1. Suppose that the data mining task is to cluster the following nine points (with (x,y) representing location) into three clusters: A(3,9), A(2,5), A(9,4), B(4,8), B(8,5), B(7,4), C(2,2), C(5,10), C(6,8). Suppose initially we assign A, B, and C as the center of each cluster, respectively. Please add a Map-reduce function for the K-means algorithm. Show the results for the first two iterations and explain how Map-reduce can help.

Answer:

Map Reduce:

Let K_1, K_2, K_3 be the three centroids for the current iteration, let d(P, K) be the distance between points P and K.

Map: We map each points with coordinates i, j, P(i, j) to l if P is closest to centroid K. A sample map function could be

\[
\text{map}(P) \{ \\
\quad \text{emit(index_of_cloest_cetroid(K_1, K_2, K_3, P), P)} \\
\}
\]

Reduce:

\[
\text{reduce(tuples)} \{ \\
\quad \text{return } \left[ \text{tuples.centroid_index, sum(tuples.x)/tuples.count, sum(tuples.y)/tuples.count} \right]; \\
\}
\]

Execution:

Iteration 1:

Cluster 1: A, C  
Centroid 1: (4, 9.5)

Cluster 2: B, C, B, B, A  
Centroid 2: (6.8, 5.8)

Cluster 3: C, A  
Centroid 3: (2, 3.5)
Iteration 2:

Cluster 1: A, B, C,
Centroid 1: (4, 9)

Cluster 2: C, B, B, A
Centroid 2: (7.5, 5.25)

Cluster 3: C, A
Centroid 1: (2, 3.5)

Benefit:

The map reduce can help in the sense that all these operations can run in parallel.

2. A database has six transactions. Let min sup = 50%.

<table>
<thead>
<tr>
<th>TID</th>
<th>items_sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>A, B, C, D, E, F</td>
</tr>
<tr>
<td>T002</td>
<td>B, H, E, C, F, T</td>
</tr>
<tr>
<td>T003</td>
<td>C, U, O, E, W, D</td>
</tr>
<tr>
<td>T004</td>
<td>W, A, B, C, F, X</td>
</tr>
<tr>
<td>T005</td>
<td>W, X, C, D, F, Y</td>
</tr>
<tr>
<td>T006</td>
<td>B, C, D, E, O, Z</td>
</tr>
</tbody>
</table>

Please add a Map-reduce function for the Apriori algorithm to generate all frequent itemsets. Show the results for each step and explain how Map-reduce can help.

Answer:

Map: For each iteration, generate frequent items.

    function(doc) {
        var iteration;
        var frequent_itemset;
        if(0 == iteration)
for_each(trans = doc.transactions)
    for_each(item = trans.items)
        frequent_itemset.add(item);
    else{
        // the function below take each two frequent_itemset, generate the union of them and add in the set.
        frequent_itemset = set( Cartesian_union(frequent_itemset) );
    }
for_each(trans in doc.transactions){
    for_each(item in frequent_itemset){
        if (trans.contains(item)) {
            emit(item, 1);
        }
    }
}

Reduce: Count all the emitted items, compare with the support threshold.
reduce(keys, values, reducer){
    var count = _count;
    if (count / doc.transactions.length > doc.support)
        return (c, value)
}

L (frequency >= 3):
B, 4
C, 6
D, 4
E, 4
F, 4
W, 3
Candidate C,

BC, BD, BE, BF, BW, CD, CE, CF, CW, DE, DF, DW, EF, EW, FW.

L,

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>4</td>
</tr>
<tr>
<td>BE</td>
<td>3</td>
</tr>
<tr>
<td>BF</td>
<td>3</td>
</tr>
<tr>
<td>CD</td>
<td>4</td>
</tr>
<tr>
<td>CE</td>
<td>4</td>
</tr>
<tr>
<td>CF</td>
<td>4</td>
</tr>
<tr>
<td>CW</td>
<td>3</td>
</tr>
<tr>
<td>DE</td>
<td>3</td>
</tr>
</tbody>
</table>

C,: BCE, BCF, CDE

L,

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BCE</td>
<td>3</td>
</tr>
<tr>
<td>BCF</td>
<td>3</td>
</tr>
<tr>
<td>CDE</td>
<td>3</td>
</tr>
</tbody>
</table>

C,

None.

Benefit:
The map reduce function can help process the map reduce function in parallel.