# BIRCH

Balanced Iterative Reducing and Clustering using

Hierarchies

# BIRCH

- Designed for very large data sets
  - Time and memory are limited
  - Incremental and dynamic clustering of incoming objects
  - Only one scan of data is necessary
  - Does not need the whole data set in advance

# BIRCH

- Incrementally construct a CF (Clustering Feature) tree, a hierarchical data structure for multiphase clustering
  - Phase 1: scan DB to build an initial in-memory CF tree (a multi-level compression of the data that tries to preserve the inherent clustering structure of the data)
  - Phase 2: use an arbitrary clustering algorithm to cluster the leaf nodes of the CF-tree

# Clustering parameters

- Centroid $\vec{x}_0$ – Euclidian center
- Radius(R) – average distance to center
- Diameter(D) – average pair wise difference within a cluster

Radius and diameter are measures of the tightness of a cluster around its center. We wish to keep these low.

$$\vec{x}_0 = \frac{\sum_{i=1}^{N} \vec{x}_i}{N}$$

$$R = \left(\frac{\sum_{i=1}^{N} (\vec{x}_i - \vec{x}_0)^2}{N}\right)^{\frac{1}{2}}$$

$$D = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\vec{x}_i - \vec{x}_j)^2}{N(N-1)}\right)^{\frac{1}{2}}$$

# Clustering Feature

- A Clustering Feature is a triple summarizing the information that we maintain about a cluster.
- Definition: Given N d-dimensional data points in a cluster: $\{\vec{x}_i\}$ where i = 1,2,…N, the Clustering Feature (CF) vector of the cluster is defined as a triple: $CF = (N, \vec{LS}, SS)$ , where N is the number of data points in the cluster, $\vec{LS}$ is the linear sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{x}_i$ and SS is the square sum of the N data points, i.e., $\sum_{i=1}^{N} \vec{x}_i^{\,2}$

# CF Additivity Theorem

- If CF1 = (N1, LS1, SS1), and

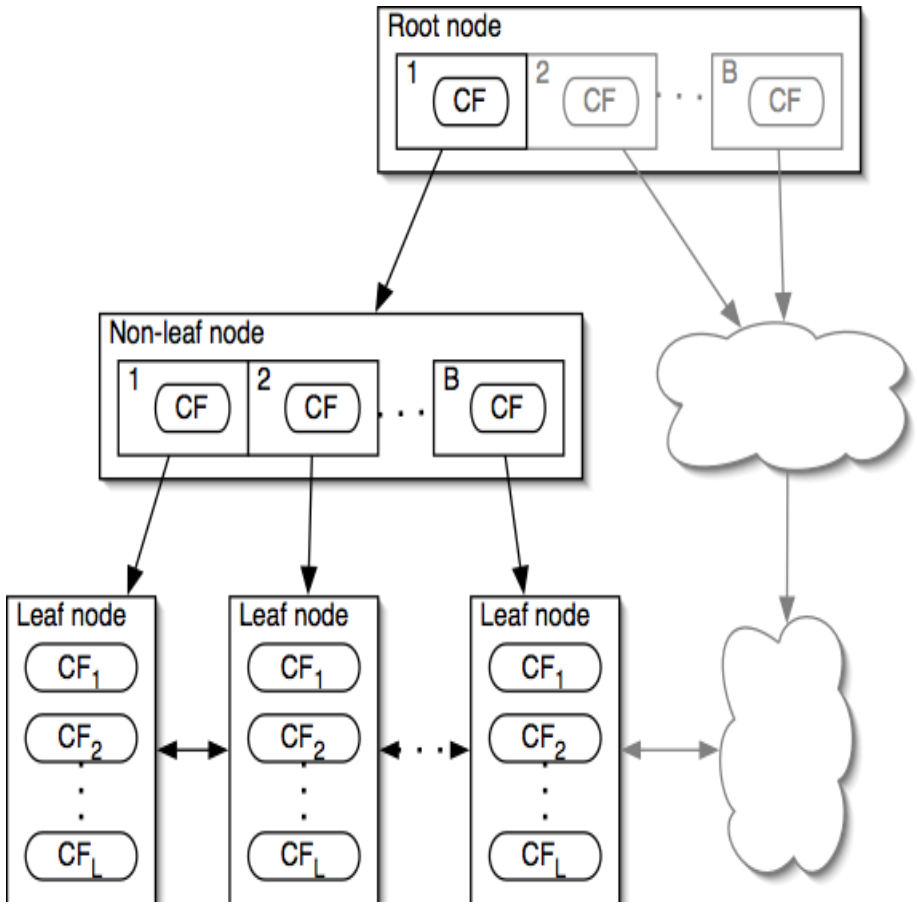
CF2 = (N2 ,LS2, SS2) are the CF entries of two disjoint sub-clusters.

- The CF entry of the sub-cluster formed by merging the two disjoin sub-clusters is:

CF1 + CF2 = (N1 + N2 , LS1 + LS2, SS1 + SS2)

# Properties of CF-Tree

- Each non-leaf node has at most B entries
- Each leaf node has at most L CF entries which each satisfy threshold T
- Node size is determined by dimensionality of data space and input parameter P (page size)



# CF Tree Insertion

- Identifying the appropriate leaf: recursively descending the CF tree and choosing the closest child node according to a chosen distance metric
- Modifying the leaf: test whether the leaf can absorb the node without violating the threshold. If there is no room, split the node
- Modifying the path: update CF information up the path.

# Birch clustering algorithm

- Phase 1: Scan all data and build an initial in-memory CF tree.
- Phase 2: condense into desirable length by building a smaller CF tree.
- Phase 3: Global clustering
- Phase 4: Cluster refining – this is optional, and requires more passes over the data to refine the results

# Birch - phase 1

- Start with initial threshold and insert points into the tree
- If run out of memory, increase thresholdvalue, and rebuild a smaller tree by reinserting values from older tree and then other values
- Good initial threshold is important but hard to figure out
- Outlier removal – when rebuilding tree remove outliers

# Birch - phase 2

- Optional
- Phase 3 sometime have minimum size which performs well, so phase 2 prepares the tree for phase 3.
- Removes outliers, and grouping clusters.

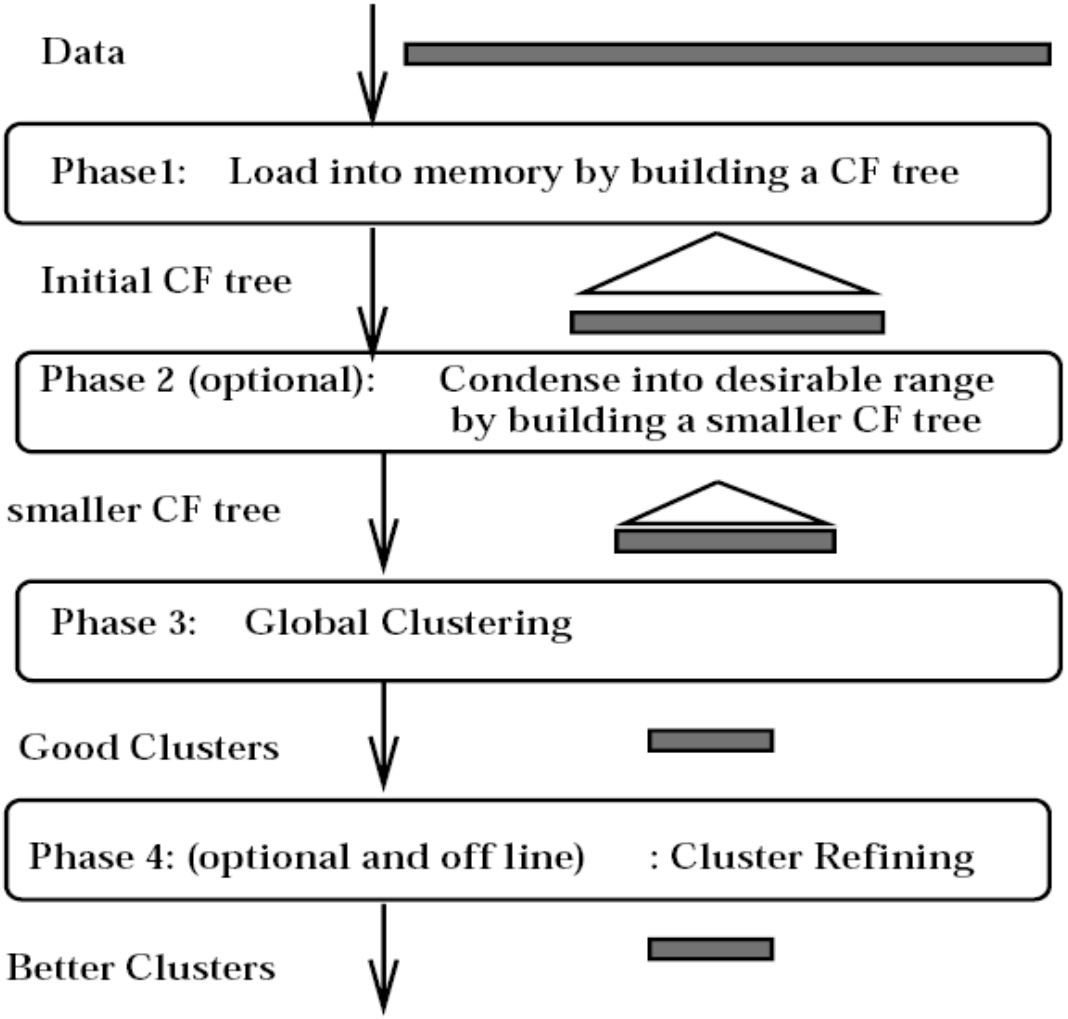
# Birch - phase 3

- Problems after phase 1:
  - Input order affects results
  - Splitting triggered by node size
- Phase 3:
  - cluster all leaf nodes on the CF values according to an existing algorithm
  - Algorithm used here: agglomerative hierarchical clustering

# Birch - phase 4

- Optional
- Do additional passes over the dataset & reassign data points to the closest centroid from phase 3
- Recalculating the centroids and redistributing the items.
- Always converges (no matter how many time phase 4 is repeated)

# BIRCH overview



# Weakness

- Handles only numeric data
- Sensitive to the order of the data record