EE 542
Lecture 10: Main Uses of Cloud

Internet and Cloud Computing

Young Cho
Department of Electrical Engineering
University of Southern California

Industry

- At Google:
  - Index building for Google Search
  - Article clustering for Google News
  - Statistical machine translation
- At Yahoo!:
  - Index building for Yahoo! Search
  - Spam detection for Yahoo! Mail
- At Facebook:
  - Data mining
  - Ad optimization
  - Spam detection

Research

- Analyzing Wikipedia conflicts (PARC)
- Natural language processing (CMU)
- Climate simulation (Washington)
- Bioinformatics (Maryland)
- Particle physics (Nebraska)
- Cancer Research

Cloud Goals

- Scalability to large data volumes
  - Scan 100 TB on 1 node @ 50 MB/s = 24 days
  - Scan on 1000-node cluster = 35 minutes
- Cost-efficiency
  - Commodity nodes (cheap, but unreliable)
  - Commodity network (low bandwidth)
  - Automatic fault-tolerance (fewer admins)
  - Easy to use (fewer programmers)

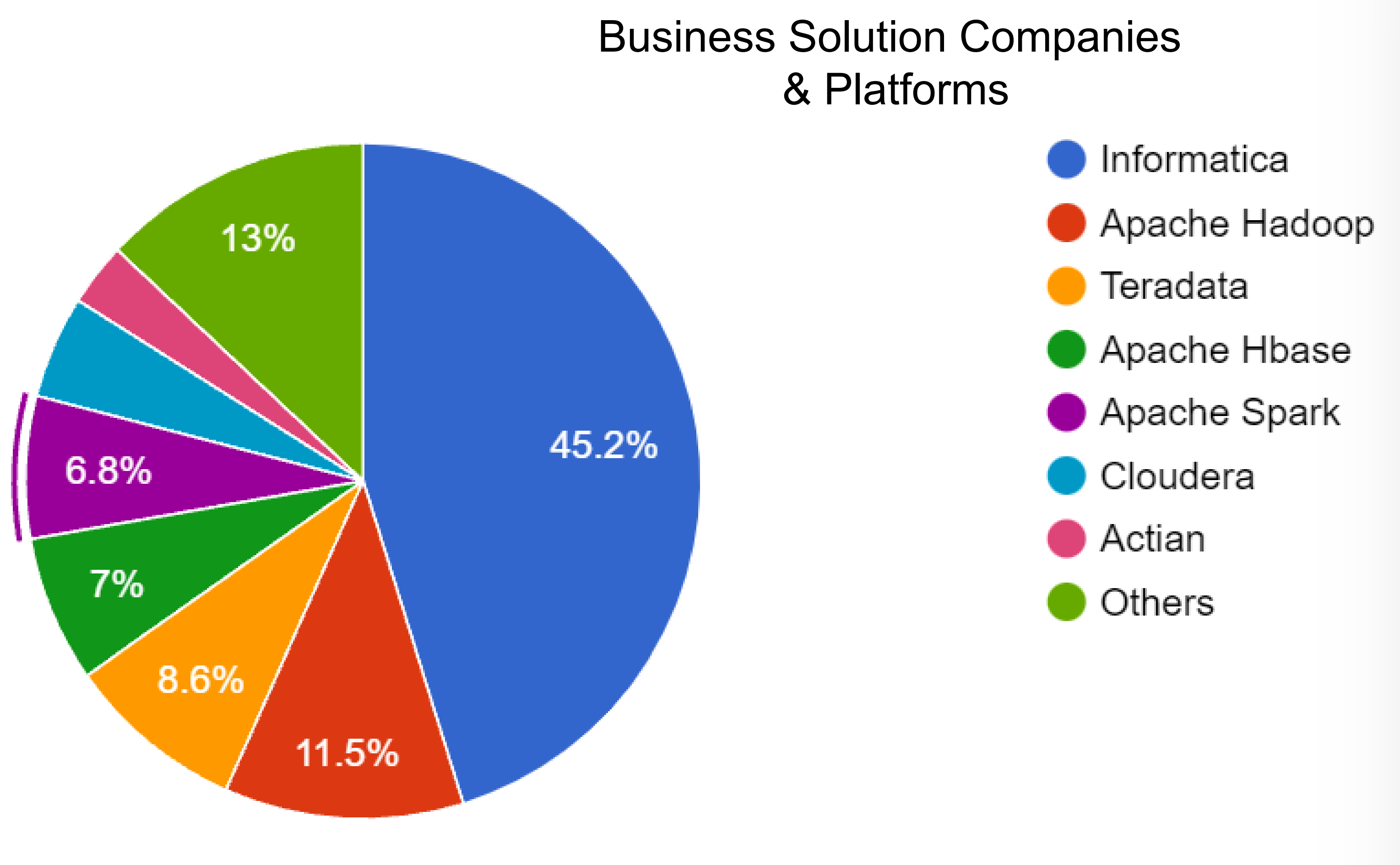
Challenges of Cloud Environment

- Cheap nodes fail, especially when you have many
  - Mean time between failures for 1 node = 3 years
  - MTBF for 1000 nodes = 1 day
  - **Solution:** Build fault tolerance into system

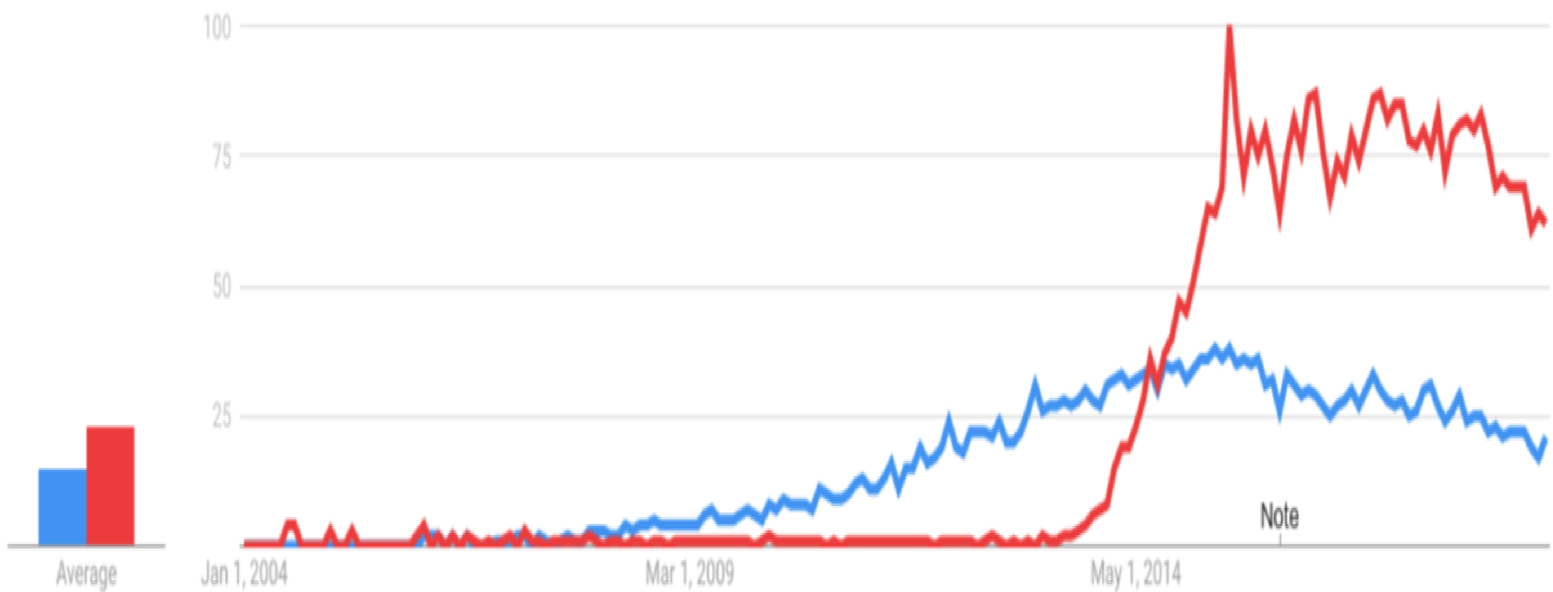
- Commodity network = low bandwidth
  - **Solution:** Push computation to the data

- Programming distributed systems is hard
  - **Solution:** Restricted programming model: users write data-parallel "map" and "reduce" functions, system handles work distribution and failures

Bigdata Market Share



Hadoop(B) vs. Spark(R)



What is Hadoop?

- Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.
- It is a flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.

- Designed to answer the question: "**How to process big data with reasonable cost and time?**"

Google Origins

2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

→ hadoop HDFS

2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

→ hadoop MapReduce

2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

igtable is a distributed storage system for managing
tured data that is designed to scale to a very large
petabytes of data across thousands of commodity
rs. Many projects at Google store data in Bigtable,
ding web indexing, Google Earth, and Google Fi-
e. These applications place very different demands
igtable, both in terms of data size (from URLs to

achieved scalability and high performance, but Big
provides a different interface than such systems. Big
does not support a full relational data model; inste
provides clients with a simple data model that sup
dynamic control over data layout and format, an
lows clients to reason about the locality properties o
data represented in the underlying storage. Data i
dexed using row and column names that can be arbi
strings. Bigtable also treats data as uninterpreted str

→ APACHE HBASE

Hadoop Milestones

- **2008 - Hadoop Wins Terabyte Sort Benchmark (**sorted 1 terabyte of data in 209 seconds, compared to previous record of 297 seconds)
- 2009 - Avro and Chukwa became new members of Hadoop Framework family
- 2010 - Hadoop's Hbase, Hive and Pig subprojects completed, adding more computational power to Hadoop framework
- **2011 - ZooKeeper Completed**
- **2013 - Hadoop 1.1.2 and Hadoop 2.0.3 alpha.**

Hadoop Use

- Hadoop is in use to handle big data:
  - Yahoo!'s Search Webmap runs on 10,000 core Linux cluster and powers Yahoo! Web search
  - FB's Hadoop cluster hosts 100+ PB of data (July, 2012) & growing at ½ PB/day (Nov, 2012)
  - Amazon and Netflix
  - NY Times was dynamically generating PDFs of articles from 1851-1922
    - Wanted to pre-generate & statically serve articles to improve performance
    - Using Hadoop + MapReduce running on EC2 / S3, converted 4TB of TIFFs into 11 million PDF articles in 24 hrs
- Key Applications
  - Advertisement (Mining user behavior to generate recommendations)
  - Searches (group related documents)
  - Security (search for uncommon patterns)

Hadoop Concept (MapReduce)

- Programming model for data-intensive computing on commodity clusters
- Pioneered by Google
  - Processes 20 PB of data per day
- Popularized by Apache Hadoop project
  - Used by Yahoo!, Facebook, Amazon, …

MapReduce Programming Model

- Data type: key-value *records*

- Map function:

$$(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})$$

- Reduce function:

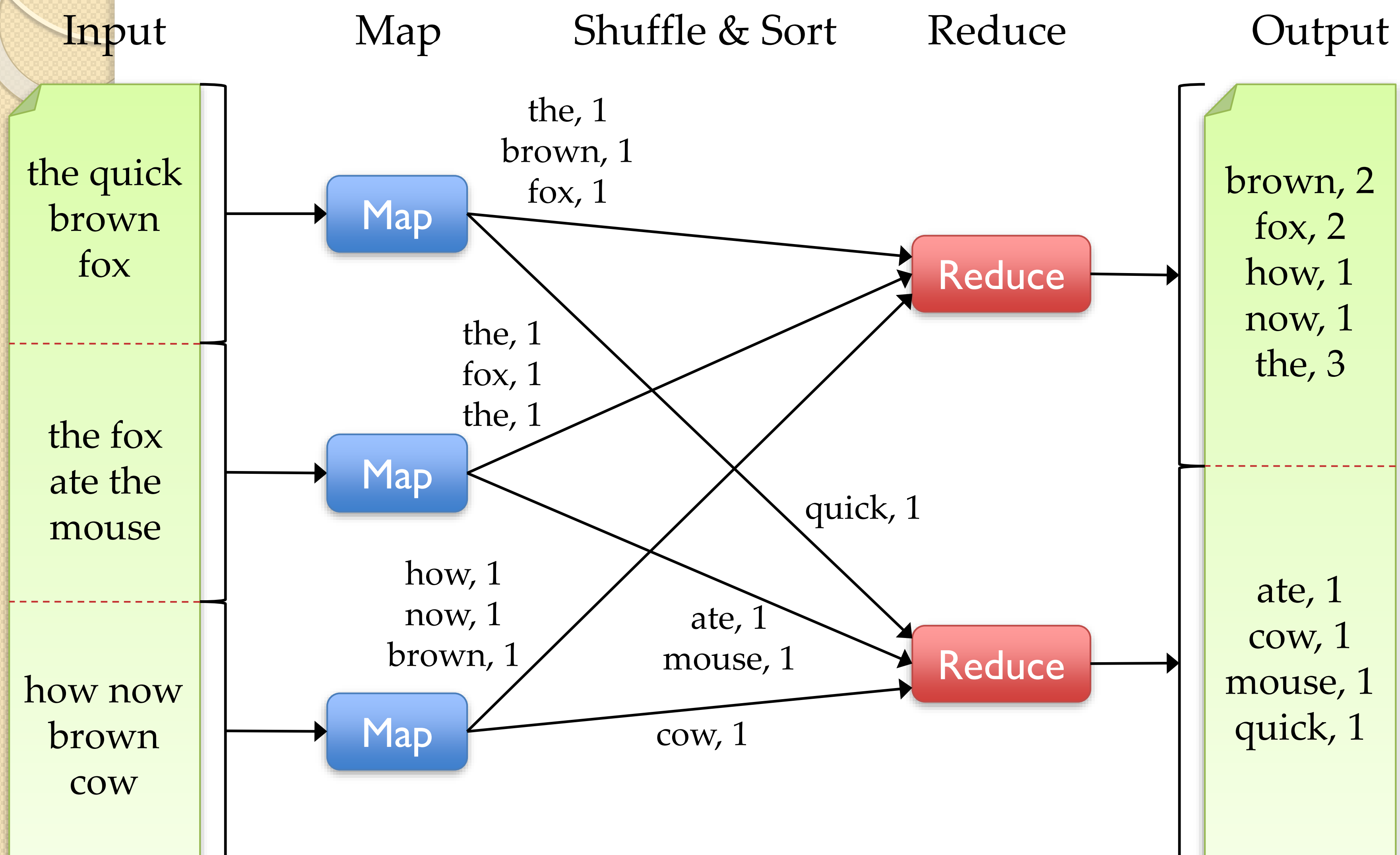
$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$

Example: Word Count

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)


def reducer(key, values):
    output(key, sum(values))
```

Word Count Execution



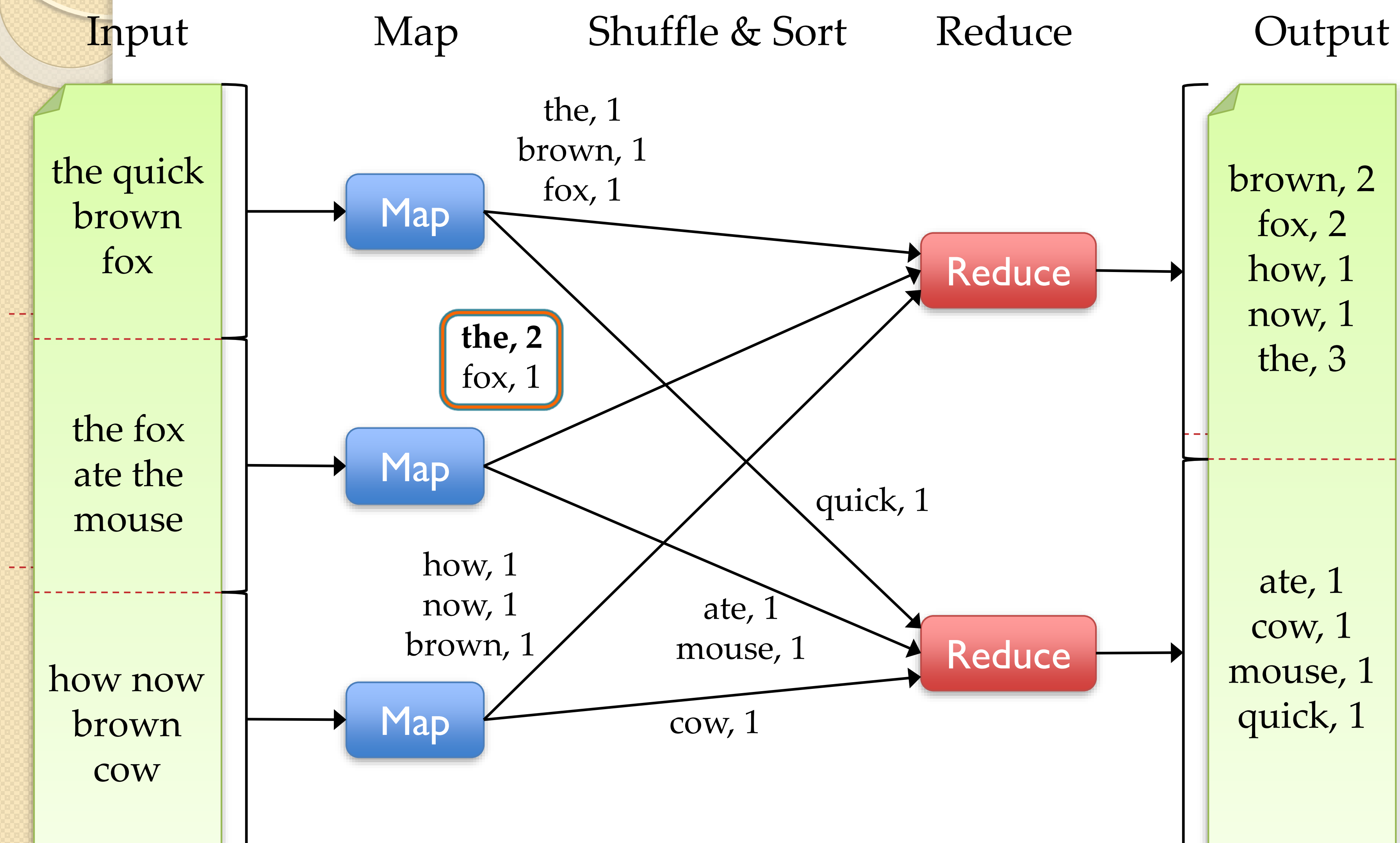
An Optimization: The Combiner

- Local reduce function for repeated keys produced by same map
- For associative ops. like sum, count, max
- Decreases amount of intermediate data

- Example: local counting for Word Count:

```
def combiner(key, values):
    output(key, sum(values))
```

Word Count with Combiner



1. Search

- **Input:** (lineNumber, line) records
- **Output:** lines matching a given pattern

- **Map:**

```
if(line matches pattern):
   output(line)
```

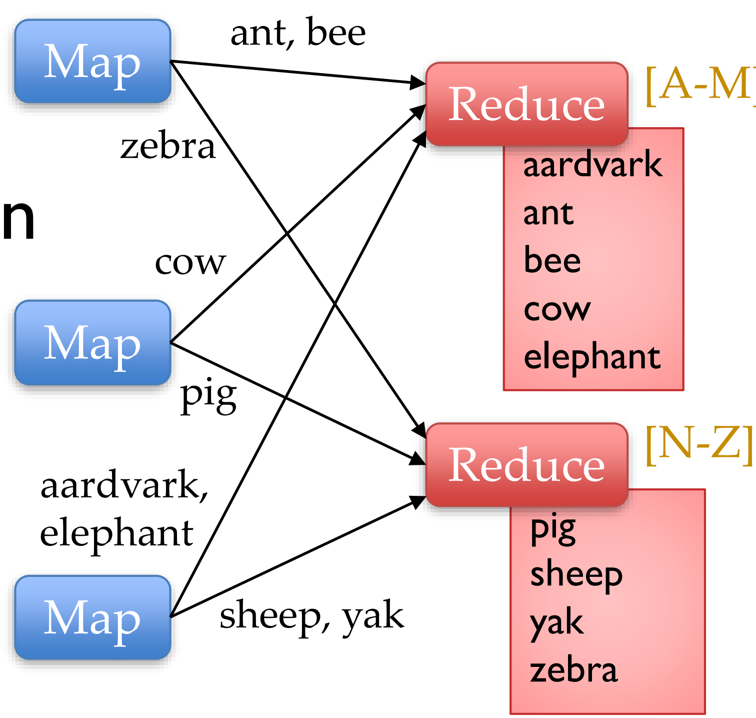
- **Reduce:** identity function
  - Alternative: no reducer (map-only job)

2. Sort

- **Input:** (key, value) records
- **Output:** same records, sorted by key

- **Map:** identity function
- **Reduce:** identify function

- **Trick:** Pick partitioning function p such that $k_1 < k_2 \Rightarrow p(k_1) < p(k_2)$



3. Inverted Index

- **Input:** (filename, text) records
- **Output:** list of files containing each word

- **Map:**

```
foreach word in text.split():
    output(word, filename)
```

- **Combine:** uniquify filenames for each word

- **Reduce:**

```
def reduce(word, filenames):
    output(word,sort(filenames))
```

Inverted Index Example

hamlet.txt

to be or
not to be

to, hamlet.txt
be, hamlet.txt
or, hamlet.txt
not, hamlet.txt

12th.txt

be not
afraid of
greatness

be, 12th.txt
not, 12th.txt
afraid, 12th.txt
of, 12th.txt
greatness, 12th.txt

afraid, (12th.txt)
be, (12th.txt, hamlet.txt)
greatness, (12th.txt)
not, (12th.txt, hamlet.txt)
of, (12th.txt)
or, (hamlet.txt)
to, (hamlet.txt)

4. Most Popular Words

- **Input:** (filename, text) records
- **Output:** the 100 words occurring in most files

- Two-stage solution:
  - **Job 1:**
    - Create inverted index, giving (word, list(file)) records
  - **Job 2:**
    - Map each (word, list(file)) to (count, word)
    - Sort these records by count as in sort job

- Optimizations:
  - Map to (word, 1) instead of (word, file) in Job 1
  - Estimate count distribution in advance by sampling

5. Numerical Integration

- **Input:** (start, end) records for sub-ranges to integrate
  - Can implement using custom InputFormat
- **Output:** integral of *f*(*x*) over entire range

- **Map:**

```
def map(start, end):
   sum = 0
   for(x = start; x < end; x += step):
      sum += f(x) * step
   output("", sum)
```

- **Reduce:**

```
def reduce(key, values):
   output(key, sum(values))
```

Word Count using Hadoop

Mapper.py:

```
import sys
for line in sys.stdin:
  for word in line.split():
    print(word.lower() + "\t" + 1)
```

Reducer.py:

```
import sys
counts = {}
for line in sys.stdin:
  word, count = line.split("\t")
    dict[word] = dict.get(word, 0) + int(count)
for word, count in counts:
  print(word.lower() + "\t" + 1)
```

Requirements at Facebook

- Design requirements:
  - Integrate display of email, SMS and chat messages between users
  - Strong control over who users
  - Stringent latency & uptime
- System requirements
  - High write throughput
  - Cheap, elastic storage
  - Low latency
  - High consistency
  - Disk-efficient sequential and random read

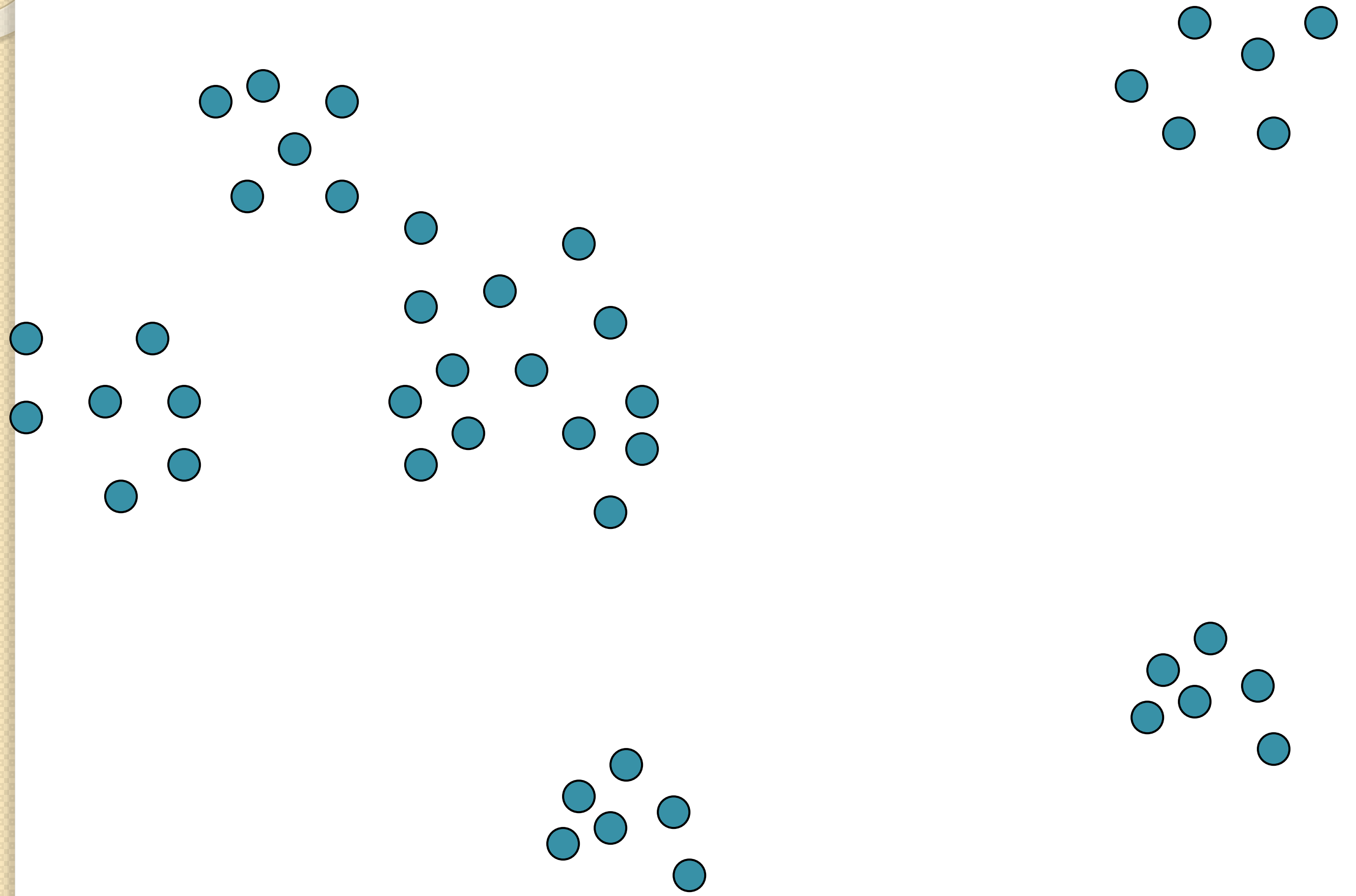
Hadoop Use at Facebook

- Classic alternatives
  - These requirements typically met using large MySQL cluster & caching tiers using Memcache
  - Content on HDFS could be loaded into MySQL or Memcached if needed by web tier
- Problems with previous solutions
  - MySQL has low random write throughput… BIG problem for messaging!
  - Difficult to scale MySQL clusters rapidly while maintaining performance
  - MySQL clusters have high management overhead, require more expensive hardware

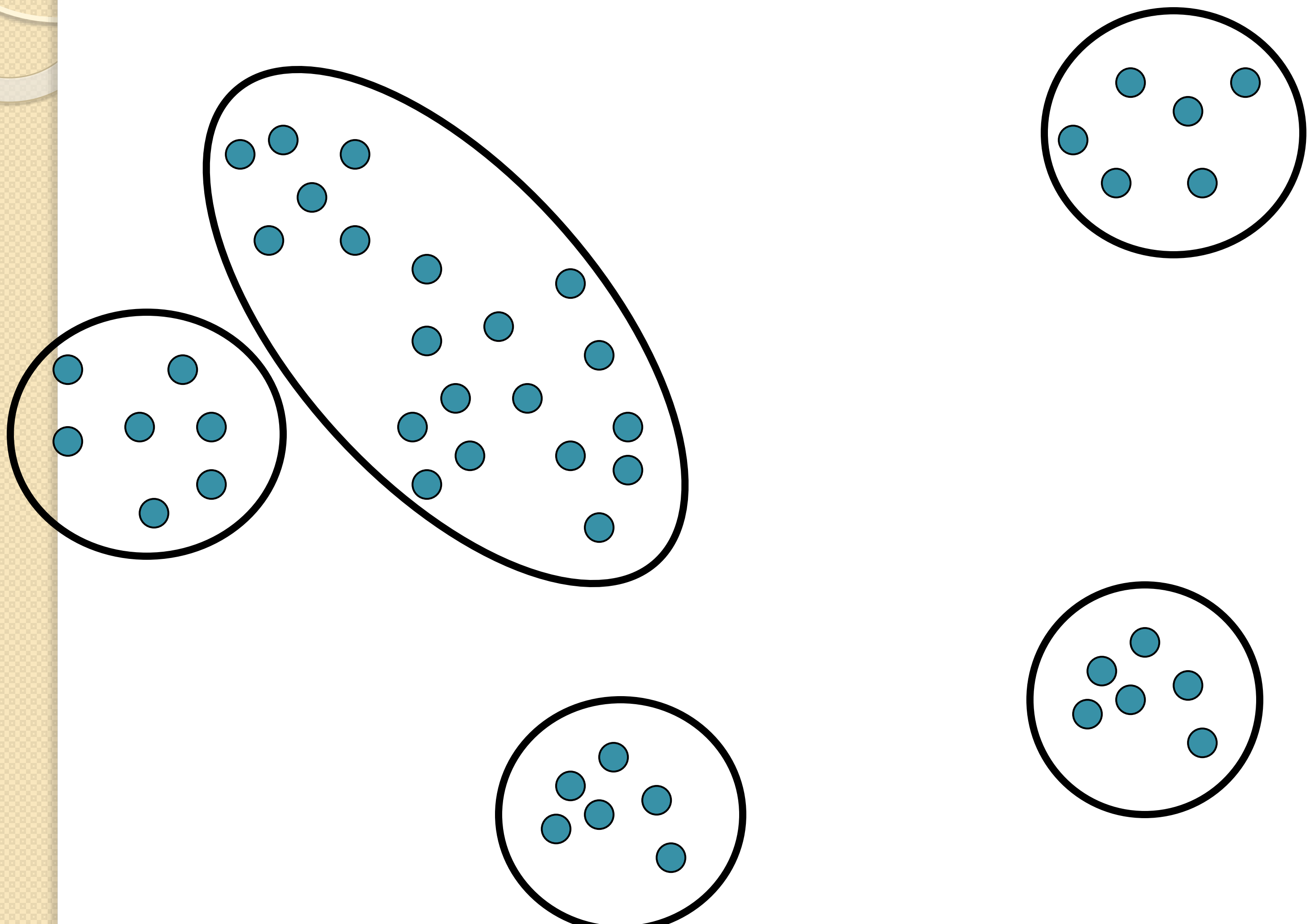
Hadoop Use at Facebook

- Hadoop + HBase as foundations
  - Improve & adapt HDFS and HBase to scale to FB's workload and operational considerations
  - NameNode is Single point of failure & failover times are at least 20 minutes
- Proprietary "AvatarNode"
  - Eliminates single point of failure makes HDFS safe to deploy even with 24/7 uptime requirement
  - Performance improvements for realtime workload: RPC timeout.
  - Rather fail fast and try a different DataNode

Clustering

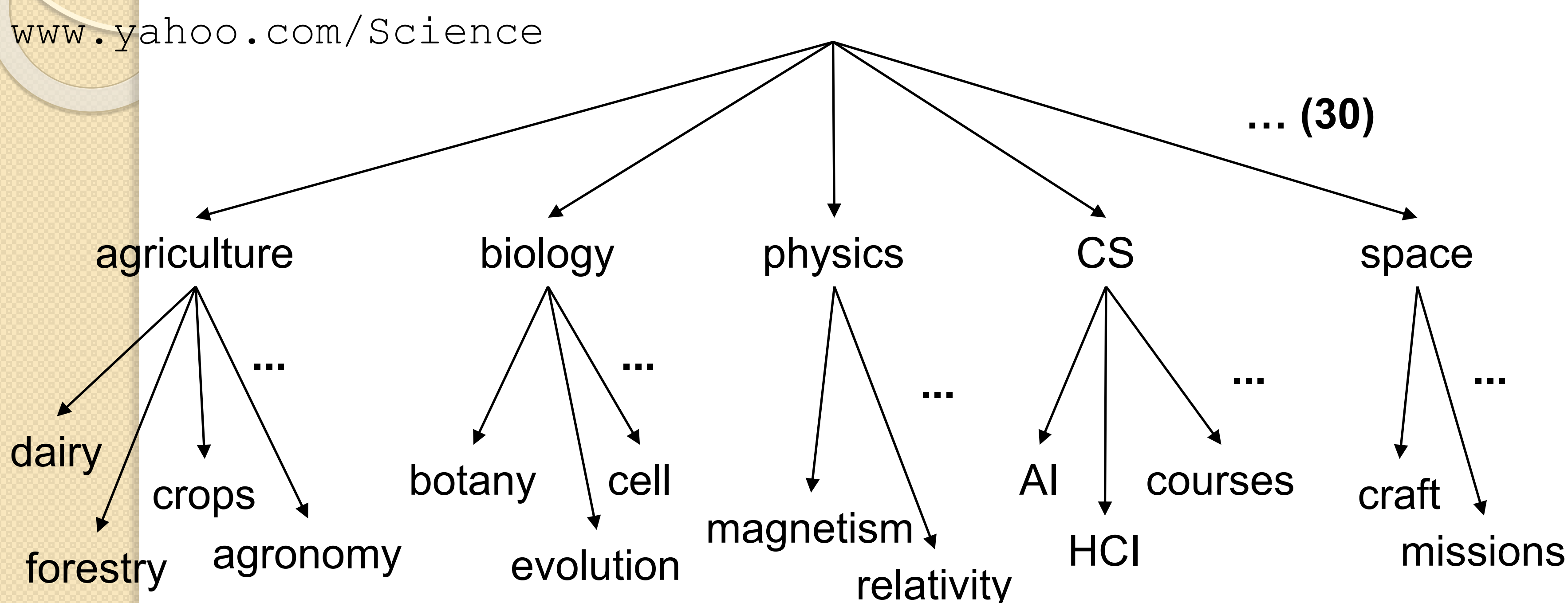


Clustering



Citation graph browsing

Clustering: Corpus browsing



Cluster Partitioning

- Iterative Partitioning
  - Training data set to learn a partition of the given data space
  - learning a partition on a data set to produce several non-empty clusters (usually, the number of clusters given in advance)
- Optimal partition
  - Minimizing the sum of squared distance to its "representative object" in each cluster

$$E = \Sigma_{k=1}^{K} \Sigma_{\mathbf{x} \in C_k} d^2(\mathbf{x}, \mathbf{m}_k)$$

e.g., Euclidean distance $d^2(\mathbf{x}, \mathbf{m}_k) = \sum_{n=1}^{N} (x_n - m_{kn})^2$

K-Means Clustering

- Given a *K*, find a partition of *K clusters* to optimize the chosen partitioning criterion (cost function)
  - global optimum: exhaustively search all partitions
- The *K-means* algorithm: a heuristic method
  - K-means algorithm (MacQueen'67): each cluster is represented by the centre of the cluster and the algorithm converges to stable centriods of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications.

K-means Algorithm

- Given the cluster number *K*, the *K-means* algorithm is carried out in three steps after initialization:

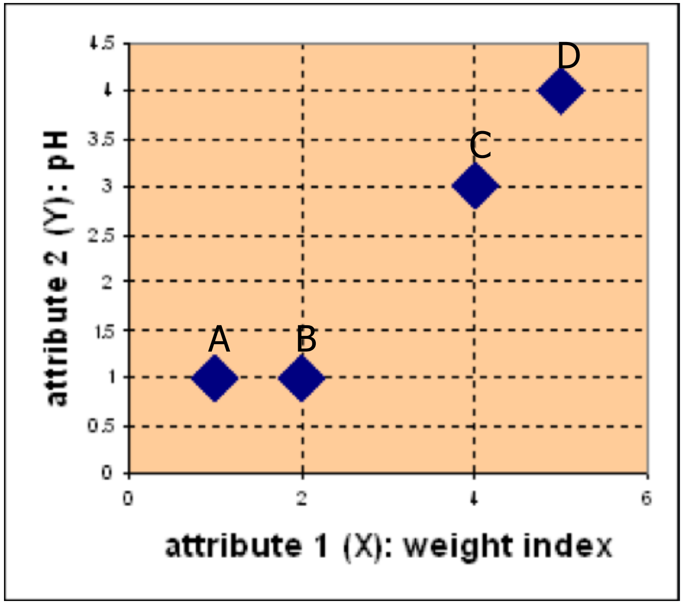
  Initialization: set seed points (randomly)

  1) Assign each object to the cluster of the nearest seed point measured with a specific distance metric
  2) Compute new seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., *mean point*, of the cluster)
  3) Go back to Step 1), stop when no more new assignment (i.e., membership in each cluster no longer changes)

K-Means Example

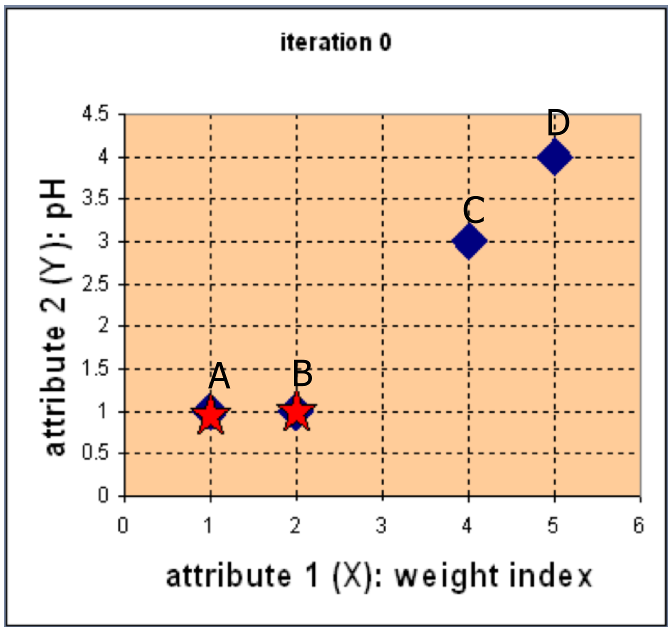
Suppose we have 4 types of medicines and each has two attributes (pH and weight index). Our goal is to group these objects into *K=2* group of medicine.

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



Example

- Step 1: Use initial seed points for partitioning



$$c_1 = A, c_2 = B$$

$$\mathbf{D}^0 = \begin{array}{cccc|l} \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4.24 \end{bmatrix} & \begin{array}{ll} \mathbf{c}_1 = (1,1) & group-1 \\ \mathbf{c}_2 = (2,1) & group-2 \end{array} \end{array}$$

$$\begin{array}{cccc} A & B & C & D \end{array}$$

$$\begin{array}{cc} \begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} & \begin{array}{l} X \\ Y \end{array} \end{array}$$

Euclidean distance

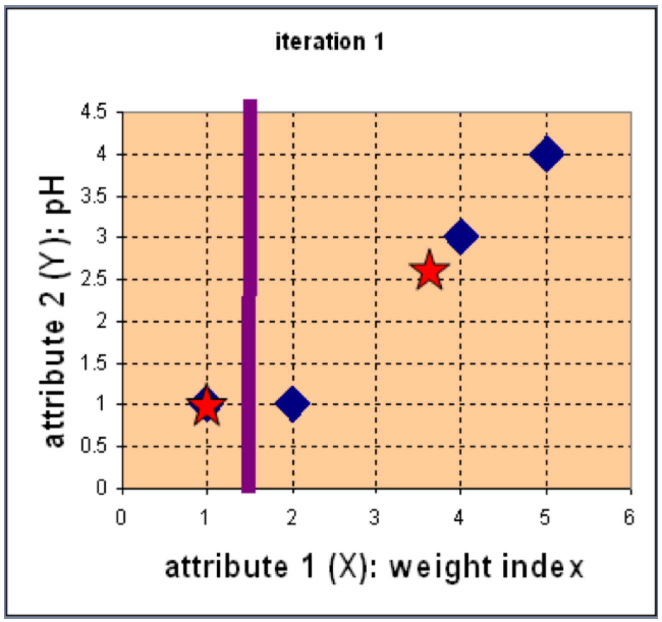
$$d(D, c_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5$$

$$d(D, c_2) = \sqrt{(5-2)^2 + (4-1)^2} = 4.24$$

Assign each object to the cluster with the nearest seed point

Example

- Step 2: Compute new centroids of the current partition



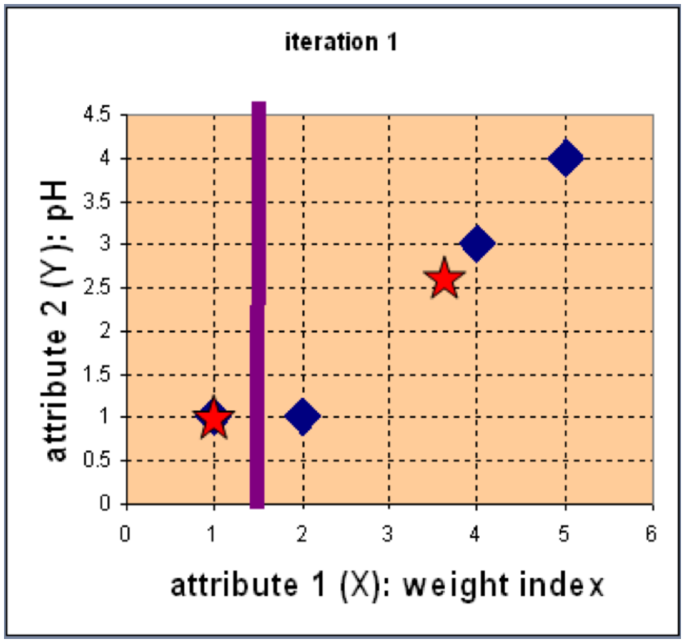
Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

$$c_1 = (1,\ 1)$$

$$c_2 = \left(\frac{2+4+5}{3},\ \frac{1+3+4}{3}\right)$$
$$= (\frac{11}{3},\ \frac{8}{3})$$

Example

- Step 2: Renew membership based on new centroids



Compute the distance of all objects to the new centroids

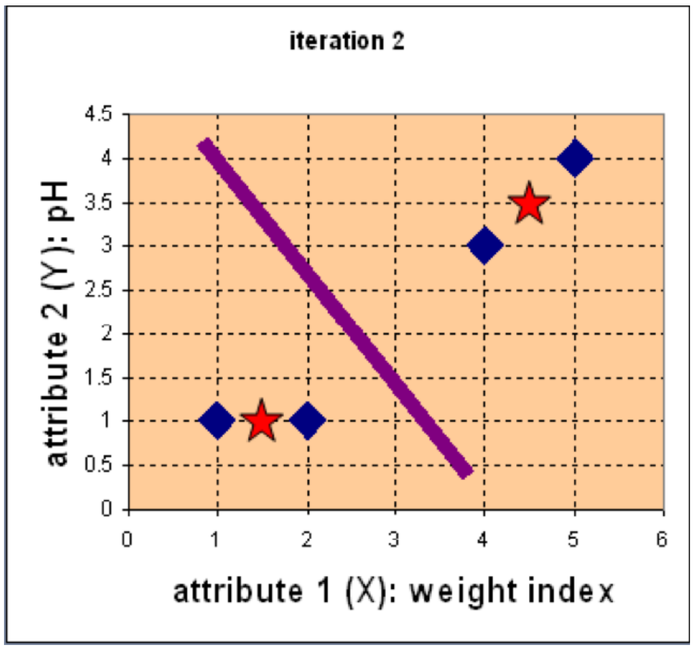
$$\mathbf{D}^1 = \begin{array}{c} \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{bmatrix} \\ \begin{matrix} A & \;\;B & \;\;C & \;\;D \end{matrix} \end{array} \begin{matrix} \mathbf{c}_1 = (1,1) & \mathit{group}-1 \\ \mathbf{c}_2 = (\frac{11}{3}, \frac{8}{3}) & \mathit{group}-2 \end{matrix}$$

$$\begin{array}{c|c} \begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} & \begin{matrix} X \\ Y \end{matrix} \end{array}$$

Assign the membership to objects

Example

- Step 3: Repeat the first two steps until its convergence



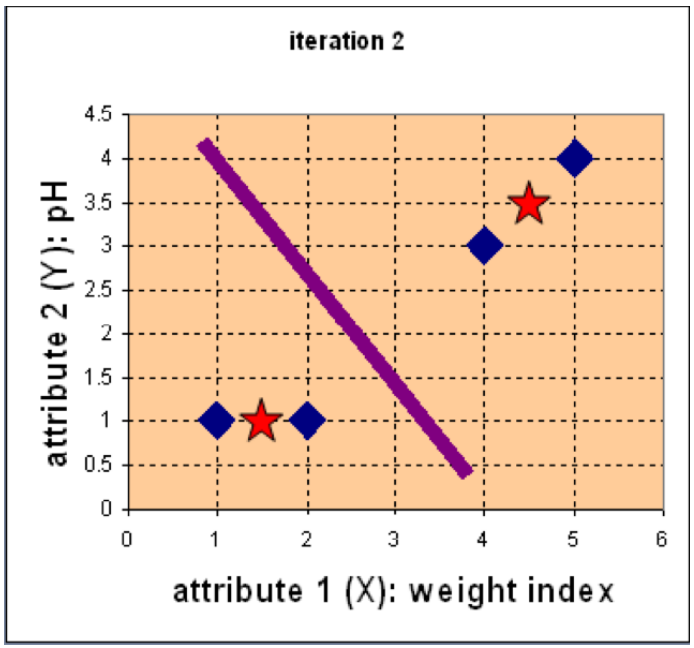
Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

$$c_1 = \left(\frac{1+2}{2},\ \frac{1+1}{2} \right) = (1\frac{1}{2},\ 1)$$

$$c_2 = \left(\frac{4+5}{2},\ \frac{3+4}{2} \right) = (4\frac{1}{2},\ 3\frac{1}{2})$$

Example

- Step 3: Repeat the first two steps until its convergence



Compute the distance of all objects to the new centroids

$$\mathbf{D}^2 = \begin{bmatrix} 0.5 & 0.5 & 3.20 & 4.61 \\ 4.30 & 3.54 & 0.71 & 0.71 \end{bmatrix} \quad \begin{array}{ll} \mathbf{c}_1 = (1\frac{1}{2}, 1) & group-1 \\ \mathbf{c}_2 = (4\frac{1}{2}, 3\frac{1}{2}) & group-2 \end{array}$$

$$\begin{array}{cc} \begin{array}{cccc} A & B & C & D \end{array} & \\ \begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} & \begin{array}{c} X \\ Y \end{array} \end{array}$$

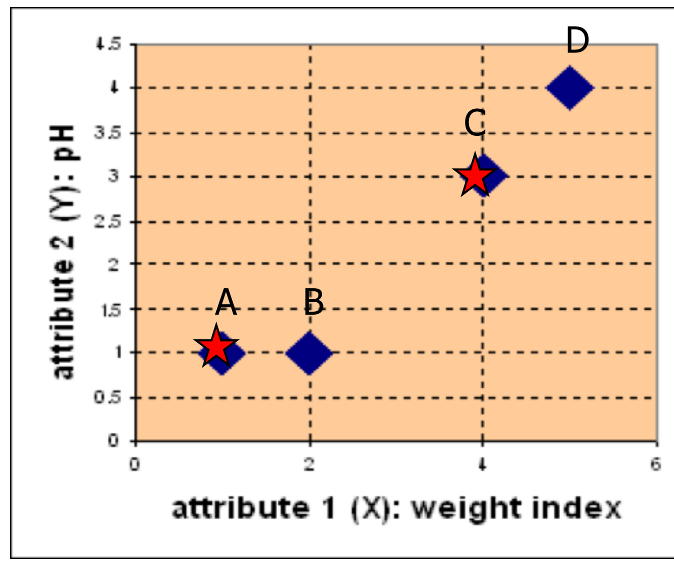
Stop due to no new assignment
Membership in each cluster no longer change

Exercise

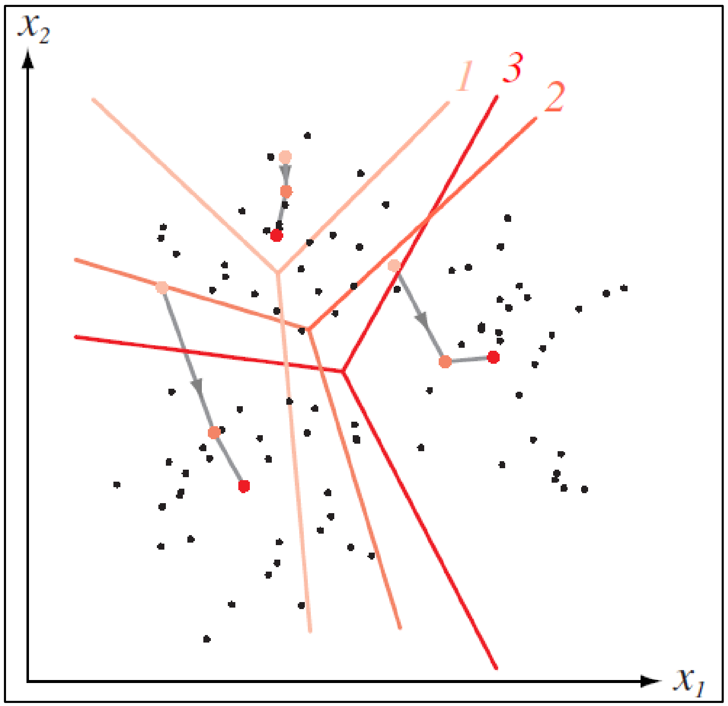
For the medicine data set, use K-means with the Manhattan distance (Taxicab/Rectilinear) metric for clustering analysis by setting $K=2$ and initialising seeds as C_1 = A and C_2 = C. Answer three questions as follows:

1. How many steps are required for convergence?
2. What are memberships of two clusters after convergence?
3. What are centroids of two clusters after convergence?

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



K-means Partitioning



When K centroids are set/fixed, they partition the whole data space into K mutually exclusive subspaces to form a partition.

Changing positions of centroids leads to a new partitioning.

Relevant Issues

- Computational complexity
  - O(tKn), where n is number of objects, K is number of clusters, and t is number of iterations. Normally, K, t << n.
- Local optimum
  - sensitive to initial seed points
  - converge to a local optimum: maybe an unwanted solution
- Other problems
  - Need to specify K, the *number* of clusters, in advance
  - Unable to handle noisy data and outliers (*K-Medoids* algorithm)
  - Not suitable for discovering clusters with non-convex shapes
  - Applicable only when mean is defined, then what about categorical data? (*K-mode* algorithm)
  - how to evaluate the K-mean performance?