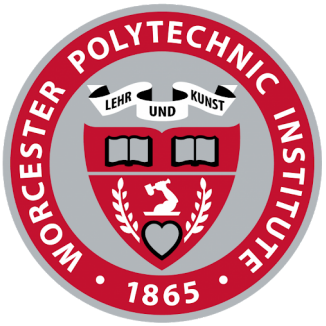


WPI

DS503

BIG DATA MANAGEMENT

Final Project: Join Over Skewed Dataset

Professor: Mohammed Eltabakh

Submitted by: Kratika Agrawal & Naisargi Dave

INTRODUCTION

The focus of our project is to join two big datasets where one of the datasets is skewed over a few keys. The Joining of two large datasets can be achieved with the use of MapReduce technique. Several frameworks like Hadoop or Spark allow us to perform the join by Mapping records key by key and pass all records that have the same join key to single Reducer after it passes through the shuffle and sort phase. All Reducers work in parallel to process the join and save their partitions on HDFS. This type of join is called the Re-partition join. In a scenario, when one of the two datasets are small (fewer MB), the task can be performed in Mappers only without involving Reducers and saving computation over expensive Shuffle and Sort modules. This type of join is called Broadcast join or Replicated join. In Broadcast join, the smaller dataset is cached and replicated to all Mappers. The join is performed internally in mappers by setting up lookup for every record of the large dataset to the entire smaller datafile. However, when both datasets are huge and one of them is skewed over few keys, Re-partition join leads to load imbalance on Reducers. Load imbalance on Reducer occurs because one Reducer gets a skewed key and hence, has to process a lot of records. This Reducer is working for a very long time. Whereas, all the other reducers that receive non-skewed keys process records quickly and stay idle. This results in slower computation.

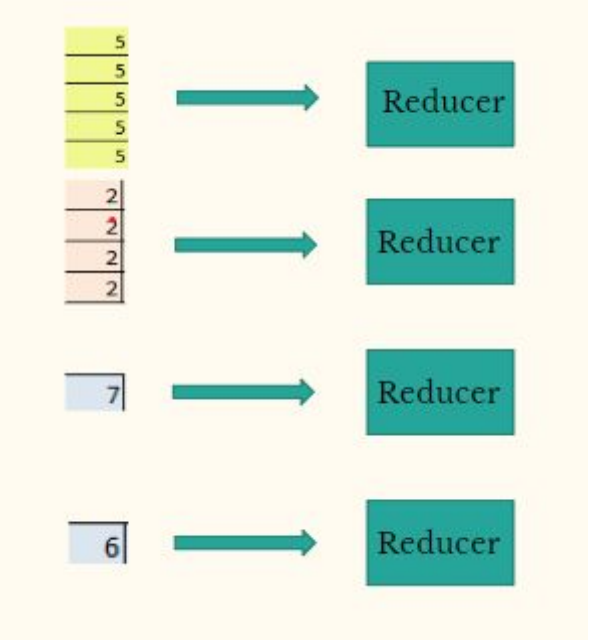


Figure 1 : Load Imbalance on Reducer

We try to solve this problem by scanning the data and identifying skewed keys to divide the join operation for skewed keys and non-skewed keys.

RELATED WORK

This problem has previously been solved by Nadeem Moidu in 2015 using the Hive framework. The solution is to perform optimized join over two structured datasets where one of the collections contain a few number of skewed records over the join key. They keep following assumptions:

- ➢ It is known that one of the dataset is skewed.
- ➢ All the join keys over which the data is skewed are already known.
- ➢ There is skewness on a few keys only.

ASSUMPTIONS

We inherit our solution from the solution provided by Moidu. However, we relax the assumption that we already know those keys on which the dataset is skewed. We scan the skewed dataset and identify all join keys on which the skewness exists. Along with this, since we make use of Spark MapReduce technique to implement optimized join, the solution holds good for unstructured or semi-structured datasets as well.

DATASET

For simplicity, we used two datasets of Customers and their Transactions. Customer data consists of approximately 1M unique records with information like CustomerID, CustomerName, CustomerAge, Gender, CountryCode and Salary for each customer. Transaction data is huge with approximately 13M transactions which includes TransactionID, CustomerID, TransactionTotal, TransactionNumItems, TransactionDescription. Transactions are skewed over few customerIds (Customer Ids: 7426, 5, 98775, 384, 67452, 2486 in our case). Also, we don't hold the assumption that we have prior knowledge of skewed keys. We scan transaction dataset to identify skewness.

CUSTOMERS

CustID	Name	Age	Gender	CountryCode	Salary
1	cokqixljgnfxfamfqo	10	female	2	4603.926
2	zjaablxjpi	30	female	1	728.2957
3	wbonikejguvtyp	52	male	4	8380.577
4	vvkcrxdsynpcmtpawdqj	35	male	8	111.025024
5	pwfpgqhrrlzf	25	male	9	144.55263
6	ohxlxufjjylzmpbw	22	female	8	2431.2893
7	wfbavpakhsmvyu	65	male	7	3123.74
8	sxpqzuhino	20	male	10	416.3094
9	ingyjigoixxiagvodub	64	female	2	2729.313
10	gvoxxbxafuqrqhhybda	17	female	9	431.1552

Figure 2: Customer Dataset

TRANSACTIONS

TransID	CustID	TransTotal	TransNumItems	TransDesc
1	4	369.91498	8	uejkyfdgsfkuseakehacvookmamopzyefesxuskpps
2	8	683.84827	6	yxafouxmvpyjpsysipwogex
3	6	363.17325	2	dgyorciaslygasarbeholpvzvkuqefnznd
4	5	128.83055	7	wkzyoyhryznqqaixetieezkcdqmmtfsqkwpakucve
5	5	428.39865	4	meozbpopxcgahhnbzklepdozpjlcmnroeanhmqtcalntlcptcd
6	5	150.15614	1	nazcvarrxmijlswymkixtgxfoy
7	5	146.23466	8	dyvoquytpdilspqoczgtjcvulsrunsz
8	5	57.34631	3	tzrzfldehuefnkefkitulohxfgxnzmfgijvdncqmlnbeqck
9	5	102.65818	6	hzdewekzovcnstrehfzhaiomcklygghsmgdhmfivm
10	5	27.884157	8	wifmqmwutdcmhzhshksakvvakpqbllflwormmbnznky
11	5	413.52927	7	vduxfmzwbkledlnihxgeyfmuphognqsmtlb
12	5	832.4867	2	oxdnlohjozrfolhkzjpmqcwyvcfzpgunpipjmrqhdetwxgj
13	7	55.506687	10	fjdoizwrnftjcqozbhvkgkumipxexoamwsnqvolbiyxoczgp
14	1	16.66314	7	gkvuutxhpcveopojujpykbg
15	2	258.46747	8	ojwhzjzopgxuoqokyvkqretzctljxxvrtrapatrtelljza
16	2	9.309454	7	vywtbfobhnpyxfjgjbrwiguhdygitvyovxkqnevqhwkixqaa
17	2	229.03754	9	hnxcrhzyzmymtcngqtffzygflchwkswngmqrggpqbro
18	2	269.31647	1	wzbtyhqmlefvmcqkuxmziwygireh
19	2	468.29593	3	toypiexvvsvydrigvehetacjujs
20	2	532.0054	4	foecmmypjnpghztuugbmvwa
21	7	411.59137	10	srqmelowktahaoffwcegapliwcmoz
22	5	479.8433	6	lzhgkycvznwthaasoleap
23	3	385.2021	2	tzczfjmnqbtpeunjpdfzamcvvalssiqsdamajhya
24	5	59.818882	7	hwoalhlpzketmpwlwgrgtukjkufbhkfidqyqbkntnznwtcqul

Figure 3: Transactions Dataset

TECHNICAL DETAILS

File JoinOverSkewedDataset.scala consists of code for implementing the join with-
Argument[0] : Customers.csv file path generated using CreateDataset.java
Argument[1] : Transactions.csv file path generated using CreateDataset.java

The task of joining Customers and Transactions dataset, where Transactions are skewed on few Customer Ids, i.e. some of the Customers have a very large number of transactions while rest have few transactions. The join has been implemented using Spark framework in Scala language using SparkSQL DataFrames. The process is executed in following steps:

Step 1- Import Customers and Transactions datasets into DataFrames:

Use SparkSQL to define schema and import the Customers and Transactions datasets into respective DataFrames for further processing.

Step 2- Identify the join keys over which the transactions table is skewed:

The idea of Transactions being skewed over some join keys is that some of the customers are processing much more transactions than others. It's almost impossible to scan huge Transaction data manually and exactly identify which all keys are more frequent. Therefore, if Transaction is highly skewed on some customer keys in large dataset and we attempt to randomly choose a Transaction record, the probability of it belonging to skewed customer key is higher.

We use the same mathematical concept, and take 10% random record samples of transactions, group by keys and count the number of transactions belonging to each join key. This gives us a clear idea of the skewness and we can easily gather all keys that cause it.

CustID	count
7426	59886
5	56789
98775	45953
384	43815
67452	35743
2486	31895
6587	22
797	22
85	22
952	21
6	21
45325	19
77	19
671	19
15	19
6160	16
773	16

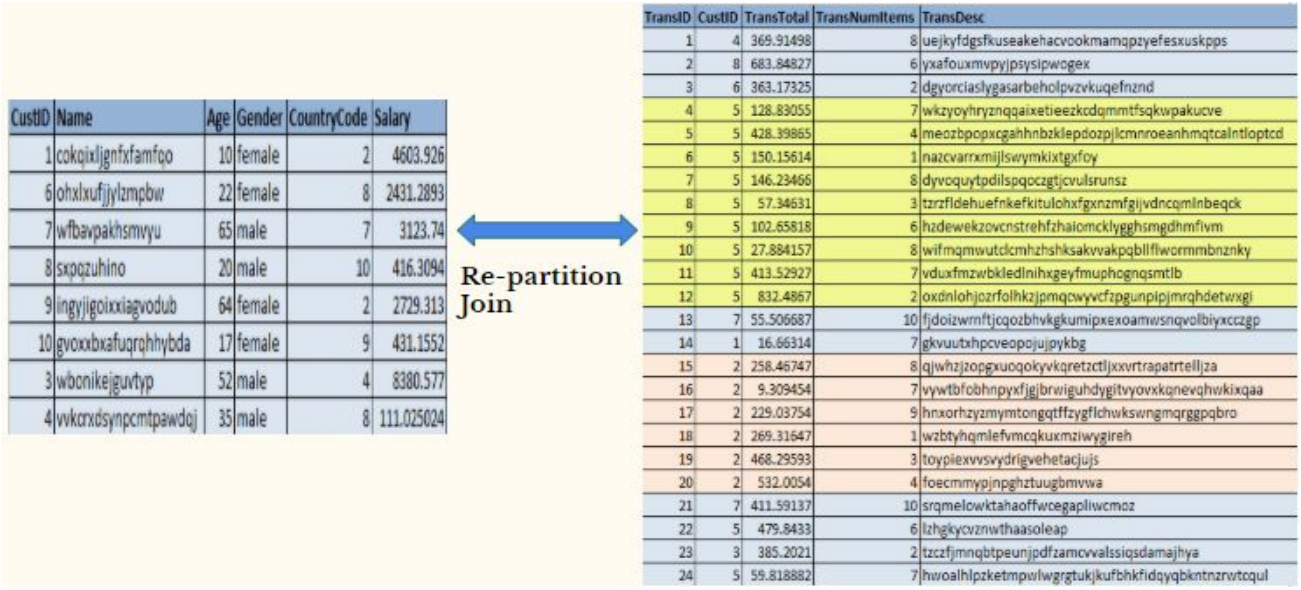
Figure 4: List of Join keys in descending order of their number of transactions

Step 3- Split Customer Data table:

Once we identify ids on which Transactions are skewed, Split the Customer dataset into two separate DataFrames: Skewed_Customers and Non-Skewed_Customers. Skewed_Customer contains all customer ids that cause skewness in transactions and Non-Skewed Customer contains the rest of the customers.

Step 4- Re-partition join between Non-Skewed Customers and Transactions:

Implemented Re-partition join between Non-Skewed Customers and Transactions where Customer Ids match. The join will only select Transactions that belong to non-skewed Customer Ids. All the Ids belonging to one join key, go to one Reducer after the sort or shuffle phase. Since, there's no skewness in the data and all keys have similar frequencies, all Reducers receive the same data load for processing. This process optimizes the joining time for the non-skewed data.



CustID	Name	Age	Gender	CountryCode	Salary
1	cokcjixljgnfxfamfqo	10	female	2	4603.926
6	ohxlxufjjylzmpbw	22	female	8	2431.2893
7	wfbavpakhsmvyu	65	male	7	3123.74
8	sxpqzuhino	20	male	10	416.3094
9	ingyjigoixxiagvodub	64	female	2	2729.313
10	gvoxxbxafuqrqhhybda	17	female	9	431.1552
3	wbcnikejguvtyp	52	male	4	8380.577
4	vvkcrxdsynpcmtpawdqj	35	male	8	111.025024

TransID	CustID	TransTotal	TransNumItems	TransDesc
1	4	369.91498	8	uejkyfdgsfkuseakehacvookmamqpzyefesxuskpps
2	8	683.84827	6	yxafouxmvpyjpsysipwogex
3	6	363.17325	2	dgyorciaslygasarbeholpvzvkuqefnznd
4	5	128.83055	7	wkzyoyhryznqqaixetieezkcdqmmtfsqkwpakucve
5	5	428.39865	4	meozbpopxcgahhnbzklepdozpjlcmnroeanhmqtcalntloptcd
6	5	150.15614	1	nazcvarrxmijlswymkixtgxfoy
7	5	146.23466	8	dyvoquytpdilspqoczgtjcvulsrunsz
8	5	57.34631	3	tzrzfldehuefnkefkitulohxfgxnzmfgijvdncqmlnbeqck
9	5	102.65818	6	hzdewekzovcnstrehfzhaiomcklygghsmgdhmfivm
10	5	27.884157	8	wifmqmwutclcmhzhshksakvvakpqbllflwormmbnznky
11	5	413.52927	7	vduxfmzwbkledlnihxgeyfmuphoqnqsmtlb
12	5	832.4867	2	oxdnlohjozrfolhkzjpmqcwyvcfzpgunpipjmrqhdetwxgi
13	7	55.506687	10	fjdoizwrnftjcqozbhvkgkumipxexoamwsnqvolbiyxcczgp
14	1	16.66314	7	gkvuutxhpcveopojujpykbg
15	2	258.46747	8	qjwhzjzopgxuoqokyvkqretzctljxxvrtrapatrtelljza
16	2	9.309454	7	vywtbfobhnpyxfjgjbrwiguhdygitvyovxkqnevqhwkixqaa
17	2	229.03754	9	hnxorhzyzmymtongqtffzygflchwkswngmqrggpqbro
18	2	269.31647	1	wzbtyhqmlefvmcqkuxmziwygireh
19	2	468.29593	3	toypiexvvsvydrigvehetacjujs
20	2	532.0054	4	foecmmypjnpghztuugbmvwa
21	7	411.59137	10	srqmelowktahaoffwcegapliwcmoz
22	5	479.8433	6	lzhgkycvznwthaasoleap
23	3	385.2021	2	tzczfjmnqbtpeunjpdfzamcvvalssiqsdamajhya
24	5	59.818882	7	hwoalhlpzketmpwlwgrgtukjkufbhkfidqyqbkntnzrwtcqul

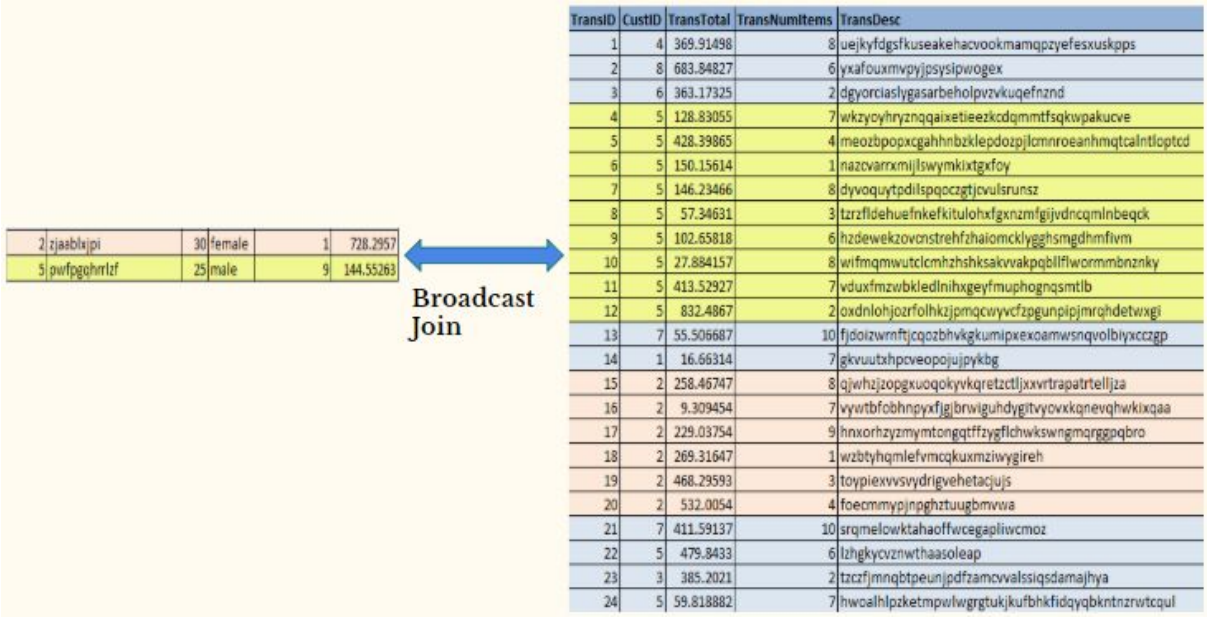
Figure 5: Repartition Join between Non-Skewed_Customers and Transactions

Step 5- Broadcast join between Skewed Customers and Transactions:

We keep an assumption that the skewness is over a small number of keys. Thus, it's easy to cache Skewed Customer DataFrame and broadcast it to all Mappers. Performed Broadcast join between Skewed_Customers and Transactions, which saved expensive computation of Sort, Shuffle and Reducer phase. Thereby, drastically reducing the processing time.

2	zjaablxjpi	30	female	1	728.2957
5	pwfpgqhrrlzf	25	male	9	144.55263

Figure 6 : Customers data that is cached



2	zjaablxjpi	30	female	1	728.2957
5	pwfpgqhrrlzf	25	male	9	144.55263

TransID	CustID	TransTotal	TransNumItems	TransDesc
1	4	369.91498	8	uejkyfdgsfkuseakehacvookmamqpzyefesxuskpps
2	8	683.84827	6	yxafouxmvpyjpsysipwogex
3	6	363.17325	2	dgyorciaslygasarbeholpvzvkuqefnznd
4	5	128.83055	7	wkzyoyhryznqqaixetieezkcdqmmtfsqkwpakucve
5	5	428.39865	4	meozbpopxcgahhnbzklepdozpjlcmnroeanhmqtcalntloptcd
6	5	150.15614	1	nazcvarrxmijlswymkixtgxfoy
7	5	146.23466	8	dyvoquytpdilspqoczgtjcvulsrunsz
8	5	57.34631	3	tzrzfldehuefnkefkitulohxfgxnzmfgijvdncqmlnbeqck
9	5	102.65818	6	hzdewekzovcnstrehfzhaiomcklygghsmgdhmfivm
10	5	27.884157	8	wifmqmwutclcmhzhshksakvvakpqbllflwormmbnznky
11	5	413.52927	7	vduxfmzwbkledlnihxgeyfmuphoqnqsmtlb
12	5	832.4867	2	oxdnlohjozrfolhkzjpmqcwyvcfzpgunpipjmrqhdetwxgi
13	7	55.506687	10	fjdoizwrnftjcqozbhvkgkumipxexoamwsnqvolbiyxcczgp
14	1	16.66314	7	gkvuutxhpcveopojujpykbg
15	2	258.46747	8	qjwhzjzopgxuoqokyvkqretzctljxxvrtrapatrtelljza
16	2	9.309454	7	vywtbfobhnpyxfjgjbrwiguhdygitvyovxkqnevqhwkixqaa
17	2	229.03754	9	hnxorhzyzmymtongqtffzygflchwkswngmqrggpqbro
18	2	269.31647	1	wzbtyhqmlefvmcqkuxmziwygireh
19	2	468.29593	3	toypiexvvsvydrigvehetacjujs
20	2	532.0054	4	foecmmypjnpghztuugbmvwa
21	7	411.59137	10	srqmelowktahaoffwcegapliwcmoz
22	5	479.8433	6	lzhgkycvznwthaasoleap
23	3	385.2021	2	tzczfjmnqbtpeunjpdfzamcvvalssiqsdamajhya
24	5	59.818882	7	hwoalhlpzketmpwlwgrgtukjkufbhkfidqyqbkntnzrwtcqul

Figure 7: Broadcast Join between Skewed_Customer and Transactions

BENEFITS:

- **Small number of skewed keys do not become bottleneck during the Reduce phase.** As we saw in Re-Partition Join for skewed dataset, reducer load imbalance takes place which acts as a bottleneck for the entire process. The algorithm implemented in this report prevents that from happening.
- **Computation is done faster.** Converting a part of the Re-Partition Join over skewed dataset to a Broadcast Join, which is a map-only job, makes it a faster process.
- **Efficient use of Reducers.** Because the load is distributed uniformly over all the Reducers, they are utilized to their maximum potential.

EXPERIMENTS:

Experiment I :

Initially we created a Customers dataset with 50,000 records and Transactions dataset with 5M records. We skewed the Transactions Dataset on one key and performed the experiment. We found out that the time difference between the two methods was not significant. We then moved on to increase the dataset size.

Experiment II :

The Customers dataset was made of 1M records and Transactions dataset was made of 13M records. Transactions dataset was skewed over key (Customer Ids): 5, 384, 2486, 7426, 67452, 98775. We performed the Re-partition join between Customers table and Transaction table on Customer Id. The process is run on Spark Map Reduce framework and takes a total of 10.236831 seconds.

The time taken to perform the join by implementing the optimization is less. The Broadcast Join between the Skewed_Customers and Transactions takes 0.019531 seconds. The Re-Partition Join between the Non-Skewed_Customers and Transactions takes 3.772760 seconds. So, the total time taken to perform the join is 3.792291 seconds which is much lesser compared to the initial 10.236831 seconds.

With these experiments, we see that the optimization saved approximately 60% of the time for computation, eliminating the imbalance from Reducers.

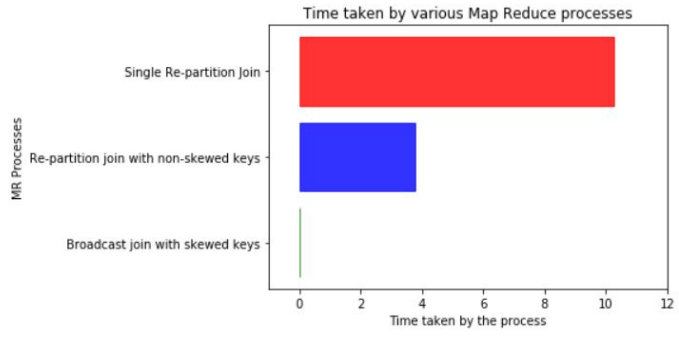


Figure 8: Time taken by various Map Reduce jobs