
6. We join two data files using the **join** function specified with **broadcast hash** on the aircrafts file and the filtered US airlines file. We then project only two columns (airline name, length of delay time)
7. We **groupBy** the US airline result, and instead of step 6 to 10, we apply **reduceMap** function with **Aggregation** function to count the number of delay and the average, min and max delay time for each US airline at the same time. We also added **ForwardedFields** and **ReadFields** to this object.

Task 3: Most Popular Aircraft Types

Aircrafts:(tail number, manufacturer, model),*Flights*: (carrier code, tail number), *Airlines*: (carrier code, name, country).

Before:

1. We join the airlines dataset on the flights dataset based on the carrier code. Furthermore, we restrict the output to only include the airline name and the flight tail number fields.
2. We join the output of step 2 with the aircrafts dataset based on the tail number. Furthermore, we restrict the output to only include the airline name, flight tail number, aircraft manufacturer, and airline model fields.
3. We apply a **groupBy** function on the result of step 1 by the flight tail number. We then apply a **reduceGroup** function whereby we count the number of unique flight tail numbers and construct a new field with a count for each tail number to append. We then sort the data by airline name and the tail number count constructed.
4. We apply a **reduceGroup** function whereby we retrieve the top 5 aircraft type for each airlines.
5. We filtered the airlines dataset for flights based in the United States.
6. We apply a **reduceGroup** function on the output of the previous step to format the result needed for the output and sort the output by the airline name alphabetically.

After:

1. We apply a **filter** function at the beginning of the program on the airlines dataset to only include US airlines and we then project only the carrier code and name field in the final output.
2. The steps are identical to before except we apply the airlines filter in step 5 to be the first step.
3. We apply a **broadcast hash join** and **ForwardedFieldFirst** function annotation in step 2 and 3 for reasons similar to the previous task.
4. One disclaimer is that the original program utilises a hashmap to keep track of frequency of each model for each airline. This is an issue as it is an unbounded memory data structure and hence can cause issues when

the dataset is too large. Hence, we have another program which simply iterates through the data once and keeps track of the top 5 aircraft types for each airlines. However, the first program ran significantly faster and did not have any memory issues.

Tuning Decisions and Justification

Tuning decisions made

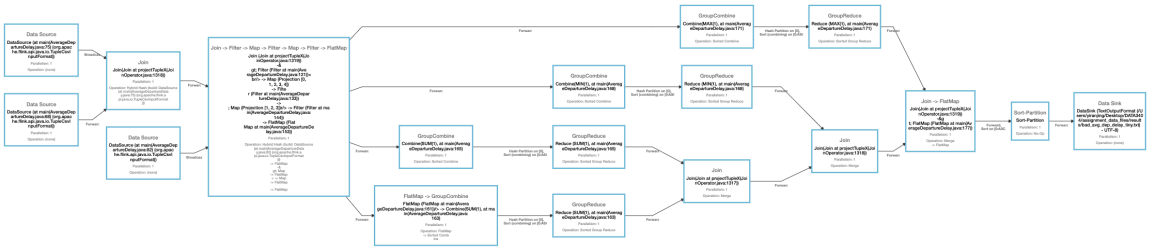
Task 1: Top-3 Cessna Models

1. **Filter and Project earlier**: In order to reduce the datatset size for later steps, we filter the dataset by Cessna manufacturer and then delete manufacturer column, project only join key and "model" at the beginning. In the following steps, we also filter out the join key immediately after join.
2. **Broad Cast Join First**: Since the filtered aircrafts file is much smaller than the flights file, broadcast hash join is used. The rationale is that a hash join will be executed whereby the data will be split into buckets, that are later merged. This achieves a speed up in run time in comparison to traditional loop joins.
3. **Add ReadFields Annotations**: Add ReadFields for the CountFlightPerModel object to specifies the "model" field is used to compute the result value.

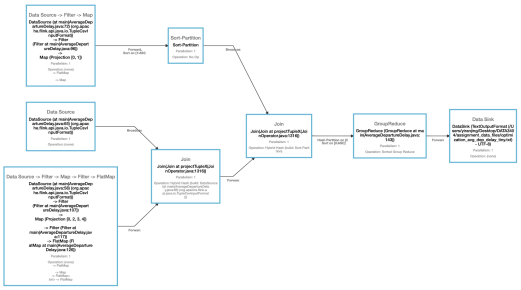
Task 2: Average Departure Delay

1. **Filter and Project earlier**: Similar with task 1, the UDF **TimeDifferenceMapper** reduces the fields of tuples and the size of dataset as early as possible. For example, before optimization, the **record received** for join is 2242390(tiny dataset), and output size **record send** of join is 1731618, both relative large compared to the post-optimized version: the **record send** is only 14753, and the **record receive** is 36449. This difference becomes much more significant using the larger and massive dataset. This is most likely the lead reason why there was a much shorter average execution time in the post-optimize version of the program.
2. **Broad Cast Join First**: Since the filtered USAirlines and aircrafts files are much smaller than the flights file, broadcast hash join is used.
3. **Efficient User Defined Function**: Using aggregation function to calculate and get average, min, max, count at the same time, this avoids having to do 4 joins as seen in the pre-optimization program. This UDF also avoids heavy computational cost.
4. **ReadFields and ForwardedFields**: These function annotation helps the Flink optimiser with the optimisation routine. Adding ReadFields for the "scheduled departure time" and "actual departure time" field to specify the "model" field that is used to compute the result value. Using ForwardedFields to let Flink know that the "carrier code" and "tail number" are copied without any changes.

Pre-Optimized Execution plan for Task 2



Optimized Execution plan for Task 2

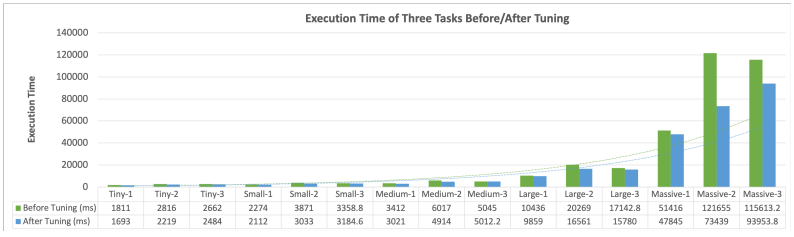


As shown in two execution plans, the change is very obvious.

Task 3: Most Popular Aircraft Types

1. **Filter** Applying a filter for only US airlines at the beginning of the program benefits the program in 2 ways. First, each operation will now carry one less field, the country field, throughout the program as it has already served its purpose at the beginning of the program for filtering out airlines. Second, the number of tuples the program has to carry out operations on will greatly decrease as we have filtered out the non-US airlines.
2. **Projection** By projecting tuples to only include relevant fields, this reduces the number of fields the program has to apply operations onto.
3. **Efficient User Defined Function** Same rationale as task 2 whereby UDF implemented for ranking and retrieval of the five most used aircrafts saves dramatic runtime. We find that read can help to improve the performance in the massive dataset.
4. **Broadcast Hash Join and Function Annotations**:The benefits are similar as seen in task 1 and task 2.

Performance improvement of three tasks

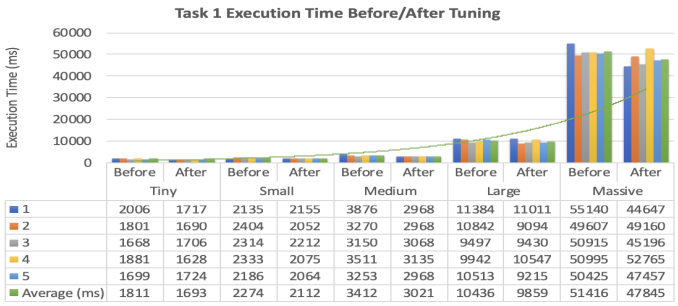


	Tiny-1	Tiny-2	Tiny-3	Small-1	Small-2	Small-3	Medium-1	Medium-2	Medium-3	Large-1	Large-2	Large-3	Massive-1	Massive-2	Massive-3
Before Tuning (ms)	1811	2816	2662	2274	3871	3358.8	3412	6017	5045	10436	20269	17142.8	51416	121655	115613.2
After Tuning (ms)	1693	2219	2484	2112	3033	3184.6	3021	4914	5012.2	9859	16561	15780	47845	73439	93953.8

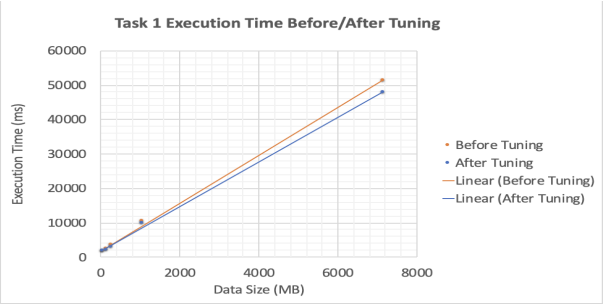
Performance Evaluation

The performance of each program before and after tuning was evaluated based on the average result of 5 executions.

Task 1: Top-3 Cessna Models

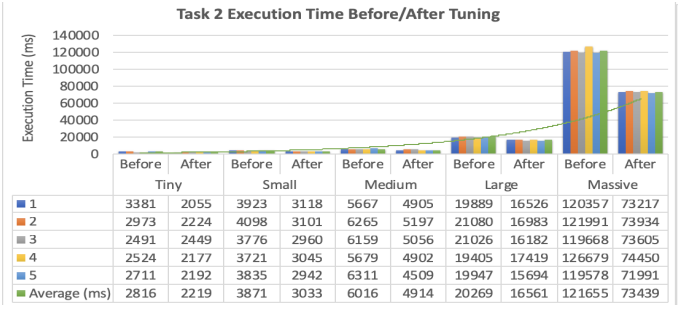


	Before	After	Before	After	Before	After	Before	After	Before	After
	Tiny		Small		Medium		Large		Massive	
1	2006	1717	2135	2155	3876	2968	11384	11011	55140	44647
2	1801	1690	2404	2052	3270	2968	10842	9094	49607	49160
3	1668	1706	2314	2212	3150	3068	9497	9430	50915	45196
4	1881	1628	2333	2075	3511	3135	9942	10547	50995	52765
5	1699	1724	2186	2064	3253	2968	10513	9215	50425	47457
Average (ms)	1811	1693	2274	2112	3412	3021	10436	9859	51416	47845

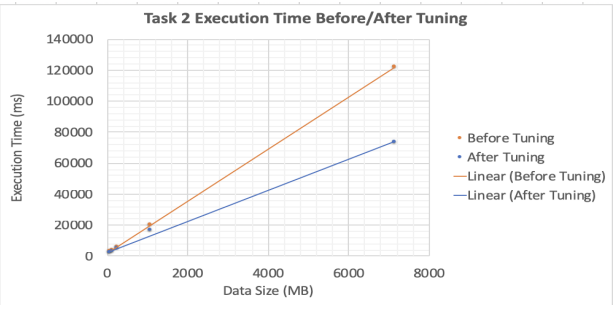


It can be seen from the left figure that overall the execution time grows almost exponentially when we increase the data set size from tiny to massive and there is a huge increase of execution time when we switched from large to massive data set. There is also a slight decrease of execution time after tuning which indicates the tuning was effective but not very noticeable for task 1. In the right figure, there is overall linear relationship between data size and execution time when the data set size was measured more accurately in Megabytes. The small gap between two linear lines before and after tuning also confirms the non-noticeable tuning effect for this task.

Task 2: Average Departure Delay

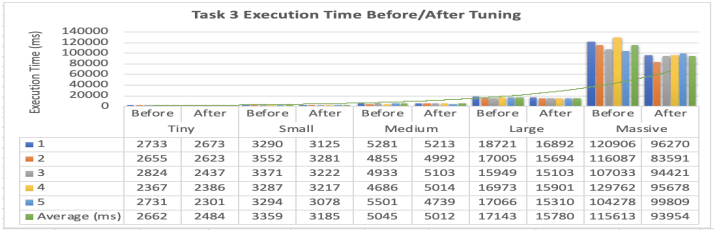


	Before	After	Before	After	Before	After	Before	After	Before	After
	Tiny		Small		Medium		Large		Massive	
1	3381	2055	3923	3118	5667	4905	19889	16526	120357	73217
2	2973	2224	4098	3101	6265	5197	21080	16983	121991	73934
3	2491	2449	3776	2960	6159	5056	21026	16182	119668	73605
4	2524	2177	3721	3045	5679	4902	19405	17419	126679	74450
5	2711	2192	3835	2942	6311	4509	19947	15694	119578	71991
Average (ms)	2816	2219	3871	3033	6016	4914	20269	16561	121655	73439

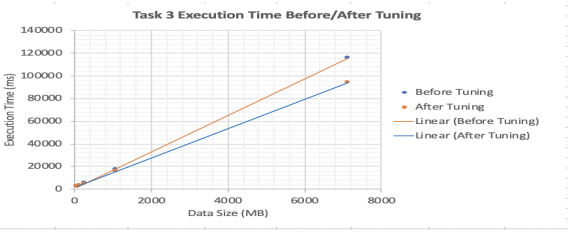


In Task 2, there is a similar exponential growth and linear growth in both left and right figures. However, the tuning effect was much more effective here compared to Task 1 and there is almost a 40 percent decrease in execution time for massive data set, which is consistent with the huge gap between two linear lines.

Task 3: Most Popular Aircraft Types



	Before	After	Before	After	Before	After	Before	After	Before	After
	Tiny		Small		Medium		Large		Massive	
1	2733	2673	3290	3125	5281	5213	18721	16892	120906	96270
2	2655	2623	3552	3281	4855	4992	17005	15694	116087	83591
3	2824	2437	3371	3222	4933	5103	15949	15103	107033	94421
4	2367	2386	3287	3217	4686	5014	16973	15901	129762	95678
5	2731	2301	3294	3078	5501	4739	17066	15310	104278	99809
Average (ms)	2662	2484	3359	3185	5045	5012	17143	15780	115613	93954



In Task 3, both exponential growth and linear growth are similar with previous tasks. Although the tuning effect less noticeable compared to last task, there is still around a 20-percent decrease in execution time for massive data set.

Appendix

HDFS location of output files

/user/jlin0701/results/