Mining Large Data with Hadoop

by Yaseen Qadah

Tutorial for
CSE 8331 – Advanced Topics in Data Mining
Spring 2012

Outlines

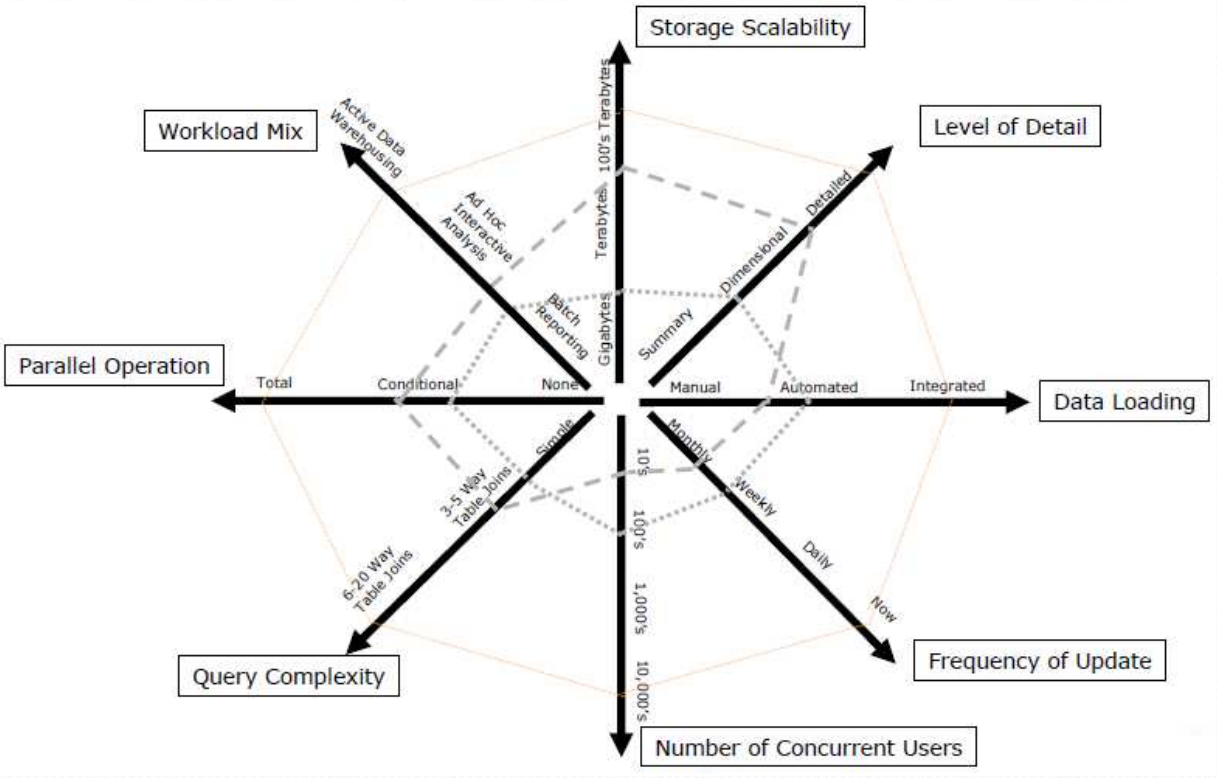
- Large Data (Big Data)
- Big Data Issues
- Data Storage and Analysis (Issues and solutions)
- What is Hadoop?
- Hadoop Distributed File System (HDFS)
- HDFS Data Model
- HDFS Master and Worker
- MapReduce
- Hadoop Projects
- Mahout
- HIVE
- RHIVE

Large Data (Big Data)

- Extremely large datasets that their sizes are beyond the ability of capturing, managing, and processing by most software tools and people. [7]
- Examples:
  - Search engines, social networking, and online advertising, as well as education, health care, and medicine

Big Data Issues

- Size of data has been exceeded Petabytes (10^{15} bytes)
- The size is not an issue but the processes are



Data Storage and Analysis (Issues and solutions)

- Data transfer rate in single database has been increased due to the size of data
- *Solution: Parallel Databases*
- Most data analysis methods need the data to be combined
- *Solution: Distributed Systems*
- Hardware failures have been increased due to the increase in the number of used devices
- *Solution: Replication to avoid data loss*

What is Hadoop?

- Open source apache project written in Java
- Software framework for distributed processing of large data sets [1]
- Collection of subprojects under the infrastructure of distributed computing.
- Enables applications to work with thousands of node and petabytes of data.
- Provides a reliable shared storage and analysis system.
- Core includes:
  - Distributed File System (DFS)
  - MapReduce
  - Job Tracker/Task Tracker

Hadoop Distributed File System (HDFS)[16]

- A distributed file system that provides access to data.
- Designed for batch processing instead of interactive use.
- Provides high-throughput access to the data.
- Provides write-once-read-many access model for files.
- Provides interfaces for applications to move closer to the location of needed data.
- Has a master/slave architecture.

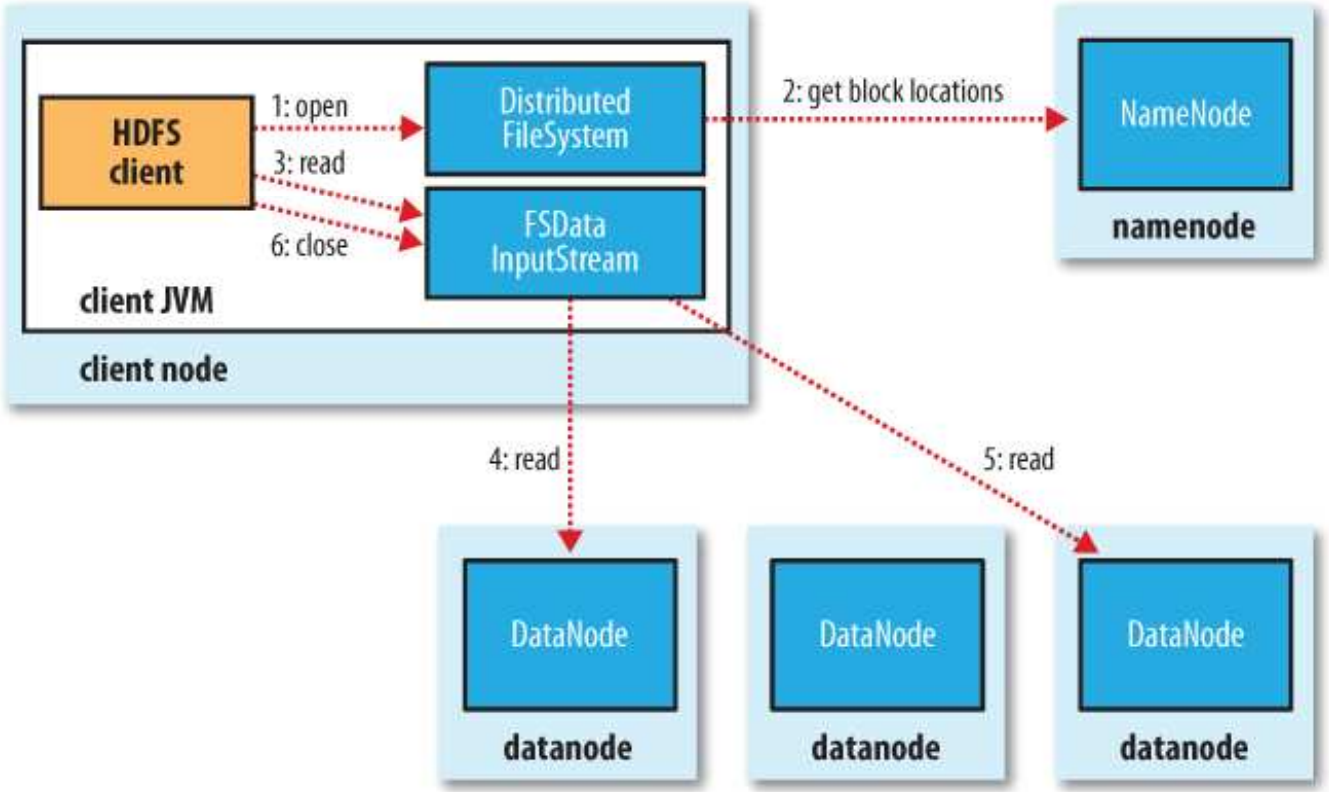
HDFS Data Model

- Data is organized into files and directories
- Files are divided into uniform sized blocks and distributed across cluster nodes
- Blocks are replicated to handle hardware failure
- Corruption detection and recovery
- Exposes block placement so that computes can be migrated to data

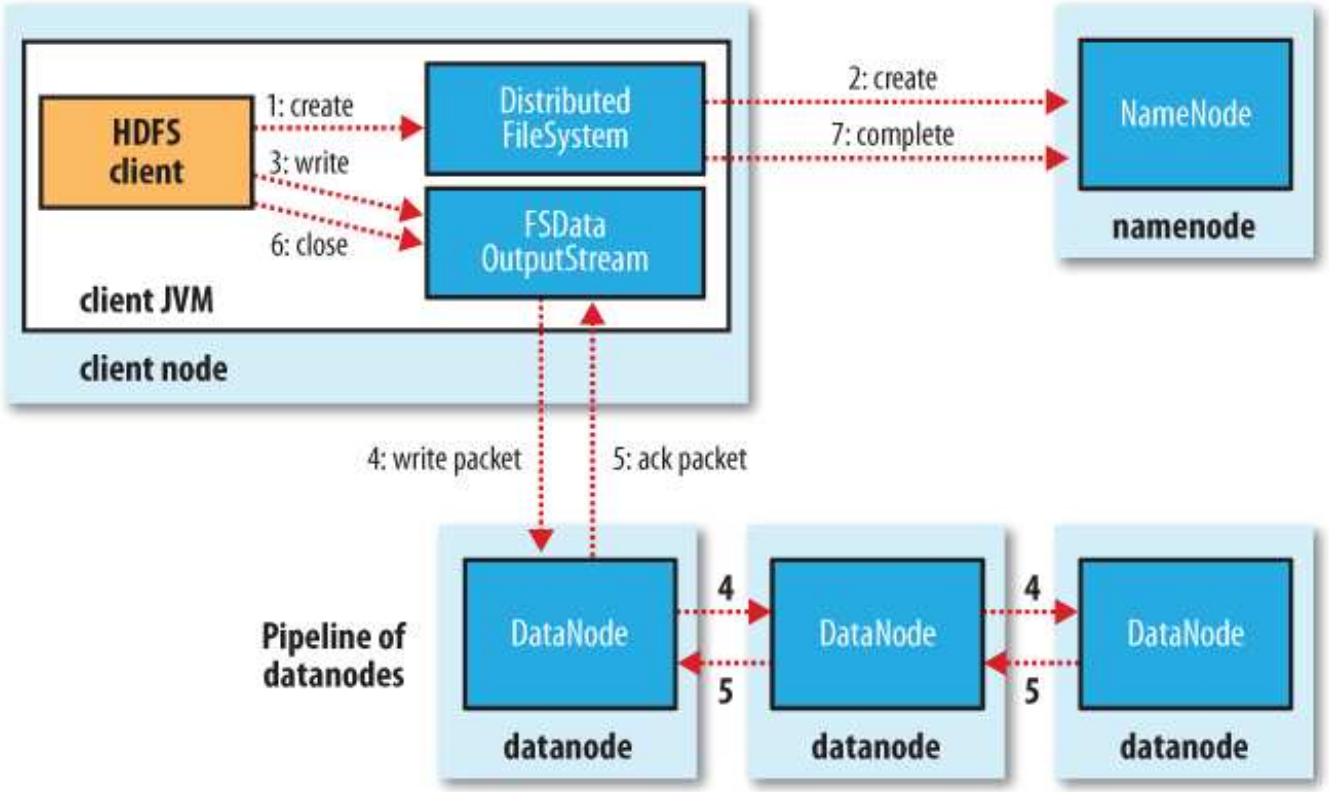
HDFS Master and Worker

- HDFS Master "Namenode"
  - Manages the filesystem namespace
  - Controls read/write access to files
  - Manages block replication
  - Reliability: Namespace checkpointing and journaling
- HDFS Workers "Datanodes"
  - Serve read/write requests from clients
  - Perform replication tasks upon instruction by Namenode

- Read from file



- Write to file



MapReduce (MR) [8, 10, 12]

- Introduced by Google in 2004
- Is a programming model for processing large datasets on distributed computing
- The model is derived from map and reduce functions
- Run MR jobs
  - Run one MR job
    - Output files on reduce node
  - Run next MR job
    - Input files are prior reducer output
    - Create new reducer files
  - Keep running MR batch jobs until solved

MR example

- Counting the number of occurrence of each word in large collection of documents:

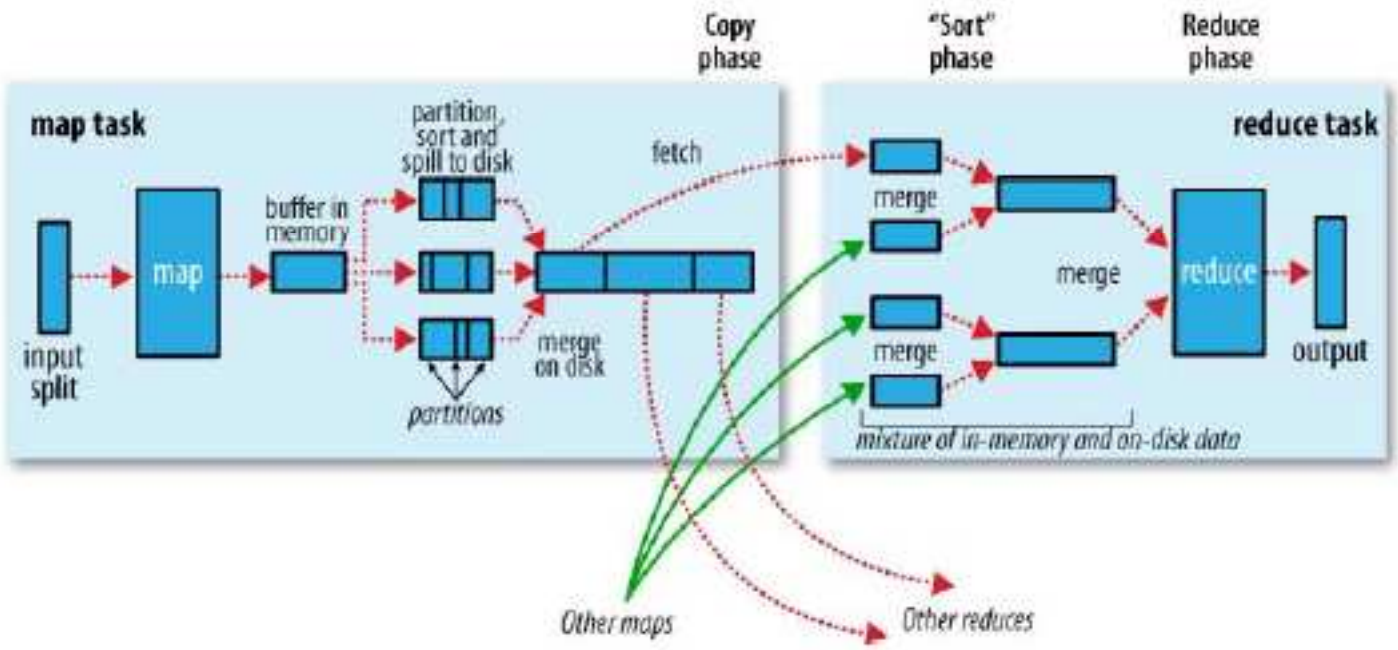
```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
    result += ParseInt(v);
Emit(AsString(result));
```

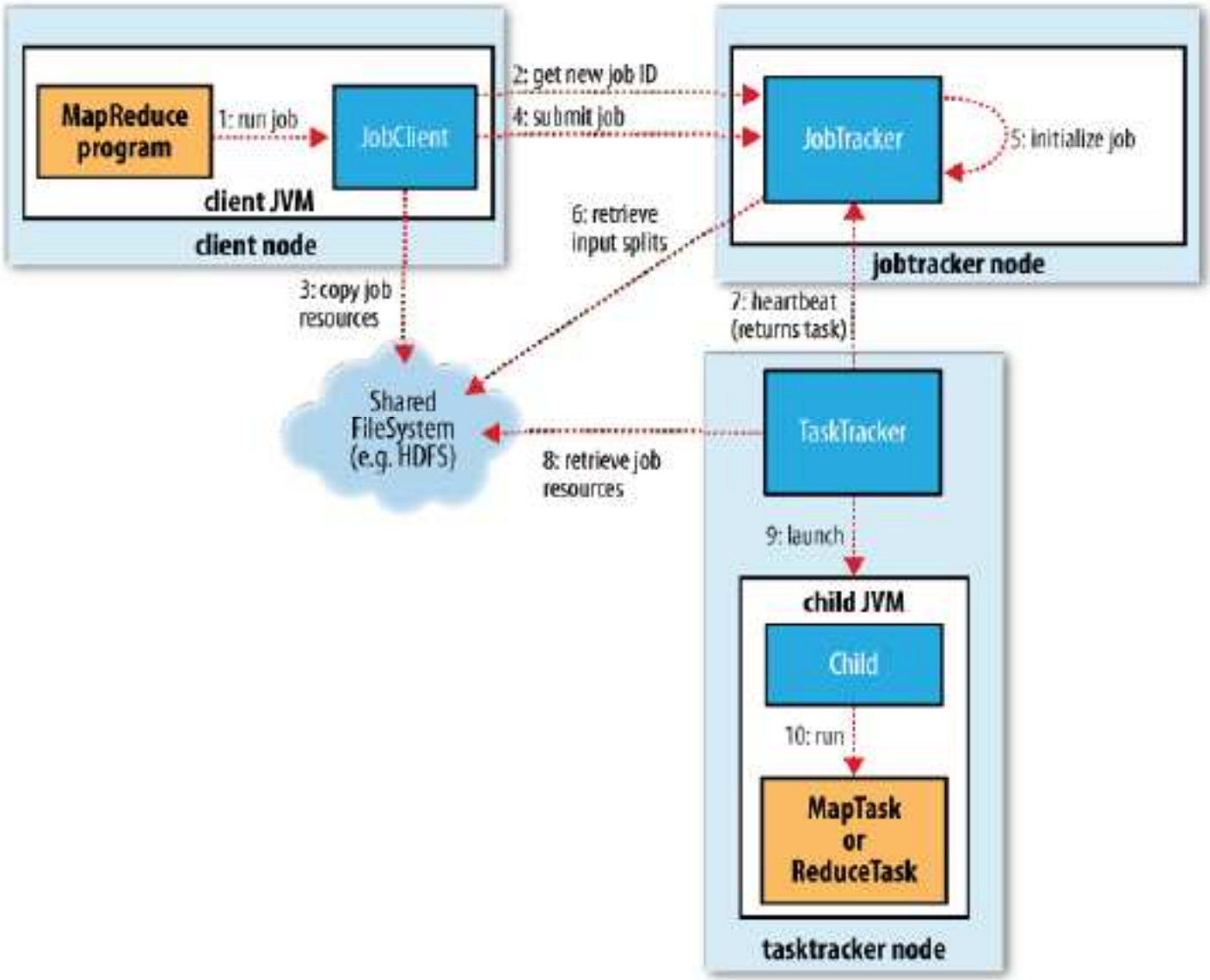
MR Dataflow of the example

- Each document is split into words
- Each word is counted by Map function using the word as the result key
- The framework puts all the pairs together with the same key
- The Reduce function uses the output from Map function as input
- The Reduce function sum all the inputs to find the total appearance of the word

Dataflow in Hadoop MR

- MR *job* splits the input dataset into chunks which are processed in a parallel manner
- The framework sorts the outputs of the maps
- The outputs of the maps becomes the input for the reduce tasks
- The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks
- The MR framework consists of a master (JobTracker) and a slave (TaskTracker) per node.
- The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks.
- The slaves execute the tasks as directed by the master.


Copy phase
"Sort" phase
Reduce phase
map task
reduce task
partition, sort and spill to disk
fetch
buffer in memory
map
input split
merge on disk
partitions
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces


MapReduce program
1: run job
JobClient
client JVM
client node
2: get new job ID
4: submit job
JobTracker
5: initialize job
jobtracker node
6: retrieve input splits
3: copy job resources
7: heartbeat (returns task)
Shared FileSystem (e.g. HDFS)
8: retrieve job resources
TaskTracker
9: launch
child JVM
Child
10: run
MapTask or ReduceTask
tasktracker node

Hadoop Streaming

- Is a utility that allows creating and running Map/Reduce jobs with any executable or script as the mapper and/or the reducer

```
hadoop jar ${HADOOP_HOME}/contrib/
streaming/hadoop-*-streaming.jar \
    -input /logs/access.log \
    -output /my/out/dir \
    -mapper /path/to/mapper.groovy \
    -reducer /path/to/reducer.groovy
```

Hadoop Pipes

- The name of the C++ interface to Hadoop MapReduce
- Pipes uses sockets as the channel which allows the tasktracker to communicate with the process running the C++ map or reduce function

```
hadoop pipes \
  -Dhadoop.pipes.java.recordreader=true \
  -Dhadoop.pipes.java.recordwriter=true \
  -input /path/to/input \
  -output /my/output/dir \
  -program bin/annihilate_spammers
```

Hadoop Projects

- **Avro™:**
  - A remote procedure call and data serialization system. It provides a serialization format for data, and wire format for communication between hadoop nodes and from client programs to hadoop services.
- **Cassandra™:**
  - A distributed database management system that provides decentralized structured data storage system for large amount of data.
- **Chukwa™:**
  - A data collection system for monitoring, managing, and analyzing large distributed systems.
- **HBase™:**
  - A distributed database that supports compression algorithms that can be used for data storage. It is used for random realtime read/write access to large data.

Hadoop Projects

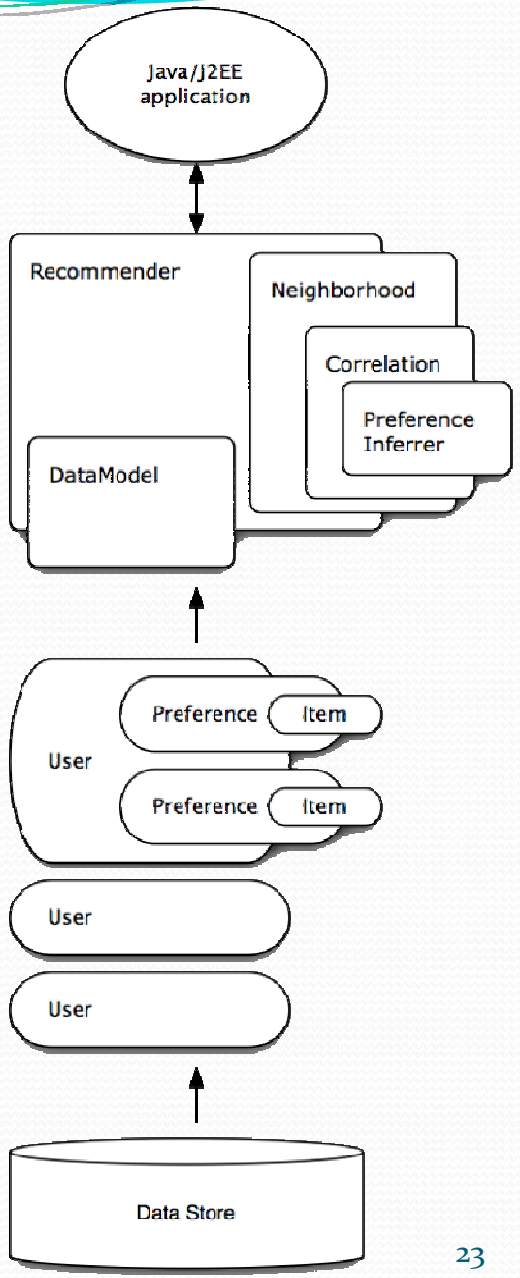
- **Hive™:**
  - A data warehouse infrastructure that provides data summarization, querying, and analysis. It provides HiveQL language which is like SQL language for interacting with data.
- **Mahout™:**
  - A distributed machine learning tool and data mining library for large data. The implemented algorithms in the library are for collaborative filtering, clustering, classification.
- **Pig™:**
  - A platform for Pig Latin, which is a high-level programming language used for parallel computation.
- **ZooKeeper™:**
  - A coordination service that provides configuration service, synchronization service, and naming registry for large distributed applications.

Mahout [11, 15]

- Mahout is a Machine Learning Library
- Mahout is a framework for developing, testing, and deploying mining algorithms
- Mahout is scalable to large datasets
- Mahout supports:
  - Recommendation mining (Collaborative Filtering - CF)
    - It takes user's preferences for items and returns estimated preferences for other items (e.g. Amazon, Netflix, or Facebook)
  - Clustering
    - It takes e.g. documents and groups them according to their topics (e.g. Google News, or Search Engines)
  - Classification
    - It learns specific categories from existing classified documents and assigns documents to correct categories (e.g. Yahoo! Mail, or Playlists in Apple's iTunes)

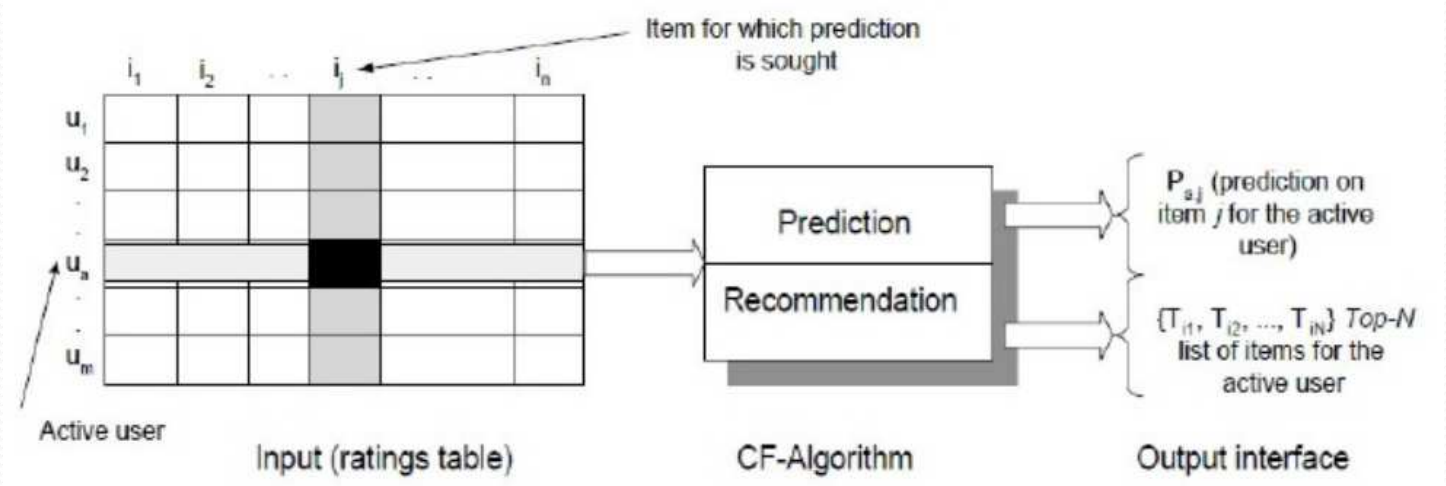
Recommender Engine

- The recommender engine contains user-based and item-based recommenders.
- User-based recommender architecture includes:
  - DataModel
  - UserSimilarity
  - UserNeighborhood
  - Recommender
- Item-based recommender system is similar except:
  - PreferenceInferrers
  - Neighborhood algorithms.



User-based CF Algorithm

- Provides item prediction for a user based on other users' common opinions (e.g. users ratings)
- User Rating scenario [9]
  - a collection of m users $U = \{u1, u2, \cdots, um\}$
  - a collection of n items $I = \{i1, i2, \cdots, in\}$
  - The items i and j are given rating scores of 1-5 by users u
  - The relationships between users and items are modeled as two-way matrix
  - each row corresponds to a user and each column an item
  - The system determines a set of users as neighbors who have same preferences to target user.
  - The system predicts a list of top-N items to target user.



The user rating data and CF process [8]

Item-based CF Algorithm

- Calculates similarity between different items based on user-item rating
- Computes prediction score based on calculated similarity scores
- Measures used to compute similarity between two items:
  - Correlation-based
  - Adjusted cosine-based

Correlation-based Similarity

- This measure computes similarity two items *i* and *j*
- Uses *Pearson* correlation $corr_{ij}$ coefficient.
- For user set *U* the correlation coefficient is:

$$sim(i,j) = corr_{ij} = \frac{\sum_{u \in U} (R_{u,i} - \overline{R_i})(R_{u,j} - \overline{R_j})}{\sqrt{\sum_{u \in U} (R_{u,i} - \overline{R_i})^2 \sum_{u \in U} (R_{u,j} - \overline{R_j})^2}}$$

- Where:
- $R_{u,i}$ is the rating score on item *i* for user *u*
- $\overline{R_i}$ is the average rating of i^{th} item

Adjusted Cosine Similarity

- In this measure two items are treated as two vectors in the form of *m*-dimensional user space
- The similarity between two items *i* and *j* is the cosine of the angle between two vectors
- An adjustment have been made to the original formula to calculate the difference in rating score between different users

$$sim(i,j) = \frac{\sum_{u \in U} \left(R_{u,i} - \overline{R_u}\right)\left(R_{u,j} - \overline{R_u}\right)}{\sqrt{\sum_{u \in U} \left(R_{u,i} - \overline{R_u}\right)^2 \sum_{u \in U} \left(R_{u,j} - \overline{R_u}\right)^2}}$$

- $\overline{R_u}$ is the average score of the u^{th} user's rating

Prediction Computation

- This computation is done after computing the similarity
- Two method have been proposed for this computation which are weighted sum and regression

- *Weighted Sum*
  - The final rating score of an item *i* for user *u* is the average sum of the total rating scores of items that are similar to item *i*

Prediction Computation (cont.)

- Each rating is weighted by the corresponding similarity s_{ij} between item i and j.

$$P_{u,i} = \frac{\sum_{j \in I_i} \left(sim(i,j) \times R_{u,j} \right)}{\sum_{j \in I_i} \left(|sim(i,j)| \right)}$$

- Where:
  - I_i is the set items that are similar to item i
  - || denotes the size of the set

Prediction Computation (cont.)

- *Regression*
  - This model is a linear regression model which predicts one variable from one or more other variables. [9]

$$\overline{R'_j} = \alpha \overline{R_i} + \beta + \varepsilon$$

  - Where:
    - Ri is the target item
    - Rj is the similar item
    - α and β are empirically determined
    - ε is the regression error

Recommender Implementation

- The input to Mahout recommenders is a set of user ID, item ID, and preference value tuples.

```
PreferenceArray user1Prefs = new GenericUserPreferenceArray(2);

user1Prefs.setUserID(0, 1L);

user1Prefs.setItemID(0, 101L);
user1Prefs.setValue(0, 2.0f);

user1Prefs.setItemID(1, 102L);
user1Prefs.setValue(1, 3.0f);

Preference pref = user1Prefs.get(1);
```

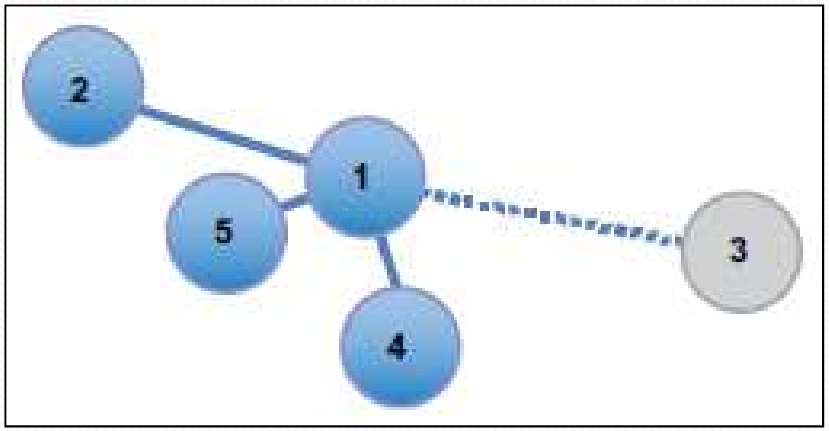
Sets user ID for preferences

Expresses preferences

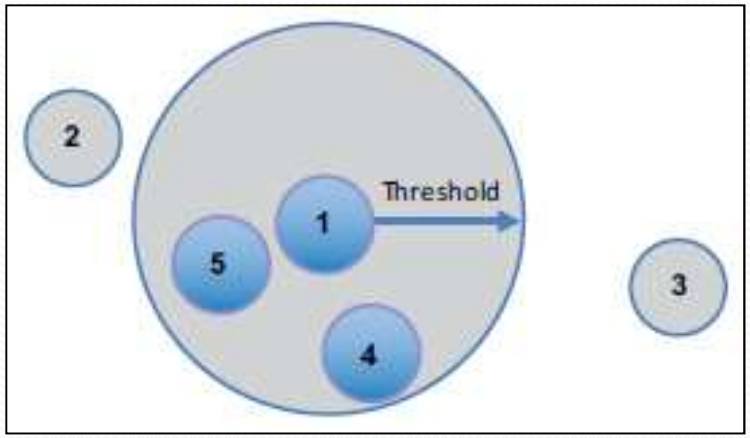
Materializes Preference for item 102

Recommender Algorithm

```
for every other user w
  compute a similarity s between u and w
  retain the top users, ranked by similarity, as a neighborhood n
for every item i that some user in n has a preference for,
      but that u has no preference for yet
  for every other user v in n that has a preference for i
    compute a similarity s between u and v
    incorporate v's preference for i, weighted by s, into a running average
```



An illustration of defining a neighborhood of most similar users by picking a fixed number of closest neighbors.



Defining a neighborhood of most similar users with a similarity threshold

Exploring similarity metrics

- Pearson correlation–based similarity

	Item 101	Item 102	Item 103	Correlation with user 1
User 1	5.0	3.0	2.5	1.000
User 2	2.0	2.5	5.0	-0.764
User 3	2.5	-	-	-
User 4	5.0	-	3.0	1.000
User 5	4.0	3.0	2.0	0.945

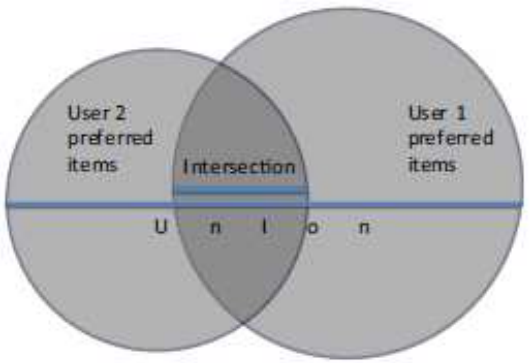
The Pearson correlation between user 1 and other users based on the three items that user 1 has in common with the others.

- Defining similarity by Euclidean distance

	Item 101	Item 102	Item 103	Distance	Similarity to user 1
User 1	5.0	3.0	2.5	0.000	1.000
User 2	2.0	2.5	5.0	3.937	0.203
User 3	2.5	-	-	2.500	0.286
User 4	5.0	-	3.0	0.500	0.667
User 5	4.0	3.0	2.0	1.118	0.472

The Euclidean distance between user 1 and other users, and the resulting similarity scores

- Ignoring preference values in similarity with the Tanimoto coefficient



The Tanimoto coefficient is the ratio of the size of the intersection, or overlap, in two users' preferred items (the dark area), to the union of the users' preferred items (the dark and light areas together).1

	Item 101	Item 102	Item 103	Item 104	Item 105	Item 106	Item 107	Similarity to user 1
User 1	X	X	X					1.0
User 2	X	X	X	X				0.75
User 3	X			X	X		X	0.17
User 4	X		X	X		X		0.4
User 5	X	X	X	X	X	X		0.5

The similarity values between user 1 and other users, computed using the Tanimoto coefficient. Note that preference values themselves are omitted, because they aren't used in the computation.

Distributing Recommendation Computations

- The shape of a MapReduce computation:
  - Input is in the form of many key-value (K1,V1) pairs.
  - A map function is applied to each (K1,V1) pair, which results in zero or more key-value pairs of a different kind (K2,V2).
  - All V2 for each K2 are combined.
  - A reduce function is called for each K2 and all its associated V2, which results in zero or more key-value pairs of a different kind (K3,V3).
  - Output the result back to HDFS.

Distributing Recommendation Computations

- Translating to MapReduce: generating user vectors
  - The computation begins with the links data file as input
  - Input files are treated as pairs. For example, 98955: 590 22 9059
  - Each line is parsed into a user ID and several item IDs by a map function. For example, 98955 / 590
  - The framework collects all item IDs that were mapped to each user ID together.
  - A reduce function constructors a Vector from all item IDs for the user, and outputs the user ID mapped to the user's preference vector. All values in this vector are 0 or 1. For example, 98955 / [590:1.0, 22:1.0, 9059:1.0]

```
public class WikipediaToItemPrefsMapper
    extends Mapper<LongWritable,Text,VarLongWritable,VarLongWritable> {

  private static final Pattern NUMBERS = Pattern.compile("(\\d+)");

  public void map(LongWritable key,
                  Text value,
                  Context context)
      throws IOException, InterruptedException {
    String line = value.toString();
    Matcher m = NUMBERS.matcher(line);
    m.find();
    VarLongWritable userID =
        new VarLongWritable(Long.parseLong(m.group()));
    VarLongWritable itemID = new VarLongWritable();
    while (m.find()) {
      itemID.set(Long.parseLong(m.group()));
      context.write(userID, itemID);
    }
  }
}
```

Locate user ID

Emit user-item pair for each item ID

A mapper that parses Wikipedia link files

```
public class WikipediaToUserVectorReducer extends
  Reducer<VarLongWritable,VarLongWritable,VarLongWritable,VectorWritable> {
  public void reduce(VarLongWritable userID,
                     Iterable<VarLongWritable> itemPrefs,
                     Context context)
      throws IOException, InterruptedException {
    Vector userVector = new RandomAccessSparseVector(
        Integer.MAX_VALUE, 100);
    for (VarLongWritable itemPref : itemPrefs) {
      userVector.set((int)itemPref.get(), 1.0f);
    }
    context.write(userID, new VectorWritable(userVector));
  }
}
```

Iterate over item-preference pairs for user

Set dimension "item ID" to item's preference value

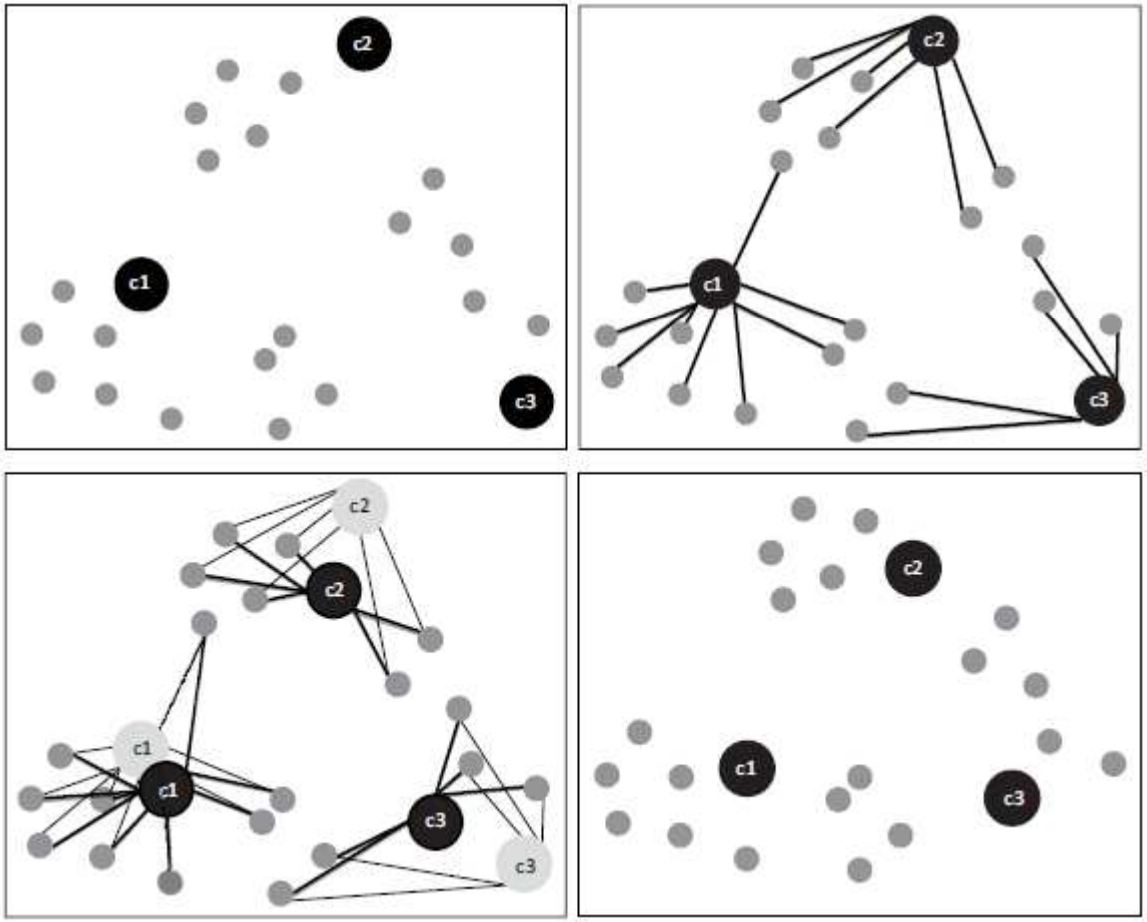
Reducer which produces Vectors from a user's item preferences

Clustering

- Groups similar pieces of data together into a set or cluster.
- Implemented Algorithms:
  - K-Means
  - Fuzzy K-Means
  - Canopy
  - Dirichlet Process

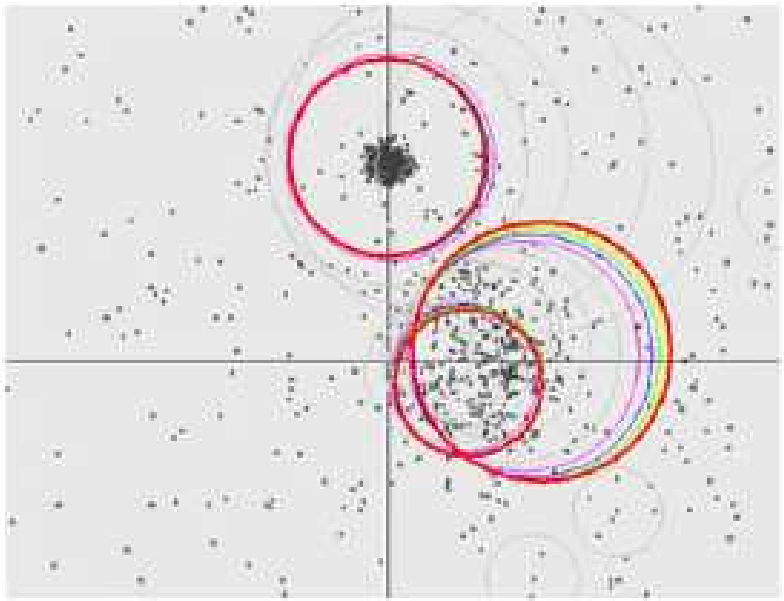
K-Means Clustering

- Assign the number of clusters k = n
- Begins with n number of random points as centroids
- In Map process, the algorithm assigns each point to the nearest cluster
- In Reduce process, the algorithm calculates the mean to produce new location for the centroids
- The processes are done iteratively until the centroids come to their final positions.



Fuzzy K-Means Clustering

- Fuzzy k-means tries to generate overlapping clusters from the dataset
- Point can belong to more than one cluster with close value
- The close value is according to the distance from the point to the centroid



Canopy Clustering

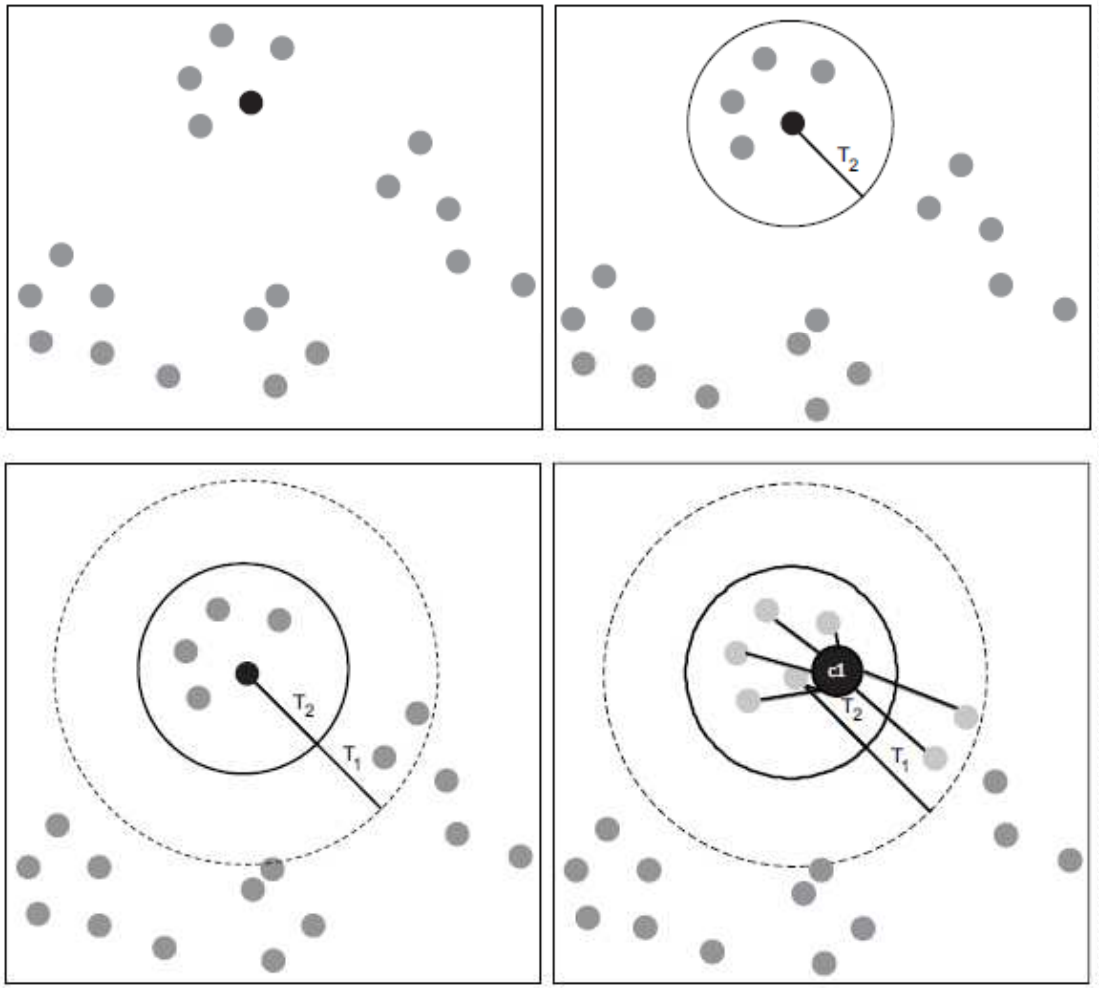
- An enclosure of points, or a cluster.
- Divides the input into overlapping clusters known as canopies.
- Estimates the optimal number of clusters without even specifying the number of clusters *k*.
- Estimates the approximate cluster centroids (or canopy centroids) using two distance thresholds with T1 > T2.

Canopy process

1. Begins with an empty list of canopies and a dataset of points
2. Creates canopies by iterating over the dataset
3. Removes a point from the dataset and adds a canopy to the list with that point as the center
4. Iterates through the rest of points in the dataset
5. Calculates the distances to all the canopy centers in the list
   - If the distance between the point and any canopy center is within T1, the point is added into that canopy.
   - If the distance is within T2, the point is removed from the list to prevent forming a new canopy in the next loops.
6. Repeats this process until the list is empty.

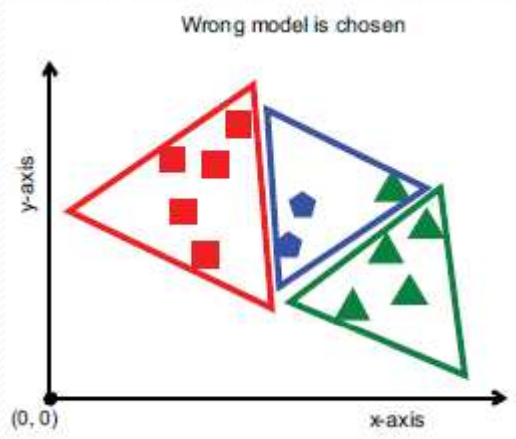
Canopy process (cont.)

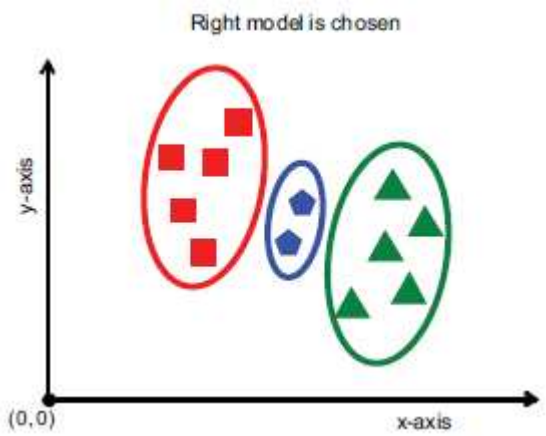
- Mahout:
  - A mapper performs canopy clustering on the points in its input set and outputs its canopies' centers
  - The reducer clusters the centers to produce the final canopy centers
  - The points are then clustered into final canopies



Dirichlet Clustering

- Dirichlet is a family of probability distributions
- Performs a mixture modeling
- The data points are concentrated and distributed within area shapes
- The models are made to fit the dataset
- The right model will fit the data better
- The right model indicates the number of clusters in the dataset





Classification

- Classifies data based on the training set and the values in a classifying attribute
- Uses the data in classifying new data

- Implemented Algorithms in Mahout:
  - Bayesian
    - Naïve Bayes
      - Assumes the presence/absence of a feature of a class is unrelated to the presence/absence of any other feature

What is Hive?

- Hive is a data warehouse system for Hadoop that facilitates easy data summarization and the analysis of large datasets stored in Hadoop file systems. [2]
- Hive Query Language (HQL) is like Structured Query Language (SQL)
- HQL in translated into a series of MapReduce jobs

Hive Example

- Basic map reduce example – count frequencies of each word!
  - 'I' - 3
  - 'data' - 2
  - 'mining' - 2
  - 'awesome' - 1
  - …
- Input: 270 twitter tweets
- sample_tweets.txt
  - T 2009-06-08 21:49:37
  - U http://twitter.com/evion
  - W I think data mining is awesome!
  - T 2009-06-08 21:49:37
  - U http://twitter.com/hyungjin
  - W I don't think so. I don't like data mining

Hive Example

- **Create table from raw data file**
  - table raw_tweets
- **Parse data file to match our format, and save to new table**
  - parser.py
  - table tweets_test_parsed
- **Run map/reduce**
  - mappr.py, reducer.py
- **Save result to new table**
  - table word_count
- **Find top 10 most frequent words from word_count table.**

Powered by Hive

RHive

- RHive is an R extension to facilitate analyzing data in distributed computing via HIVE query
- RHive provides using HQL and R objects

Conclusion

References

- [1] Apache Hadoop. Available at http://hadoop.apache.org
- [2] Apache HBase. Available at http://hbase.apache.org
- [3] Apache HDFS. Available at http://hadoop.apache.org/hdfs
- [4] Apache Hive. Available at http://hive.apache.org
- [5] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu and Raghotham Murthy. Hive – A Petabyte Scale Data Warehouse Using Hadoop.
- [6] David Stockburger. Introductory Statistics: Concepts, Models, and Applications. http://www.psychstat.missouristate.edu/introbook/sbkoo.htm, 1996
- [7] D. Krishna. "Big Data". http://www.irmac.ca/1011/Big%20Data%20v2.1.pdf
- [8] Google Code University. "Introduction to Parallel Programming and MapReduce". http://code.google.com/edu/parallel/mapreduce-tutorial.html
- [9] Guandong Xu, Yanchun Zhang, Lin Li. Web Mining and Social Networking, Web Mining and Recommendation Systems. http://www.scribd.com/doc/51482678/14/Correlation-based-Similarity. pages 169-172, 2011

- [10] J. Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. 2004
- [11] Mahout Wiki, https://cwiki.apache.org/MAHOUT/mahout-wiki.html
- [12] MapReduce Tutorial, http://hadoop.apache.org/common/docs/current/mapred_tutorial.html
- [13] N. Marz, and S. Ritchie. “Big Data: Principles and best practice of scalable realtime data systems”. 2011
- [14] NexR. Package ‘RHive’. 2012
- [15] S. Owen, R. Anil, T. Dunning, and E. Friedman. “Mahout in Action”. 2012
- [16] T. White. “Hadoop: The Definitive Guide”. 2009

Questions

Thank you