COMP 6521

Project 2: Frequent Itemset Mining in MapReduce

We certify that this submission is our original work and meets the Faculty's Expectations of Originality

Submitted by:

Name	**ID**	**Signature**
Zaki Chammaa	**26584233**	
Marzieh Montazeri	**26714498**	

<u>Introduction</u>

In this project, we were tasked to implement an algorithm to mine frequent itemsets from market basket data. A requirement for this project was to develop this algorithm using Hadoop and MapReduce in Java. Ultimately, it was important to get the correct result as well as have an optimized algorithm.

<u>Algorithm</u>

For this project, we have implemented the SON MapReduce algorithm in Java. There were 3 major steps in our procedure:

1. Input preprocessing
2. MapReduce phase 1
3. MapReduce phase 2

<u>Input Preprocessing</u>

For this part, the input file is taken and split into multiple input files depending on the memory requirements. This is done so as to create as many mappers as there are input files and therefore speed up the phase 1 mapping process. This part was a little tricky because although it is quite simple to read a file and split it to sub-files with a specific size, it was hard not to cut a basket into parts that would be into files. Nevertheless, we were able to do that without cutting any baskets and have files of equal size (save the last one).

<u>MapReduce Phase 1:</u>

In this step, the first mapper class is called and proceeds to reading the inputs in the input folder. Since we have k mappers (and k input files), we divided the global support threshold by k so as to get local_treshold to be used by each mapper. Also, since the mapper in Java reads the input line by line, we were forced to collect all the input in a list in order to be able to run our Apriori algorithm on the sub-file. After we collected the baskets from the sub-file, we fed it to the Apriori algorithm to get the frequent itemsets within the sub-file given the local threshold.

The result of the mapper is then fed to the reducer which simply aggregates the result and outputs them to a file. The value here isn't important as we know that all the itemsets found in the output of the reducer as local frequent itemsets.

To sum up phase 1, we had k sub-files and k-mappers. Each mapper reads from a sub-file and feeds the baskets to the Apriori algorithm to find local frequent itemsets, which are then fed to the reducer to be outputted to an output file.

MapReduce Phase 2

For phase 2, we had to load the frequent itemsets found in phase 1 into memory. The input for the mapper was the market baskets provided for the project. For each mapper iteration, we check each candidate on the basket to see if the candidate is part of the basket. If it is, the candidate is sent to the reducer.

The reducer in this phase takes the output of the mapper and aggregates the results. If the count for a candidate is equal to or greater than the global support threshold, then it is a frequent itemset and is outputted to final output along with its frequency.

Apriori Algorithm

Apriori Algorithm is one of the classic algorithm used to find association rules. For this project, a $F_{k-1} * F_{k-1}$ method Apriori algorithm was used in the phase 1 mapper.

Pass 1: Scan the Transactions and calculate frequents of all items in the main memory. In this step, the itemsets contain just one single item. The frequency of candidate itemsets is calculated by using a Hashmap. After finding all the candidate items and their support, the next step is to remove the infrequent items that have a smaller support value than the value of the minimum support. This result is saved as it is needed for the subsequent passes.

Pass 2: Since our algorithm is based on $F_{k-1} * F_{k-1}$ method, in this pass we generated the candidates by merge a pair of frequent (k-1)-itemsets which has been found in pass one

only if their first k-2 items are identical. Similar to the first pass, the frequency of the pairs of items is counted by Hashmap. After finding all candidates we need a candidate pruning step in order to compare their value with the minimum support and eliminate those pairs which have a support smaller than minimum support. We keep repeating this step k-times until there are no more frequent itemsets.
Apriori algorithm could be very slow when the size of the file increases. The reason is that it needs to generate candidates in each step, scan the whole file to count the frequency of each candidate, and compare the frequent numbers of each candidate with the minimum support to eliminate infrequent items. This algorithm also consumes a lot of memory. On the other hand, the Apriori algorithm is easy to be implemented and can be improved by using various methods like using Hash-based itemset counting.

Discussion

Our implementation of the frequent itemset mining algorithm with mapreduce works very well and produces the desired output. Along the way, there were some optimizations that were done, such as splitting the input file into smaller sub-files so as to speed up the mapping process and consume less memory when running the Apriori algorithm. We can also specify the number of reducers in phase 1 to also consume less memory during the process.

Contribution

Zaki Chammaa

For this project, my main responsibilities were to research and implement the mapreduce algorithm along with the all the preprocessing that came with it. Also, I had to find ways to optimize the program as it can get quite computationally heavy. I wrote parts of the report which apply to my work.

Marzieh Montazeri

For this project, i did research about mapreduce algorithm and helped a bit in the developing part of it. Moreover, I implemented and adapted the Apriori algorithm according to F_{k-1} * F_{k-1} method. I did research about how to modify the java heap size using eclipse. I wrote the parts of the report which applies to my work.