COMPARE TWO K-MEANS PARALLEL ALGORITHMS

Instructor: Dr. Russ Miller
Nannan Zhai
nannanzh@buffalo.edu

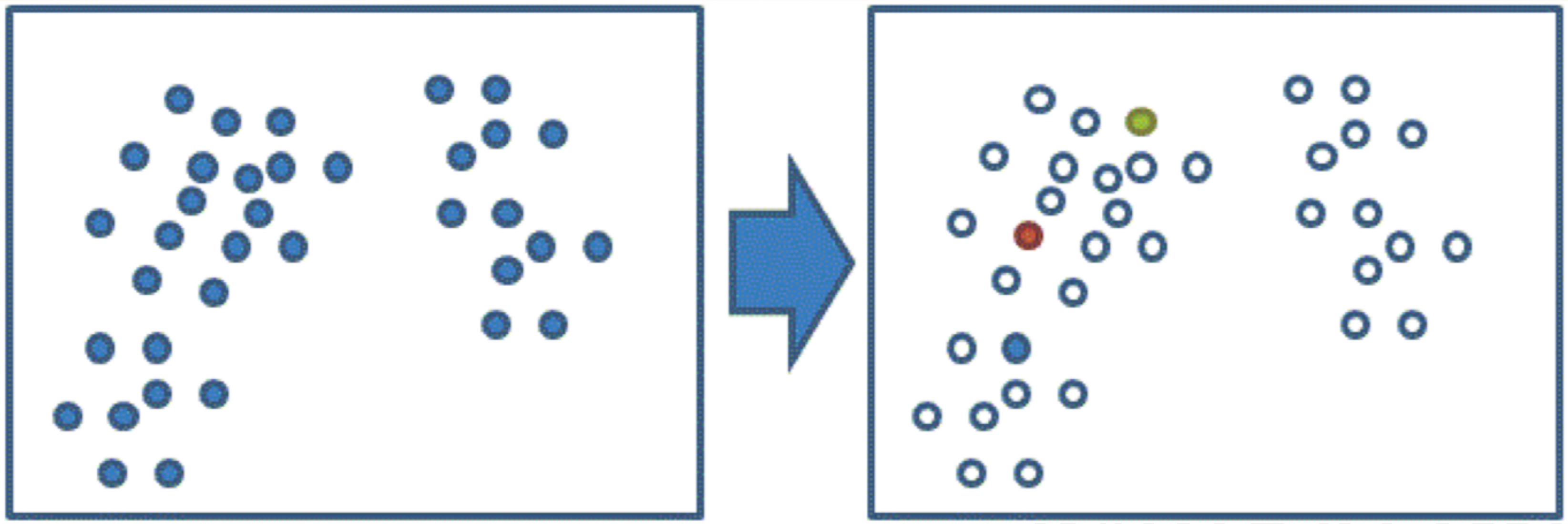
University at Buffalo The State University of New York

K-means algorithm

K-means algorithm is the most well-known and commonly used clustering method. It takes the input parameter, k and partitions a set of n objects into k clusters so that the resulting intra-cluster similarity is high whereas the intercluster similarity is low. Cluster similarity is measured according to the mean value of the objects in the cluster, which can be regarded as the cluster's "center of gravity".

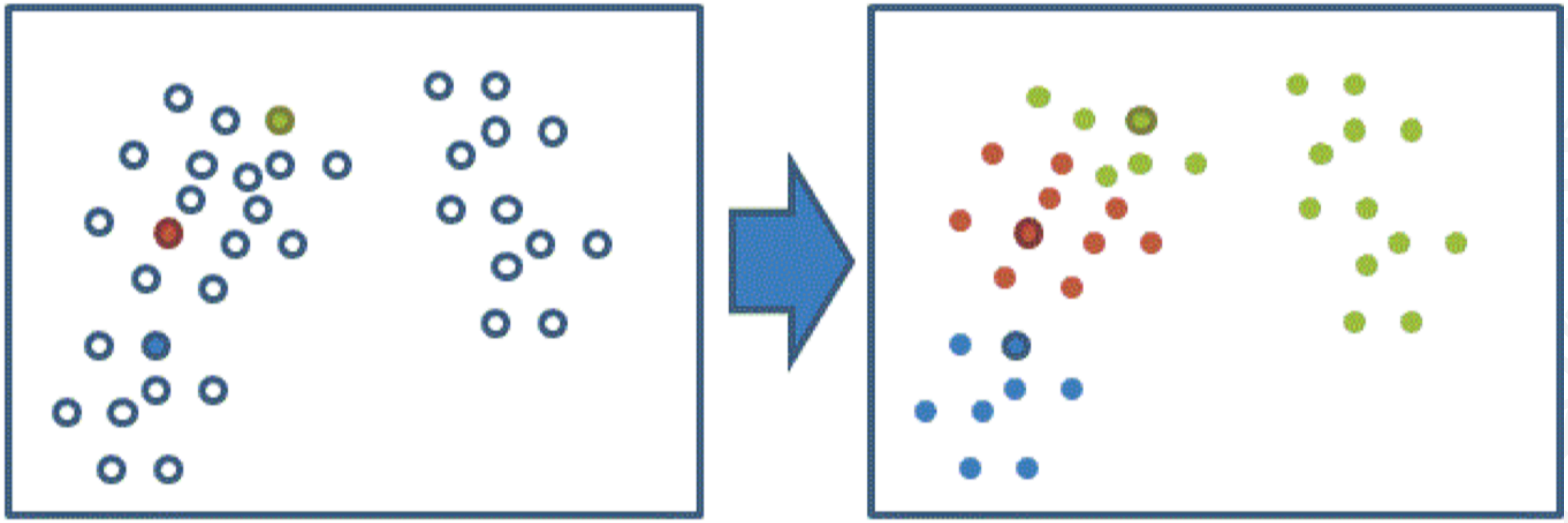
K-means algorithm

Firstly, it randomly selects k objects from the whole objects which represent initial cluster centers



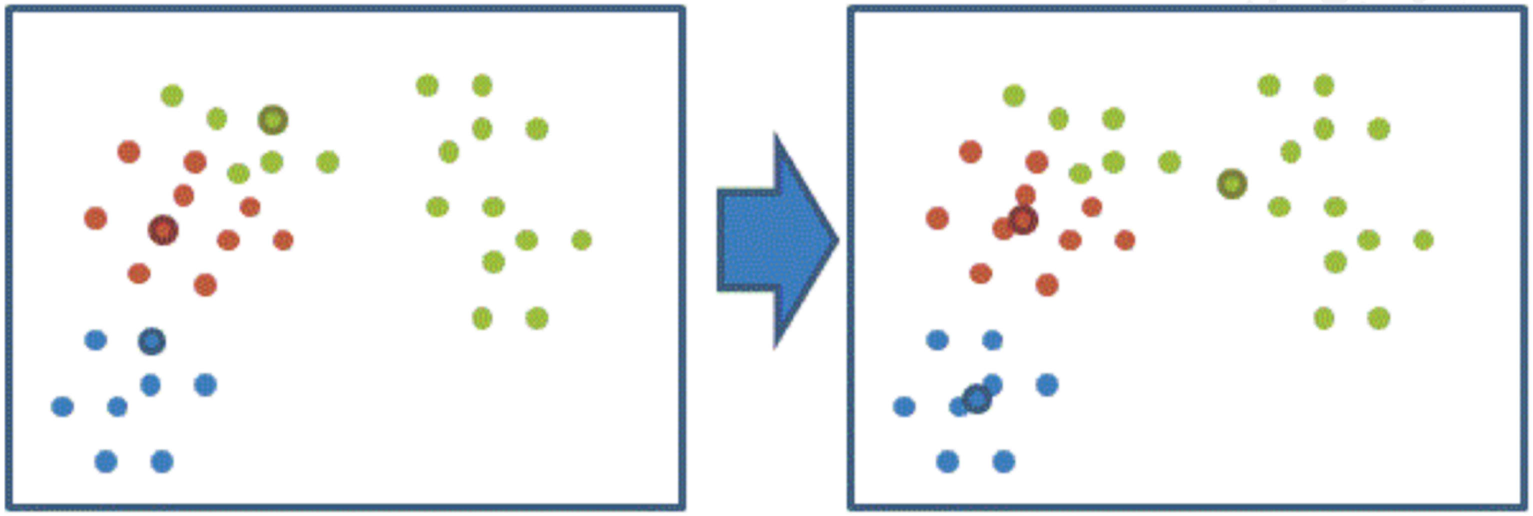
K-means algorithm

Each remaining object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster center.



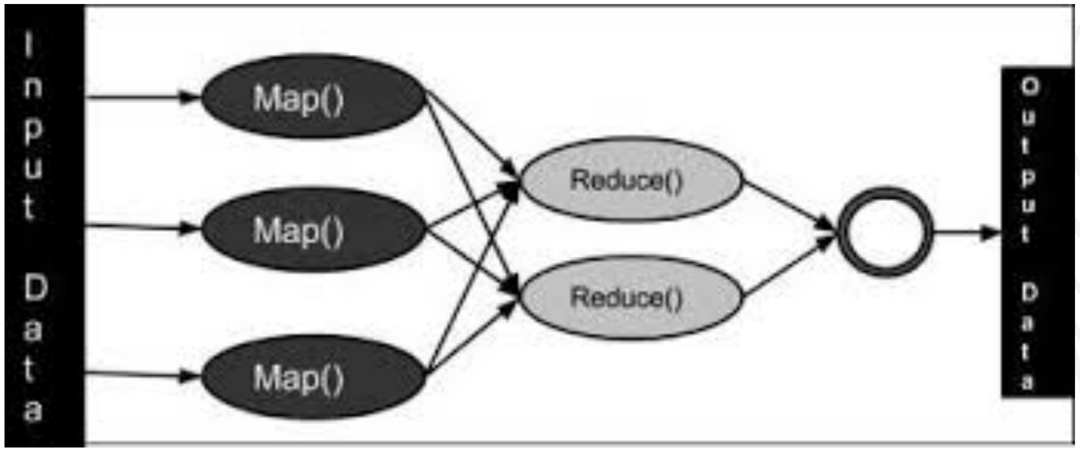
K-means algorithm

The new mean for each cluster is then calculated. This process iterates until the criterion function converges.

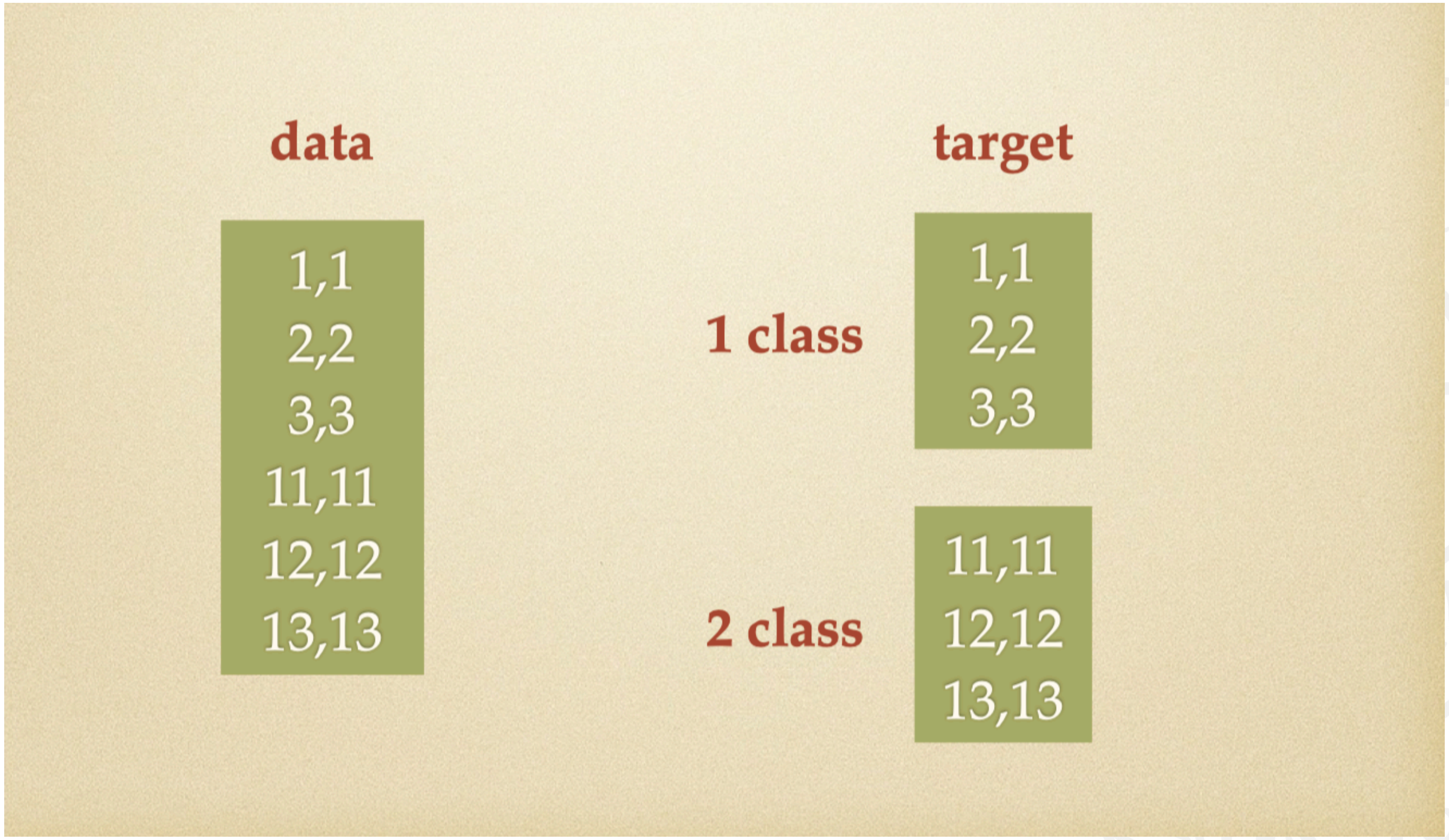


K-Means Algorithm parallel using MapReduce

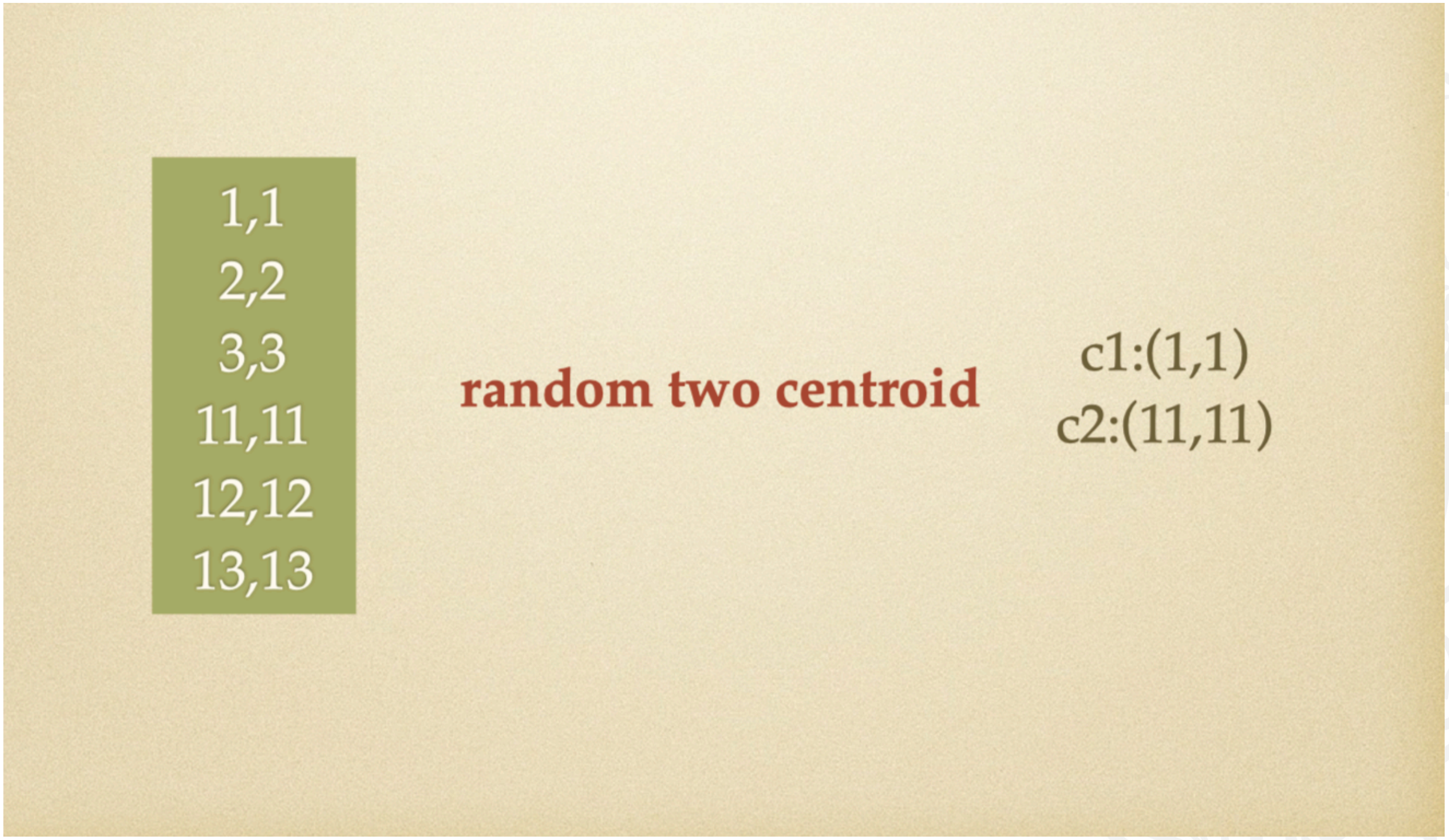
MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).



Example of MapReduce



Example of MapReduce

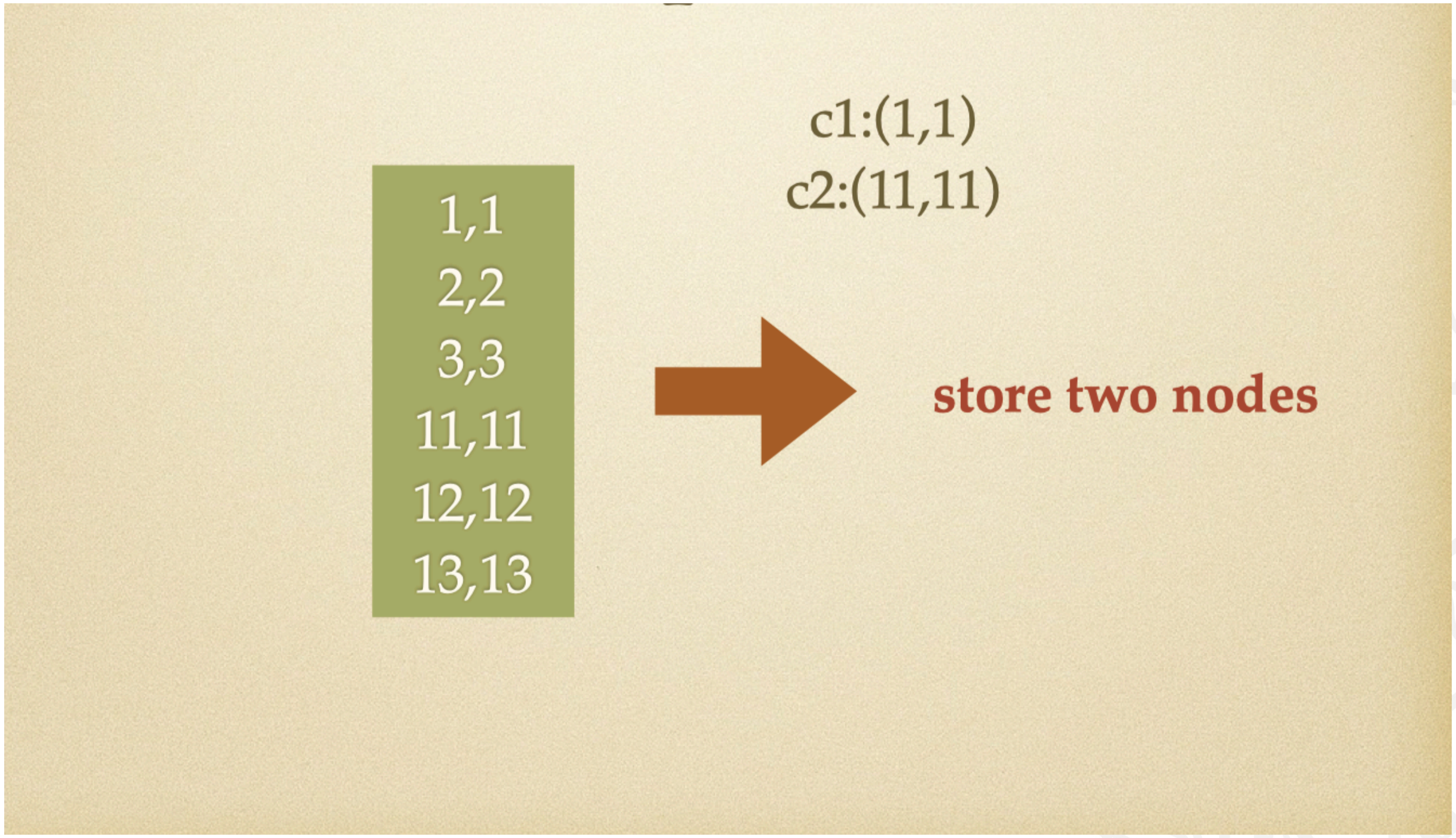


1,1
2,2
3,3
11,11
12,12
13,13

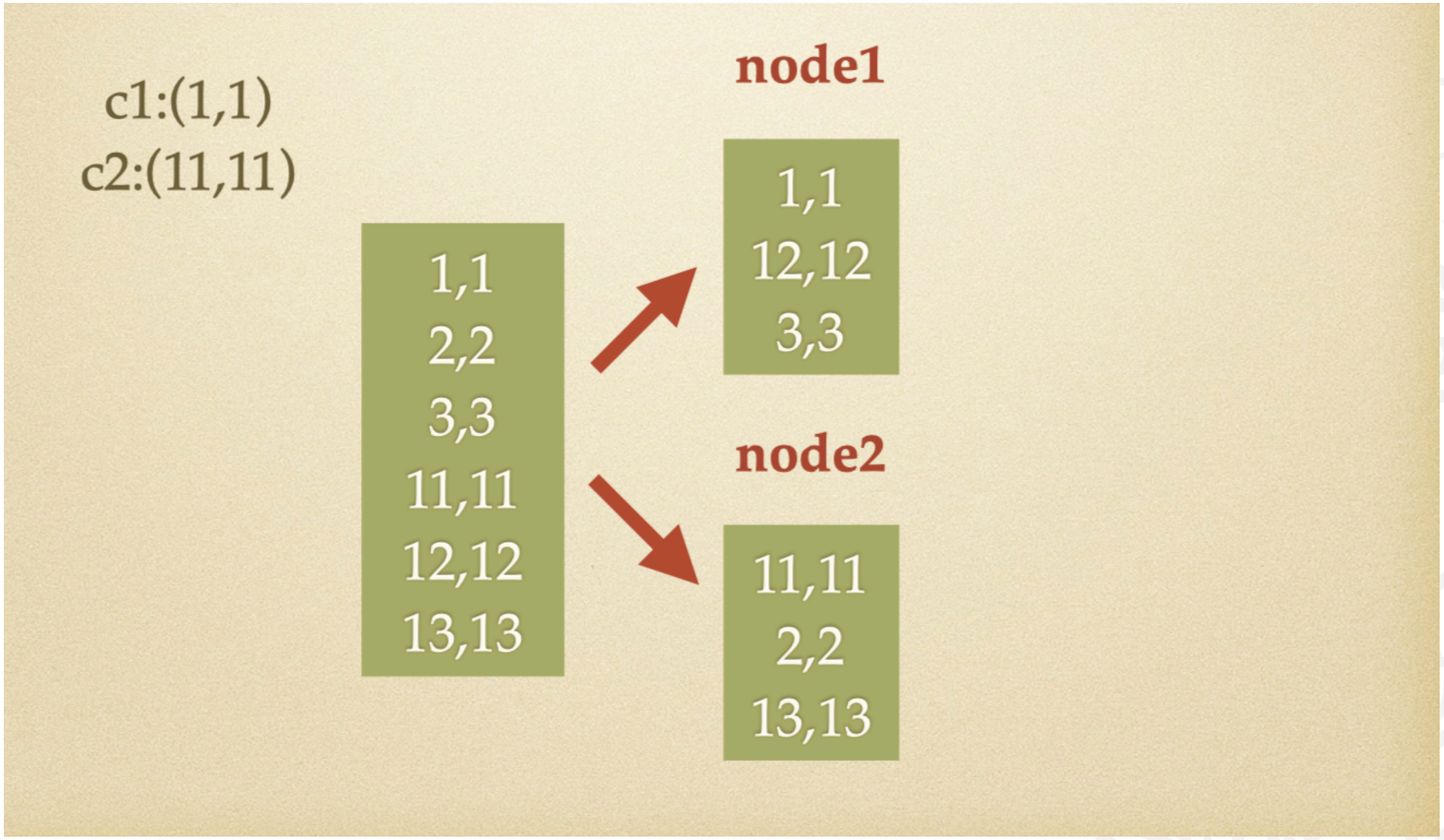
random two centroid

c1:(1,1)
c2:(11,11)

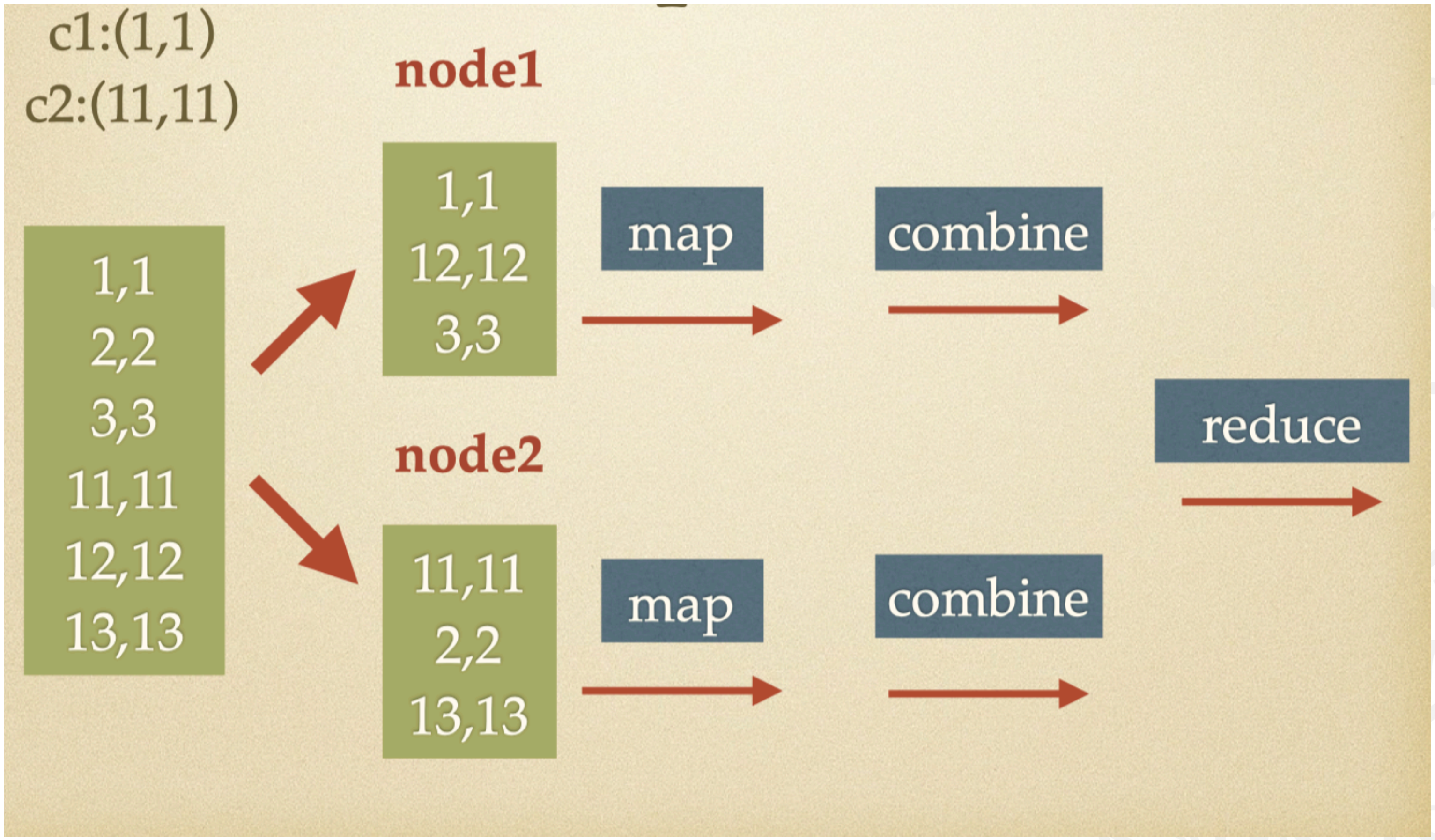
Example of MapReduce



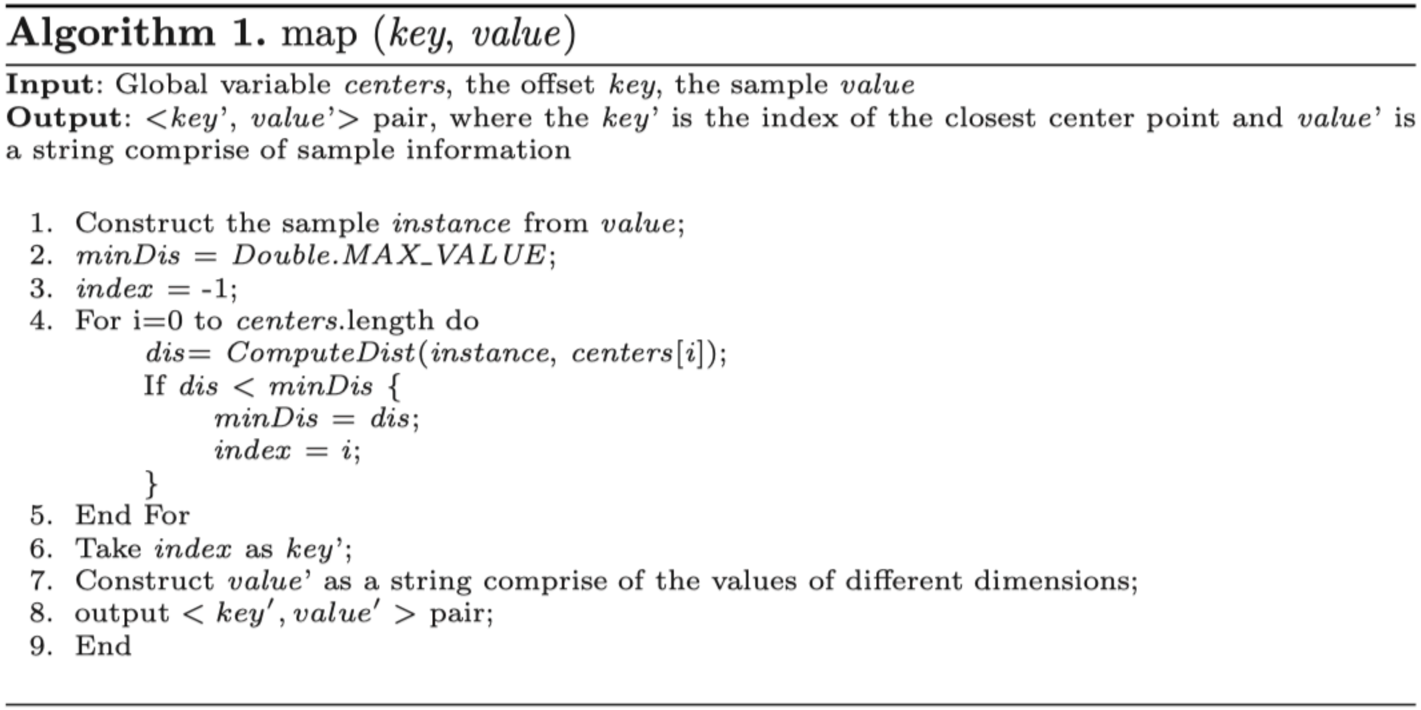
Example of MapReduce



Example of MapReduce



Step1. Map

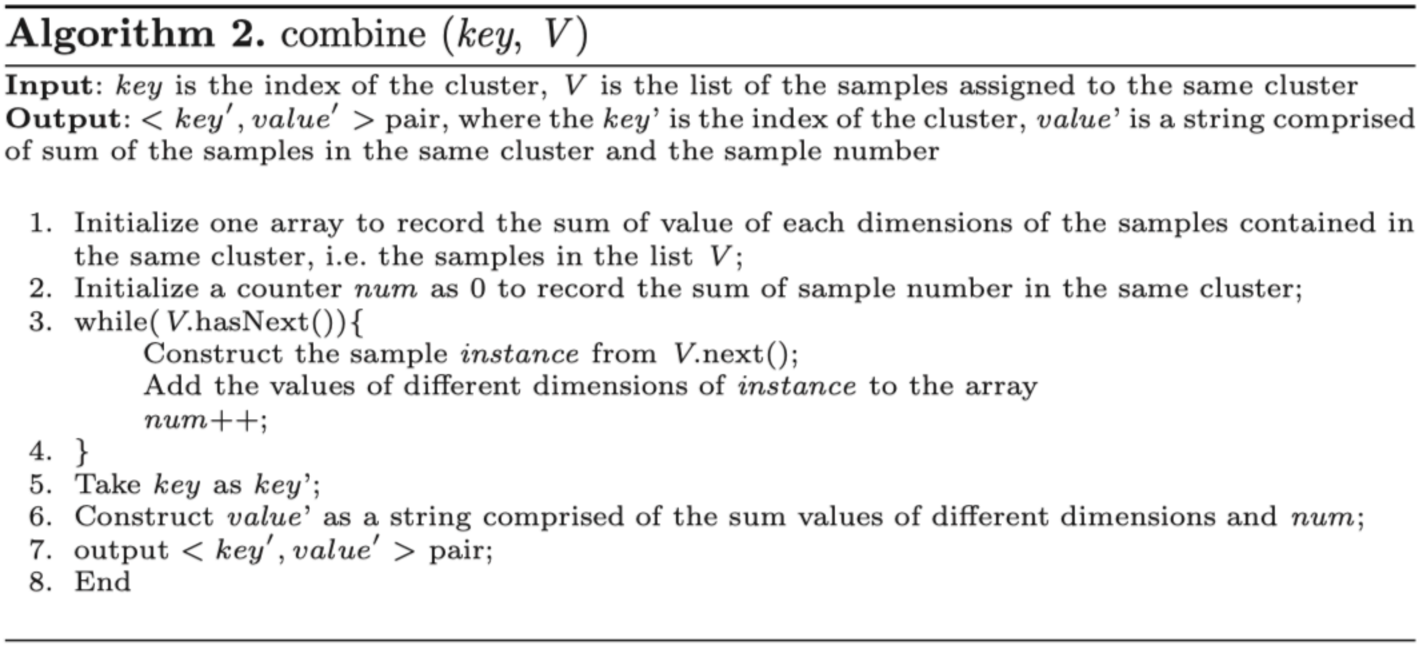


Algorithm 1. map (*key*, *value*)

Input: Global variable *centers*, the offset *key*, the sample *value*
Output: <*key*', *value*'> pair, where the *key*' is the index of the closest center point and *value*' is a string comprise of sample information

```
1. Construct the sample $instance$ from $value$;
2. $minDis = Double.MAX\_VALUE$;
3. $index$ = -1;
4. For i=0 to $centers$.length do
       $dis = ComputeDist(instance,\ centers[i])$;
       If $dis < minDis$ {
           $minDis = dis$;
           $index = i$;
       }
5. End For
6. Take $index$ as $key$';
7. Construct $value$' as a string comprise of the values of different dimensions;
8. output $< key', value' >$ pair;
9. End
```

Step2. Combine

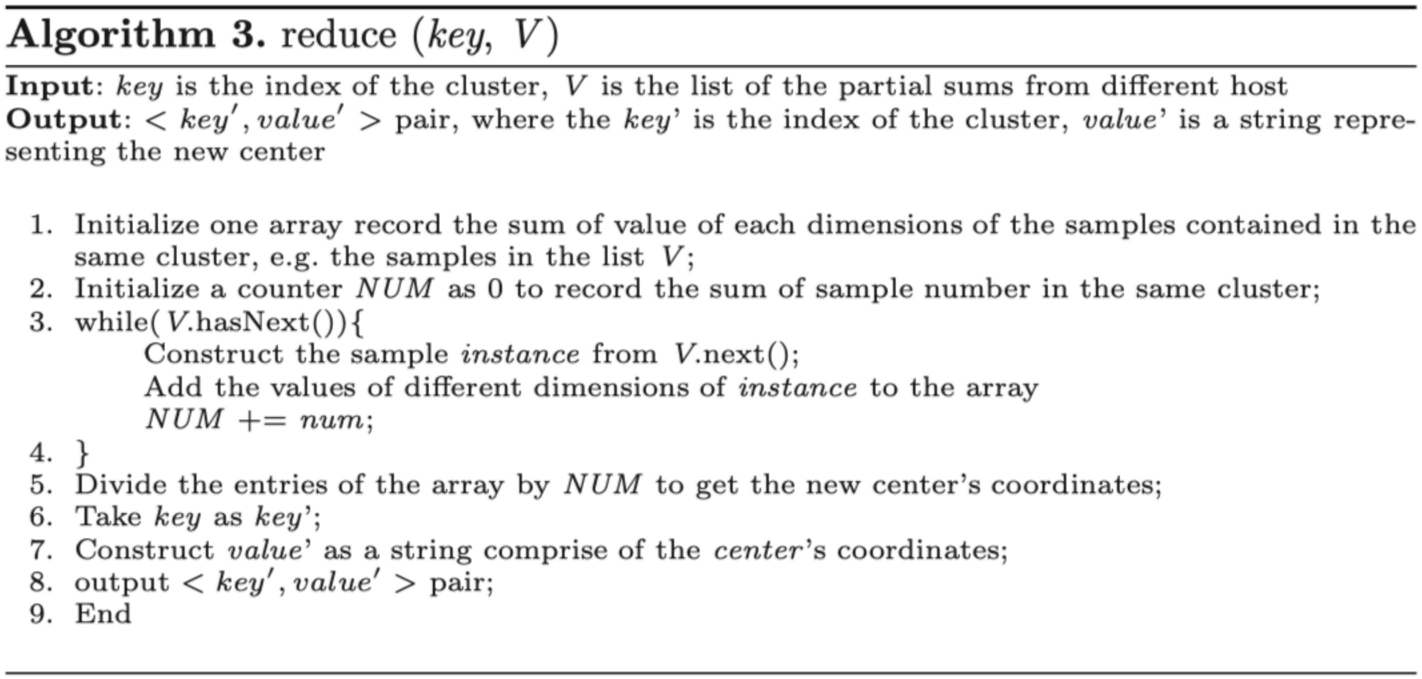


Algorithm 2. combine (*key*, *V*)

Input: *key* is the index of the cluster, *V* is the list of the samples assigned to the same cluster
Output: $< key', value' >$ pair, where the *key*' is the index of the cluster, *value*' is a string comprised of sum of the samples in the same cluster and the sample number

```
1. Initialize one array to record the sum of value of each dimensions of the samples contained in
   the same cluster, i.e. the samples in the list $V$;
2. Initialize a counter $num$ as 0 to record the sum of sample number in the same cluster;
3. while( $V$.hasNext()){
        Construct the sample $instance$ from $V$.next();
        Add the values of different dimensions of $instance$ to the array
        $num$++;
4. }
5. Take $key$ as $key$';
6. Construct $value$' as a string comprised of the sum values of different dimensions and $num$;
7. output $< key', value' >$ pair;
8. End
```

Step3. Reduce

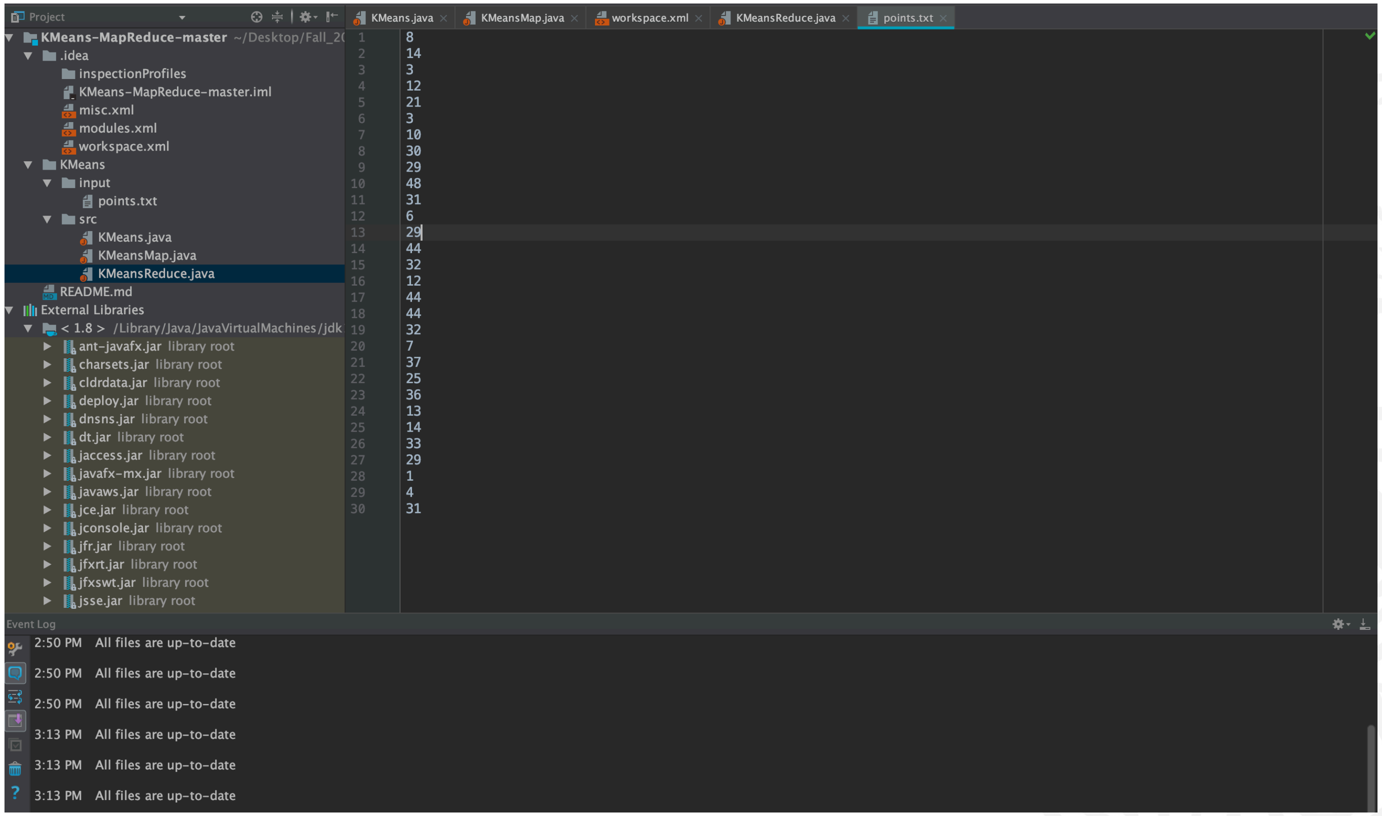


Algorithm 3. reduce $(key,\ V)$

Input: *key* is the index of the cluster, V is the list of the partial sums from different host
Output: $< key', value' >$ pair, where the *key*' is the index of the cluster, *value*' is a string representing the new center

```
1. Initialize one array record the sum of value of each dimensions of the samples contained in the
   same cluster, e.g. the samples in the list $V$;
2. Initialize a counter $NUM$ as 0 to record the sum of sample number in the same cluster;
3. while( $V$.hasNext()){
          Construct the sample $instance$ from $V$.next();
          Add the values of different dimensions of $instance$ to the array
          $NUM$ += $num$;
4. }
5. Divide the entries of the array by $NUM$ to get the new center's coordinates;
6. Take $key$ as $key$';
7. Construct $value$' as a string comprise of the $center$'s coordinates;
8. output $< key', value' >$ pair;
9. End
```

Demo



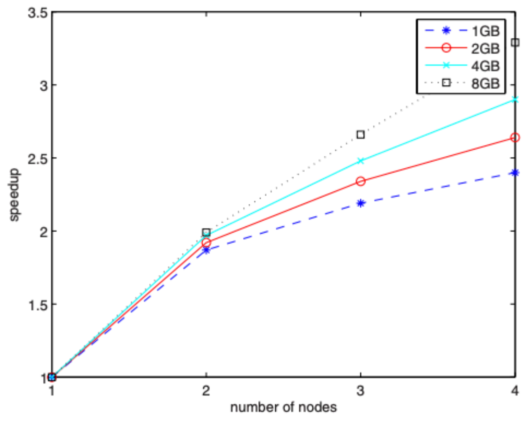
```
KMeans-MapReduce-master ~/Desktop/Fall_20
  .idea
    inspectionProfiles
    KMeans-MapReduce-master.iml
    misc.xml
    modules.xml
    workspace.xml
  KMeans
    input
      points.txt
    src
      KMeans.java
      KMeansMap.java
      KMeansReduce.java
  README.md
External Libraries
  < 1.8 > /Library/Java/JavaVirtualMachines/jdk
    ant-javafx.jar library root
    charsets.jar library root
    cldrdata.jar library root
    deploy.jar library root
    dnsns.jar library root
    dt.jar library root
    jaccess.jar library root
    javafx-mx.jar library root
    javaws.jar library root
    jce.jar library root
    jconsole.jar library root
    jfr.jar library root
    jfxrt.jar library root
    jfxswt.jar library root
    jsse.jar library root
```

```
8
14
3
12
21
3
10
30
29
48
31
6
29
44
32
12
44
44
32
7
37
25
36
13
14
33
29
1
4
31
```

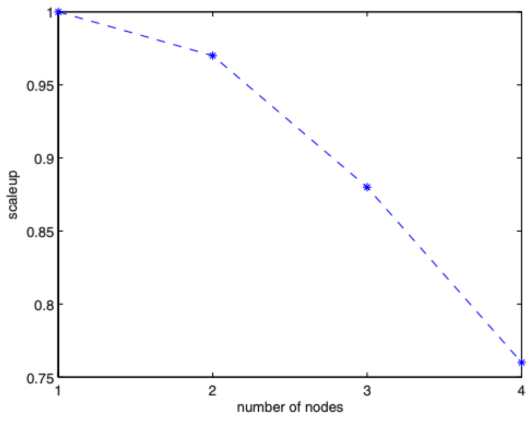
```
2:50 PM  All files are up-to-date
2:50 PM  All files are up-to-date
2:50 PM  All files are up-to-date
3:13 PM  All files are up-to-date
3:13 PM  All files are up-to-date
3:13 PM  All files are up-to-date
```

K-Means Algorithm parallel using MapReduce

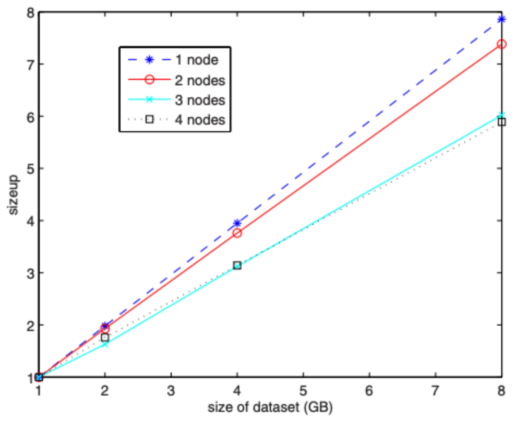
We have performed the speedup evaluation on datasets with different sizes and systems. The number of computers varied from 1 to 4. The size of the dataset increases from 1GB to 8GB.



(a) Speedup



(b) Scaleup

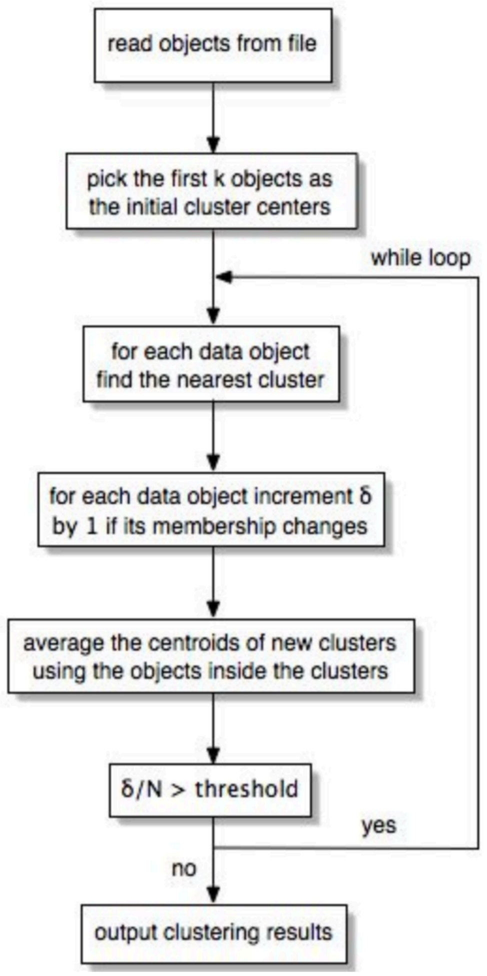


(c) Sizeup

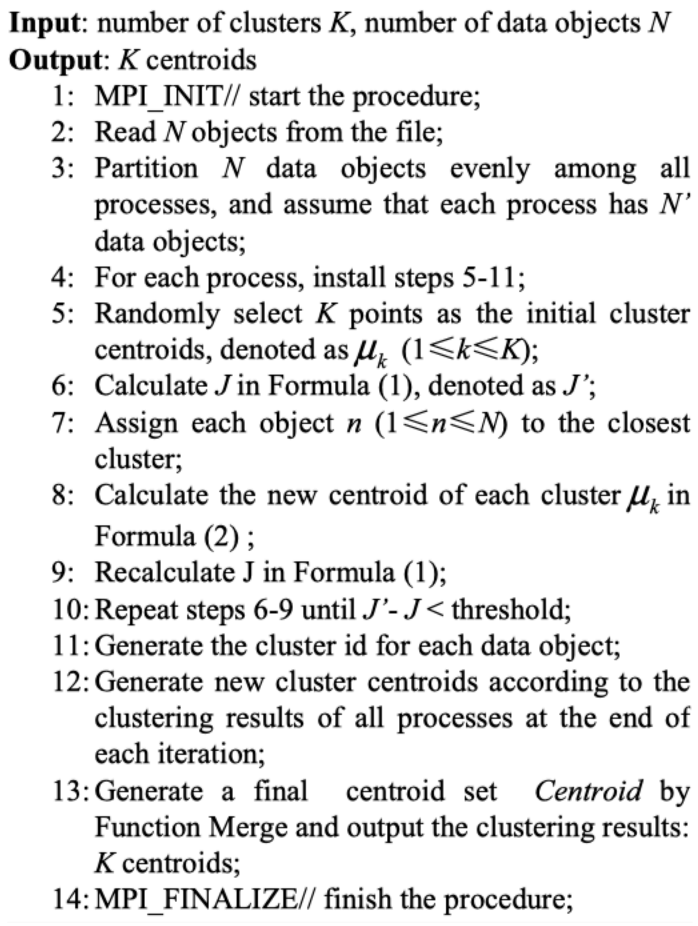
K-Means Algorithm parallel using MPI

Message Passing Interface (MPI) is a standard developed by the Message Passing Interface Forum (MPIF). MPI is a standard library specification designed to support parallel computing in a distributed memory environment.

K-Means Algorithm parallel using MPI


read objects from file
pick the first k objects as the initial cluster centers
while loop
for each data object find the nearest cluster
for each data object increment δ by 1 if its membership changes
average the centroids of new clusters using the objects inside the clusters
δ/N > threshold
yes
no
output clustering results

K-Means Algorithm parallel using MPI



Input: number of clusters K, number of data objects N
Output: K centroids

```
1:  MPI_INIT// start the procedure;
2:  Read $N$ objects from the file;
3:  Partition $N$ data objects evenly among all
    processes, and assume that each process has $N'$
    data objects;
4:  For each process, install steps 5-11;
5:  Randomly select $K$ points as the initial cluster
    centroids, denoted as $\mu_k$ $(1 \leqslant k \leqslant K)$;
6:  Calculate $J$ in Formula (1), denoted as $J'$;
7:  Assign each object $n$ $(1 \leqslant n \leqslant N)$ to the closest
    cluster;
8:  Calculate the new centroid of each cluster $\mu_k$ in
    Formula (2) ;
9:  Recalculate J in Formula (1);
10: Repeat steps 6-9 until $J' - J <$ threshold;
11: Generate the cluster id for each data object;
12: Generate new cluster centroids according to the
    clustering results of all processes at the end of
    each iteration;
13: Generate a final centroid set *Centroid* by
    Function Merge and output the clustering results:
    $K$ centroids;
14: MPI_FINALIZE// finish the procedure;
```

Two different K means parallel algorithms

1. Same Dataset
2. Same size
3. Same number of instance

Compare the performance (time cost is the key performance)

K-Means Algorithm parallel using MPI should be relatively stable and portable, and efficient in the clustering on large data sets.

K-Means Algorithm parallel using MPI （Performance）

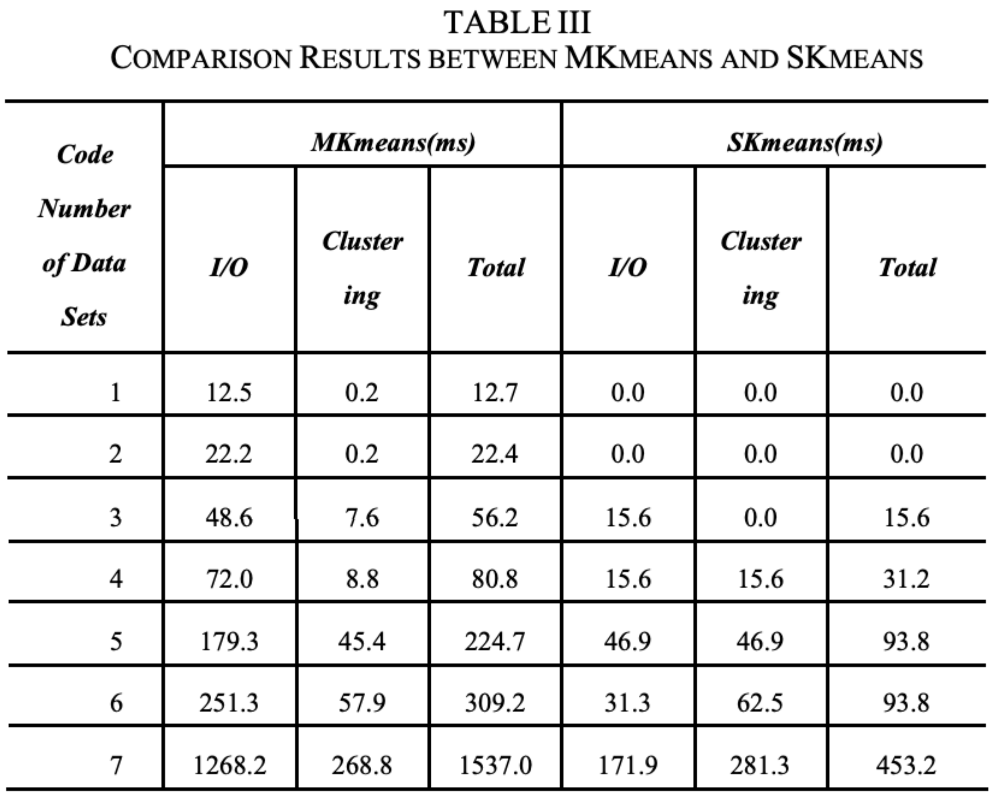


TABLE III
COMPARISON RESULTS BETWEEN MKMEANS AND SKMEANS

Code Number of Data Sets	*MKmeans(ms)*			*SKmeans(ms)*		
	I/O	*Clustering*	*Total*	*I/O*	*Clustering*	*Total*
1	12.5	0.2	12.7	0.0	0.0	0.0
2	22.2	0.2	22.4	0.0	0.0	0.0
3	48.6	7.6	56.2	15.6	0.0	15.6
4	72.0	8.8	80.8	15.6	15.6	31.2
5	179.3	45.4	224.7	46.9	46.9	93.8
6	251.3	57.9	309.2	31.3	62.5	93.8
7	1268.2	268.8	1537.0	171.9	281.3	453.2

Reference

[1] Parallel K-Means Clustering Based on MapReduce (Weizhong Zhao1,2, Huifang Ma1,2, and Qing He1 2009)

[2] A Parallel K-Means Clustering Algorithm with MPI (Jing Zhang, Gongqing Wu, Xuegang Hu, Shiying Li, Shuilong Hao)

[3] Y. Aoyama, J. Nakano, "RS/6000 SP: Practical MPI Programming," International Technical Support Organization, August 1999.