Co-clustering with High Machine Scalability

April 29, 2015

Abstract

DisCo, the clustering method using MapReduce, is introduced in 2008. It requires the main thread to collect the assignment of cluster to all rows and columns, and broadcast the assignment to all mappers before the each MapReduce task starts. The time complexity of this process is only related with number of rows and column. Therefore, although the matrix is sparse or not, gathering and broadcasting the cluster assignment takes $O(m+n)$ times. In this paper, we introduce a improved version of DisCo, by deleting it's bottleneck processes.

1 Itroduction

Clustering is one of the key issues of graph mining to find valuable knowledges from the raw data. As the graph size is getting bigger, the methods to manage these big dataset is also needed. The *DisCo* is one of the clustering methods managing big data by using Map-Reduce introduced in 2008. Each task of Disco is consisted with broadcasting parameter r and c, MapReduce job, modify $r(i)$, $c(j)$ for all rows and columns. As m and n grows, the most burden part is broadcasting the $r(i)$, $c(j)$. The biggest problem of DisCo is, broadcasting process has no machine scalability. Whether the matrix is sparse or not, amount of information that the main thread should broadcast is same as $O(m+n)$. Therefore the machine scalability is very poor as the row and column dimension getting bigger.

Our main contribution is giving DisCo to have higher machine scalability, i.e. removing broadcasting process. By this method, we removed the broadcasting, gathering process from the all MapReduce tasks, and having good machine scalability.

2 Background : DisCo

DisCo is consisted with two main phases. First phase is pre-processing, making row, column adjacency list as sequence file on HDFS as follows.

Key : i
Value : $radj_i$

Second phase is Co-Clustering process. Before the DisCo Co-Clustering the row once, it broadcasts S, c to all mapper. The space complexity of the parameters is $O(k \times l + n)$. Because the broadcasting process has no machine scalability, as the n getting bigger, the time to broadcasting parameters is higher whether the adjacency matrix is sparse or environment has many machines. This is main disadvantage of DisCo. Also, the CollectResult() is conducted by main thread which requares $O(m)$ calculation.

Symbol	Definition
A	the m by n matrix
m, n	Number of rows and columns
i, j	Row and Column indices
$radj, cadj$	Row and column adjacency list
$radj_i$, $cadj_j$	Row adjacency list of row i, column adjacency list of column j
$a_{i,j}$	(i, j) element of A
r, c	Cluster assignment of row, column
$r(i)$	Cluster assignment of row i
$c(j)$	Cluster assignment of column j
$I(v)$	Set of row indices with $r(i) = v$
$J(w)$	Similar to $I(v)$, but for columns
R, C	Size of row, column cluster
$R(v)$	Size of row cluster v
$C(w)$	Size of column cluster w
S	$k \times l$ statistic matrix
k, l	Size of row, column cluster, respectly
v, w	row and column cluster index
$S_{v,w}$	The (v, w) element of S
$S_{v,:}$	vth row vector of S
$S_{:,w}$	wth column vector of S
rst_i	Row Statistic vector of size l driven by $radj_i$ and c
cst_j	Similar to rst_i, but for columns
$Rdata$	HDFS sequence file of row information
$Cdata$	HDFS sequence file of column information
$Rdata_i$	$(key, value)$ pair of sequence file $Rdata$. $(key, value) = (i, (r(i), rst_i, radj_i))$
$Cdata_j$	$(key, value)$ pair of sequence file $Cdata$. $(key, value) = (j, (c(j), cst_j, cadj_j))$

Table 1: Definition of Symbols

Algorithm 1: CC(A, k, l)

```
Initialize r and c
preprocess()
repeat
    CC-MapReduce() begin
        Broadcast c, S to all mapper.
        CCRowMapper(key key, value value)begin
            (key, value) = (i, rad_i)
            rst_i = row statistic(radj_i, c) for v = 1..k do
                if assigning i to v would lower cost then
                    r(i) = v
                end
            end
            Out(r(i), (rst_i, i))
        end
        CCRowReducer(key key, values V)
        begin
            v = key
            S_v= empty array of size l I_v = ∅ for (rst_i, i) ∈ V do
                S_v = comibine statistics(S_v, rst_i)
                I_v = I_v ∪ i
            end
            Out(v, (S_v, I_v))
        end
    end
    CollectResults() begin
        /* Conducted by main thread                                        */
        S = k by l empty matrix
        for reducer output (v, (S_v, I_v)) do
            S_{v,:} = comibine statistics(S_{v,:}, S_v) for i ∈ I_v do
                r(i) = v
            end
        end
    end
    Do the same for columns
until cost does not decrease
return r and c
```

3 Co-Clustering without broadcasting

Our method is consisted with two phases. First is preprocessing the data. The difference between DisCo and our method in preprocessing data is that the initialized r, c is broadcasted to all mapper to set $r(i)$ and calculate rst_i for all row index i.

$$\text{Key : } i$$
$$\text{Value : } r(i),\ \ rst_i,\ \ radj_i$$

It is same for column. These two MapReduce tasks are the *only tasks* requires r and c need to be broadcasted.

The sceond phase is Co-Clustering. The difference between DisCo and our method is the parameter should be maintained, gathered and broadcasted is only S. The space complexity of the paramter is $O(k \times l)$ while

the DisCo requires $O(n)$ data to run the row coclustering task. The key idea to remove broadcasting c from all Co-Clustering the row is make the data already has the information of rst_i. In the line 6 of Algorithm1, rst_i is calcluated by $radj_i$ and c. If the data already has $radj_i$, then broadcasting c is unnecessary.
In summary, row co-clustering requires rst_i to change $r(i)$ of all rows and cst_j of all columns. Our key idea is is let the data already has rst_i, and make the result change $r(i)$ and cst_j within one row coclustering task. For acheiving this, our method contains one Map-only task, two MapReduce tasks.

1. CC-Mapper : Map-Only Tasks. Takes $Rdata$, $Cdata$ and return $Rdata$ and following three types of data for oncoming MapReduce tasks
   (a) S_{part} : set of $(r(i), (1, rst_i))$ to update S, R
   (b) $Rdata_{part}$: set of $(r(i), radj_i)$ to calculate cst_j of all column j
   (c) $Cdata_{part}$: set of $(j, (c(j), cadj_j))$ to reconstruct $Cdata$
2. CC-Constructor : MapReduce Task. Takes $Rdata_{part}$ and $Cdata_{part}$ to construct $Cdata$
3. CC-Collector : MapReduce Task. Takes S_{part} to update S and R

Algorithm 2: CC-*new*(A, k, l)

repeat
    CC-Mapper()
    CC-Constructor()
    CC-Collector()
    Do the same for columns
until *cost does not decrease*
return r and c

Algorithm 3: CC-Mapper

CC-*new*-Mapper(key key, value $value$)/* Map only task */
Input: $Rdata_i, Cdata_j$
Output: $Rdata, S_{part}, Rdata_{part}, Cdata_{part}$
begin
    if *data form is* $Rdata$ **then**
        $(i, (r(i), rst_i, radj_i)) = (key, value)$
        for $v = 1..k$ **do**
            if *assigning* i *to* v *would lower cost* **then**
                $r(i) = v$
            end
        end
        $Rdata_i = (r(i), rst_i, radj_i)$
        $Out(i, Rdata_i)$ to $Rdata$
        $Out(r(i), (1, rst_i))$ to S_{part}
        $Out(r(i), (i, radj_i))$ to $Radat_{part}$
    end
    else
        $(j, (c(j), cst_j, cadj_j)) = (key, value)$
        $Out(j, (c(j), cadj_j))$
    end
end

Note that the line 28 of Algorithm 4 calculate cst_j with Bag, which is hashmap having tuple as (cluster id, set of indices). DisCo calculate cst_j using $cadj_j$ and r, but most of Co-Clustering algorithm such as *CrossAs-*

sociation needs only Bag to calculate cst_j.

Algorithm 4: CC-Constructor

Input: $Rdata_{part}, Cdata_{part}$
Output: $Cdata$
begin
  CC-Const-Mapper(key key, value $value$) **begin**
    if *data has form* $Rdata_i$ **then**
      $(r(i), radj_i) = (key, value)$
      for $j \in radj_i$ **do**
        $Out(j, (r(i), i))$
      end
    end
    else
      $Out(key, value)$
    end
  end
  CC-const-Reducer(key key, values $Values[]$)
  begin
    $key = j$
    Bag = empty HashMap.
    cst_j =empty array of size k
    for $value \in Values$ **do**
      if $value = (c(j), cadj_j)$ **then**
        continue
      end
      else
        $(v, i) = value$
        add i to $Bag.get(v)$
      end
    end
    Bag became set of tuple (v, set of row index i with $r(i) = v$)
    cst_j = *column statistic*(Bag)
    $Cdata_j = (c(j), cst_j, cadj_j)$
    $Out(j, Cdata_j)$ to $Cdata$
  end
end

Algorithm 5: CC-Collector

Input: S_{part}
Output: S, R

begin
    CC-collect-Mapper(key key, value $value$) **begin**
        $(r(i), (1, rst_i)) = (key, value)$ for some row i
        $x =$ arbitrary value of 0 to 1
        $key = (r(i) + x) \times NumOfReducer$
        `/* To make data spreads to all reducer equivalently */`
        $Out(key, value)$
    end
    CC-collect-Reducer(key key, values $Values[]$)
    begin
        $v = (key - (key\%NumOfReducer)) \div NumOfReducer$
        $S_{v,:}$= empty array of size l
        $R(v) = 0$
        for $value \in Values$ **do**
            $value = (1, rst_i)$ for some row i
            $R(v) = R(v) + 1$
            $S_{v,:} =$ *comibine statistics*$(S_{v,:}, rst_i)$
        end
        $Out(v, (R(v), S_{v,:}))$ to S
    end
    Combine all reducer output to R, S
end
Update R, S
Update H

4 Experiment

Our method require several intermediate data, but showed good machine scalablity. DisCo also showed the good machine scalabiltiy in MapReduce task and it is faster than our methods if we compare only MapReduce task. But, DisCo takes a lot of time to broadcast the parameter as m, n gets bigger, even if the MapReduce task takes less time than ours. The number of machines, density of the data are the two factor of the Co-Clustering to decide which method to choose. The conclusion is, if the data is sparse and the enviornment has lots of machines, our method is better than DisCo.

5 Colclusion