K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: BDA2 Roll No.: 1211061

Experiment / assignment / tutorial No.___4___

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Implementation of any one Frequent Itemset algorithm using Map Reduce.

AIM : Implementation of any one Frequent Itemset algorithm using Map Reduce.

Expected Outcome of Experiment :

CO : Interpret business modes and scientific computing paradigm and apply software tools for Big data analytics.

Books/ Journals/ Websites referred:

1. Anand Rajaraman and Jeff Ullman "Mining of Massive Datasets", Cambridge University Press,
2. Alex Holmes "Hadoop in Practice", Manning Press, Dreamtech Press.
3. Big data analytics by Radha Shankarmani, M. Vijayalakshmi. Wiley publication

---

Pre Lab/ Prior Concepts:

Frequent sets play an essential role in many Data Mining tasks that try to find interesting patterns from databases, such as association rules, correlations, sequences, episodes, classifiers and clusters. The mining of association rules is one of the most popular problems of all these. The identification of sets of items, products, symptoms and characteristics, which often occur together in the given database, can be seen as one of the most basic tasks in Data Mining.
When the dataset size is huge, both memory use and computational cost can still be very expensive. In addition, single processor's memory and CPU resources are very limited, which make the algorithm performance inefficient. Furthermore; because of the exponential growth of worldwide information, enterprises (organizations) have to deal with an ever growing amount of data. As these data grow past hundreds of gigabytes towards a terabyte or more, it becomes nearly impossible to process (mine) them on a single sequential machine. The solution for the above problems is parallel and distributed computing.

Code:
Mapper:

```
package org.apache.mrapriori;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class FrequentItemsetMapper extends MapReduceBase implements Mapper<LongWritable, Text,
Text, IntWritable> {

	private Text itemset = new Text();

	@Override
	public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
im_output, Reporter reporter)
			throws IOException {

		String transaction = value.toString();
		List<String> itemsets = getItemSets(transaction.split(" "));

		for(String itmset : itemsets){
			itemset.set(itmset.replaceAll(" ", ""));
			im_output.collect(itemset, new IntWritable(1));
		}
	}

	//getting powersets (excluding empty set)
	private List<String> getItemSets(String[] items) {

		List<String> itemsets = new ArrayList<String>();

		int n = items.length;

		int[] masks = new int[n];

		for (int i = 0; i < n; i++)
			masks[i] = (1 << i);

		for (int i = 0; i < (1 << n); i++){

			List<String> newList = new ArrayList<String>(n);

			for (int j = 0; j < n; j++){
```

```
                                    if ((masks[j] & i) != 0){
                                            newList.add(items[j]);
                        }

                        if(j == n-1 && newList.size() > 0 && newList.size() < 5){
                                itemsets.add(newList.toString());
                        }
                                }
                }

                return itemsets;
        }
}
```

Partitioner:

```
package org.apache.mrapriori;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Partitioner;

public class FrequentItemsetPartitioner implements Partitioner<Text, IntWritable> {

        @Override
        public int getPartition(Text key, IntWritable value, int numReduceTasks) {

                int keySize = key.toString().replace("[", "").replace("]", "").split(",").length;

                if(numReduceTasks == 0)
     return 0;


                if(keySize == 1)
                        return 0;
                else if(keySize == 2)
                        return 1;
                else
                        return 2;
        }

        @Override
        public void configure(JobConf arg0) {
        }
}
```

Reducer:

```
package org.apache.mrapriori;
```

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class FrequentItemsetReducer extends MapReduceBase implements Reducer<Text, IntWritable,
Text, Text> {

        @Override
        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, Text>
output, Reporter reporter)
                        throws IOException {

                int sum = 0;
          while (values.hasNext()) {
                sum += values.next().get();
          }

          output.collect(key, new Text(Integer.toString(sum)));
        }
}
```

Computation Mapper:

```
package org.apache.mrapriori;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class ComputationMapper extends MapReduceBase implements Mapper<LongWritable, Text,
Text, Text> {

        private Text itemset = new Text();
        private Text val = new Text();

        @Override
        public void map(LongWritable key, Text value, OutputCollector<Text, Text> im_output,
Reporter reporter)
                        throws IOException {

                String[] valueSplit = value.toString().split("\\t");

                itemset.set(valueSplit[0].replaceAll(" ", ""));  //[1,2,3]
```

```
                        val.set("0;" + valueSplit[1]); //2

                        //original itemset
                        im_output.collect(itemset, val);

                        String[] items = valueSplit[0].replace("[", "").replace("]", "").split(",");

                        if(items.length > 1){

                                List<String> subitemsets = getItemSets(items);

                                for(String itmset : subitemsets){

                                        itemset.set(itmset.replaceAll(" ", ""));
                                        val.set(valueSplit[0] + ";" + valueSplit[1]);
                                        im_output.collect(itemset, val);
                                }
                        }
                }

                //generate powersets of exactly n-1 size
                private List<String> getItemSets(String[] items) {

                        List<String> itemsets = new ArrayList<String>();
                        int n = items.length;
                        int[] masks = new int[n];

                        for (int i = 0; i < n; i++)
            masks[i] = (1 << i);

                        for (int i = 0; i < (1 << n); i++){
                                List<String> newList = new ArrayList<String>(n);
                                for (int j = 0; j < n; j++){
              if ((masks[j] & i) != 0){
                newList.add(items[j]);
              }

              if(j == n-1 && newList.size() == n-1){
                itemsets.add(newList.toString());
              }
                                }
                        }

                        return itemsets;
                }
}
```

ComputationReducer:

```
package org.apache.mrapriori;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class ComputationReducer extends MapReduceBase implements Reducer<Text, Text, Text,
FloatWritable> {

		@Override
		public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, FloatWritable>
output, Reporter reporter)
				throws IOException {

				int keyCount = 0;
				HashMap<String,Integer> hashMap = new HashMap<String,Integer>();

				String[] keyitems = key.toString().replace("[", "").replace("]", "").split(",");

				while (values.hasNext()) {

						String val = values.next().toString();
						String[] valuesplit = val.split(";");   //[1,2,3];0 OR 0;1

								if(valuesplit[0].equals("0")){
										keyCount = Integer.parseInt(valuesplit[1]);
								}
								else{
										String[] parentItems = valuesplit[0].replace("[",
"").replace("]", "").split(",");
										hashMap.put(key + " -> " +
getSeparateItem(parentItems,keyitems),  Integer.parseInt(valuesplit[1]));
								}
				}

				Iterator<String> iterator = hashMap.keySet().iterator();

				while(iterator.hasNext()){
						String k = iterator.next().toString();
						int v = hashMap.get(k);

						output.collect(new Text(k.replace("[", "{").replace("]", "}")), new
FloatWritable(v/(float)keyCount));
				}
		}

		private String getSeparateItem(String[] parentItems, String[] keyitems) {

				String item = null;
				List<String> items = Arrays.asList(keyitems);
```

```
			for(String s : parentItems){
				if(!items.contains(s)){
					item = s;
					break;
				}
			}
			return item;
		}
}
```

Rule Mining:

```
package org.apache.mrapriori;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class RuleMining  extends Configured implements Tool {

	public static void main(String[] args) throws Exception {

		int res = ToolRunner.run(new RuleMining(), args);
    System.exit(res);
	}

	@Override
	public int run(String[] arg0) throws Exception {

		try{

			JobConf itemsetsJob = new JobConf(RuleMining.class);
			itemsetsJob.setJobName("Frequent Itemset generation");

			itemsetsJob.setOutputKeyClass(Text.class);
		  itemsetsJob.setOutputValueClass(IntWritable.class);

			itemsetsJob.setMapperClass(FrequentItemsetMapper.class);
		  itemsetsJob.setReducerClass(FrequentItemsetReducer.class);
		  itemsetsJob.setPartitionerClass(FrequentItemsetPartitioner.class);

		  itemsetsJob.setNumReduceTasks(4); //Each for itemsets of size 1, 2 and 3

		  itemsetsJob.setInputFormat(TextInputFormat.class);
		  itemsetsJob.setOutputFormat(TextOutputFormat.class);
```

```
                FileInputFormat.setInputPaths(itemsetsJob, new Path(arg0[0]));
                FileOutputFormat.setOutputPath(itemsetsJob, new Path(arg0[1]));

                JobConf computationJob = new JobConf(RuleMining.class);
                        computationJob.setJobName("Association rules confidence
computation");

                        computationJob.setOutputKeyClass(Text.class);
                computationJob.setOutputValueClass(Text.class);

                computationJob.setMapperClass(ComputationMapper.class);
                computationJob.setReducerClass(ComputationReducer.class);

                computationJob.setInputFormat(TextInputFormat.class);
                computationJob.setOutputFormat(TextOutputFormat.class);
                FileInputFormat.setInputPaths(computationJob, new Path(arg0[1]));
                FileOutputFormat.setOutputPath(computationJob, new Path(arg0[2]));

                        JobClient.runJob(itemsetsJob);
                        JobClient.runJob(computationJob);

              } catch (Exception e) {

                        e.printStackTrace();
              }
              return 0;
        }
}
```

Conclusion:

Thus the Frequent Itemset SON algorithm using Map Reduce has been implemented.

Post Lab Questions:

1. Illustrate PCY algorithm with an example.

- Roadmap:
  -> Frequent Patterns
  -> A-Priori Algorithm
  -> Improvements to A-Priori
  -> Park-Chen-Yu Algorithm
  -> Multistage Algorithm
  -> Approximate Algorithms
  -> Compacting Results

- PCY Algorithm
  -> Hash-based improvement to A-Priori.
  -> During Pass 1 of A-priori, most memory is idle.

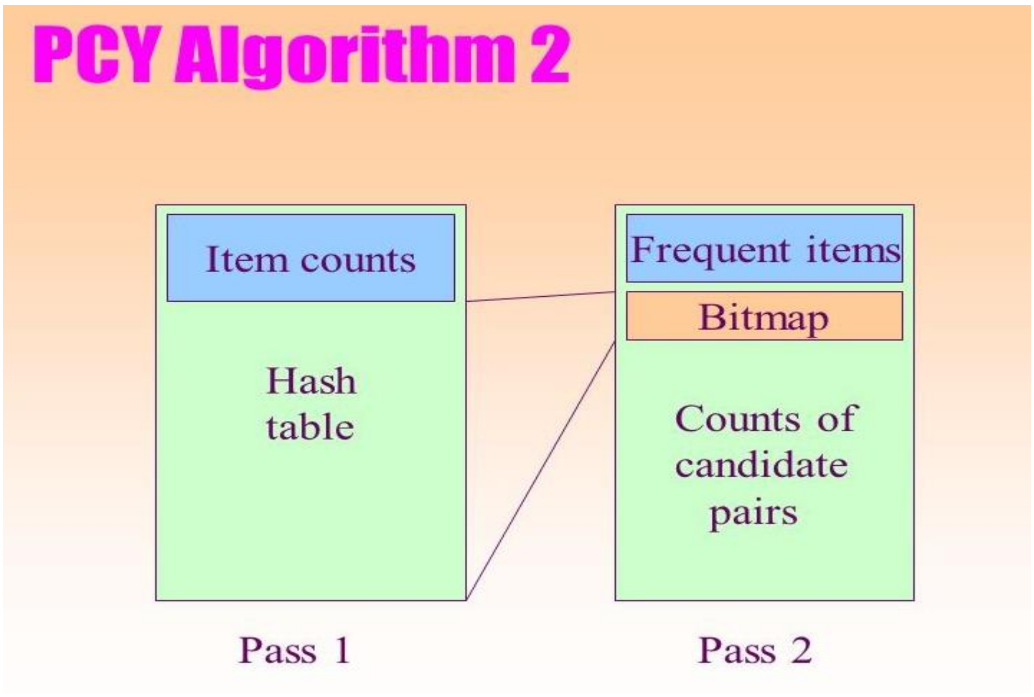
-> Use that memory to keep counts of buckets into which pairs of items are hashed.
-> Just the count, not the pairs themselves.
-> Gives extra condition that candidate pairs must satisfy on Pass 2

- Before Pass 1 Organize Main Memory
  -> Space to count each item.
  -> One (typically) 4-byte integer per item.
  -> Use the rest of the space for as many integers, representing buckets, as we can.

- PCY Algorithm --- Pass 1

```
FOR (each basket)
{
FOR (each item)
add 1 to item's count;
        FOR (each pair of items)
        {
                hash the pair to a bucket;
                add 1 to the count for that
                bucket
        }
}
```

- Observations About Buckets
  1. If a bucket contains a frequent pair, then the bucket is surely frequent.
     -> We cannot use the hash table to eliminate any member of this bucket.
  2. Even without any frequent pair, a bucket can be frequent.
     -> Again, nothing in the bucket can be eliminated.
  3. But in the best case, the count for a bucket is less than the support s.
     -> Now, all pairs that hash to this bucket can be eliminated as candidates, even if the pair consists of two frequent items.



- Between Passes
  -> Replace the buckets by a bit-vector:
  -> 1 means the bucket count exceeds the support s(frequent bucket); 0 means it did not.
  -> Integers are replaced by bits, so the bit-vector requires little second-pass space.
  -> Also, decide which items are frequent and list them for the second pass.

- PCY Algorithm --- Pass 2
  -> Count all pairs {i,j } that meet the conditions:
  1. Both i and j are frequent items.
  2. The pair {i,j }, hashes to a bucket number whose bit in the bit vector is 1.
  -> Notice all these conditions are necessary for the pair to have a chance of being frequent.

- Memory Details
  -> Hash table requires buckets of 2-4 bytes.
  -> Number of buckets thus almost 1/4-1/2 of the number of bytes of main memory.
  -> On second pass, a table of (item, item, count) triples is essential.
  -> Thus, hash table must eliminate 2/3 of the candidate pairs to beat a-priori.