Clustering Large scale data using MapReduce
Topics

• Clustering Very Large Multi-dimensional Datasets with MapReduce by Robson L. F. Cordeiro et al.

• Fast Clustering using MapReduce by Alina Ene et al.
Background stuff

• Hyper rectangle
  – A rectangle in n-dimensions. Used as cluster output.

• Map Reduce phases
  – Mapper: Takes a single key, value pair and generates any number of key, value output pairs
  – Shuffle: Sorts key, value pairs given by Mapper so that a single Reducer can get values of same keys.
  – Reducer: Takes values from the output of a Mapper for a single key and outputs key, value pairs
Topic 1

Clustering Very Large Multi-dimensional Datasets with MapReduce by Robson L. F. Cordeiro et al.
Contributions

• Proposes a novel Best of Both Worlds (BoW) approach
• Chooses one among two approaches - SnI and ParC for clustering
• Proposes cost estimates for each of this approaches based on environment parameters to make the decision
• Modular – Can allow plugging in sequential clustering algorithms
• Shows good performance improvements – 0.2 TB getting clustered in 8 minutes using 128cores.
ParC – Partition data

- P1: Parition data based on different approaches
  - *random data partitioning*: elements are assigned to machines at random (load balanced)
  - *address-space data partitioning*: eventually nearby elements in the data space often end up in the same machine (not load balanced)
  - arrival order or ‘file-based’ data partitioning: first several elements in the collection go to one machine, the next batch goes to the second, and so on (load balanced)
- P2: Shuffle partitioned output
- P3: Reducers clusters the data received. Uses plug in which does the actual serial clustering
- P4: Send cluster description
- P5: Merge all cluster descriptions to produce final set of clusters.
ParC Example

Taken from “Clustering Very Large Multi-dimensional Datasets with MapReduce” by Robson L. F. Cordeiro et al
Prons/Cons

+ Parses input only once
- High network cost:
  - All data is passed from mappers to reducers in the P2 shuffle step.
SnI approach

• S1: Read data and sample
• S2: Shuffle sampled data
• S3: Single reducer which looks for clusters
• S4: Send cluster descriptions to mappers
• S5: Read data in parallel in ignore clustered ones
• S6: Shuffle the unclustered data
• S7: Cluster unclustered data
• S8: Send cluster descriptions to one machine
• S9: Merge all resulting clusters
SnI example

Taken from “Clustering Very Large Multi-dimensional Datasets with MapReduce” by Robson L. F. Cordeiro et al
Pros/Cons

• + Low network and shuffle cost
  – Sampling reduces the amount of data
• - Reads the input data twice (I/O cost)
Which approach is better?

• Answer depends on the environment and input parameters
  – Network speed
  – File size
  – Disk speed
  – Instance start up task
  – Sequential clustering plug-in complexity
Cost estimate for ParC

\[ costC = costM(m, F_s) + costS(r, F_s) + costR(r, F_s) \]

where,

\[ costM(m, F_s) = startUpCost(m) + \frac{s}{m \times D_s} \]

\[ costS(r, F_s) = \frac{s \times D_r}{r \times N_s} \]

costM – cost of mappers

costS – Shuffling cost

costR – cost of reducer step

m – number of mappers

r – number of reducers

F_s – size of data file

N_s – network data rate

D_r – ratio of data transferred from mapper to reducer. This is < 1 as map reducer optimizes by assigning reducer to machine which already ran mapper
Cost estimate for SnI

\[ costCs = 2 \times costM(m, F_s) + costS(1, F_s \times S_r) + \text{costR}(1, F_s \times S_r) + costS(r, F_s \times R_r) + \text{costR}(r, F_s \times R_r) \]

where,

- \( costM \) – cost of mappers
- \( costS \) – Shuffling cost
- \( costR \) – cost of reducer step
- \( m \) – number of mappers
- \( r \) – number of reducers
- \( F_s \) – size of data file
- \( S_r \) – Sampling rate
- \( R_r \) – Ratio of data that doesn’t belong to major clusters
Best of Both Worlds Algorithm

- Compute costC
- Compute costCs
- If costC > costCs
  - Use SnI
- else
  - Use ParC
Results

Twitter dataset, using ~700 mappers

(a) Wall-clock time (seconds) vs. Number of reducers

(b) Wall-clock time (seconds) vs. Number of reducers

Characteristics:
- Shrink network cost
- Shrink disk accesses

Best: BoW
Evaluation

• Cluster quality was compared using synthetic data
• Shows significant amount of scaling up. Processed 0.2 TB in 8 minutes using 128 cores
• The results show that cost estimates are quite accurate. (Correct approach was chosen)
Topic 2

Fast Clustering using MapReduce
Contributions

• Proposes fast clustering algorithms with performance guarantees
• Theoretical proof that the proposed algorithm belongs to class MRC0
• Can be applied to both K-median and K-center clustering techniques
MRC class

• The total memory used on a specific machine is sub-linear
• The total number of machines used is sub-linear
• The number of rounds is constant
K center and K median

- **K center** – Choose k points each representing a cluster. The maximum distance between a center and a point assigned to it is minimized.

- **K median** – Minimize the sum of distances from the centers to each of the points assigned to the centers.

- Both are NP complete
Some definitions...

• Distance from a point P to set of points S is the minimum distance between P and any point in S.
• Instead of clustering the entire data, sampled data can be clustered.
• The sample data might not represent the entire set of points in the dataset.
• The left-out data is called non-represented data.
• Basic Idea: Keep sampling until non-representation data is below a certain number.
Sequential Iterative Sampling

• Set $S$ – sampled data, $R$ – not-represented data
• $S \leftarrow \emptyset$, $R \leftarrow V$ (input set)
• while $|R| >$ some threshold
  – Add each point in $R$ to $S$
  – Add each point in $R$ to $H$
• Sort descending order points in $H$ from farthest to smallest. Pick $8 \log n$th point in this order as $v$.
• Remove points from $R$ if the distance is smaller than the distance of $v$ to $S$.
• Repeat while
• Output = $S \cup R$ (include the not-represented data as well)
MapReduce version of iterative sampling

• While $|R| > T$
• Arbitrarily partition $R$ into $n$ sets. Each of the sets will go to same reducer
• At each reducer, sample the received data and form sets $S_i$ and $H_i$
• Read samples in a single mapper machine and form $H = \text{union of } H_i's$, $S = \text{union of } R_i's$
• Reducer receives this $H$ and $S$ finds $v$ (as in sequential case) $8 \log n$th value in the sorted order of points in $H$ based on the descending order of distance from the point to $S$.
• Mappers partition the points in $R$ into subsets. The next reducer receives the $R_i$ (a single partition), $v$ and $S$.
• Reducer $i$ finds the distance of each point in $R_i$ to $S$. The point is removed if distance $< \text{distance}(v,S)$
• Union all $R_i's$ at reducer
• Repeat while
• Output $C : S \cup R$
Properties of Iterative sampling

• The number of iterations in the loop is constant with high probability
• The set returned by Iterative-Sample is in the order of $\log(n)$ with high probability
**MapReduce-kcenter**

- $C \leftarrow \text{Iterative-Sample()}$
- Map $C$ and all pairwise distances between points in $C$ to a reducer
- The reducer runs a k-center clustering algorithm on $C$ (sequential version)
- Return the constructed clusters
- Note: Sampling works? It is said at results section that sampling doesn’t work very well with k-center.
kMedian

• Since k-median needs sum of distances, each unsampled point has to be represented somehow.
• The closest sampled point is the representative.
• Assign weight to every sample point based on the number of points it represents
MapReduce-K Median

- C <- MapReduce-Iterative-Sample
- Partition input data V into sets Vi
- Mappers assign Vi, C to a reducer
- Each reducer i, computes weight for every point in C based on points in Vi
- Map all weights, C and pair-wise distances in C to a single reducer
- The reducer computes weight $w(y) = \sum(w_i(y))$ calculated in previous step
- The reducer then runs a weighted k-median clustering algorithm
- Return the clusters formed
Results observed

• K-Centers doesn’t perform very well with sampling
• Maximum distance from a point to a center can be disturbed even if a single point is missed.
• Results presented only for k-median
• Shows 1000x speedup over LocalSearch
• About 20x speedup over Parallel-Lloyd