Synchronous and asynchronous clusterings

Matthieu Durut

September 20, 2012

Clustering aim

- Let $x = (x_i)_{i=1..n}$ be n points of $\mathbb{R}^d$ (data points)
- Let $c = (c_k)_{k=1..K}$ be k points of $\mathbb{R}^d$ (centroids)
- We define the empirical loss by :

$$\Phi(x, c) = \sum_{i=1}^{n} \min_{k=1..K}(||x_i - c_k||_2^2) \tag{1}$$

- and the optimal centroids by :

$$(c_k)_{k=1..K}^* = Argmin_{c \in \mathbb{R}^{d*K}} \Phi(x, c) \tag{2}$$

Some approximating algorithms

- Empirical minimizer too long to compute.
- Algorithms for approximating best clustering :

Some approximating algorithms

- Empirical minimizer too long to compute.
- Algorithms for approximating best clustering :

- K-Means
- Self-Organising Map
- Hierarchical Clustering...

Batch K-Means

- Batch K-Means steps :
  - i Initialisation of centroids
  - ii Distance Calculation
    for each x_i, get the distance $||x_i - c_k||_2$ and find the nearest centroid
  - iii Centroid Recalculation
    for each cluster, recompute centroid as the average of points assigned to this cluster
  - iv Repeat steps ii and iii till convergence
- Immediate evidence of the convergence of the algorithm

Online K-Means

- Online K-Means steps :
  i. Initialisation of centroids
  ii. Get a dataset point. Select the nearest centroid. Update this centroid.
  iii. Repeat steps ii till convergence
- Probabilist result of convergence of the Online K-Means

Algorithm 1 Sequential Batch K-Means

Select K initial centroids $(c_k)_{j=1..K}$
repeat
  for $i = 1$ to n **do**
    for $k = 1$ to K **do**
      Compute $||x_i - c_k||_2^2$
    end for
    Find the closest centroid $c_{k^*(i)}$ to x_i;
  end for
  for $k = 1$ to K **do**
    $c_k = \frac{1}{\#\{i,k^*(i)=k\}} \sum_{\{i,k^*(i)=k\}} x_i$
  end for
until no c_k has changed since last iteration or empirical loss stabilizes

K-Means Sequential cost

The cost of a sequential Batch K-Means algorithm has been studied by Dhillon. More precisely :

$$
\begin{aligned}
KMeans\ Sequential\ Cost\ &= I(n+K)d + IKd\ readings \\
&+ InKd\ soustractions \\
&+ InKd\ square\ operations \\
&+ InK(d-1) + I(n-K)d\ additions \\
&+ IKd\ divisions \\
&+ 2In + I * Kd\ writings \\
&+ IKd\ double\ comparisons \\
&+ I\ counts\ of\ K\ sets_{k=1..K}\ of\ size\ n(k) \\
&where \sum_{k=1}^{K} n(k) = n
\end{aligned}
$$

K-Means Sequential cost (2)

$$\begin{aligned} KMeans\ Sequential\ Time &= (3Knd + Kn + Kd + nd) * I * T^{flop} \\ &\simeq 3Knd * I * T^{flop} \end{aligned}$$

Distributing K-Means

1. Different ways to split computation load
2. Splitting load without affinity (worker/cluster) : each worker responsible of n/P points
3. Splitting load with affinity : each worker responsible of K/P clusters

- clustering without affinity seems more adequate.

Algorithm 2 Synchronous Distributed Batch K-Means without affinity

p = GetThisNodeId() (from 0 to P-1)
Get same initial centroids $(c_k)_{k=1..K}$ in every node
Load into local memory $S_p = \{x_i, i = p * (n/P)..(p+1) * (n/P)\}$
repeat
  for $x_i \in S_p$ **do**
    for $k = 1$ to K **do**
      Compute $||x_i - c_k||_2^2$
    end for
    Find the closest centroid $c_{k^*(i)}$ to x_i
  end for
  for $k = 1$ to K **do**
    $c_{k,p} = \frac{1}{\#\{i,\ x_i \in S_p\ \&\ k^*(i)=k\}} \sum_{\{i,\ x_i \in S_p\ \&\ k^*(i)=k\}} x_i$
  end for
  Wait for other processors to finish the for loops.
  for $k = 1$ to K **do**
    Reduce through MPI the $(c_{k,p})_{p=0..P-1}$ with the corresponding weight : $\#\{i,\ x_i \in S_p\ \&\ k^*(i) = k\}$
    Register the value in c_k
  end for
until no c_k has changed since last iteration or empirical loss stabilizes

SMP Distributed K-Means costs

- Distributed K-Means cost is dependant of hardware and how well workers can communicate.
- SMP : Symmetric MultiProcessor (shared memory)

$$
\begin{aligned}
&KMeans\ SMP\ Distributed\ Cost \\
&= T_P^{comp} \\
&= \frac{(3Knd + Kn + Kd + nd) * I * T^{flop}}{P} \\
&\simeq \frac{3Knd * I * T^{flop}}{P}
\end{aligned}
$$

DMM Distributed K-Means costs

- KMeans DMM Distributed Cost

$$
\begin{aligned}
&= T_P^{comp} + T_P^{comm} \\
&= \frac{(3Knd + Kn + Kd + nd) * I * T^{flop}}{P} + T_P^{comm} \\
&\simeq \frac{3Knd * I * T^{flop}}{P} + O(log(P))
\end{aligned}
$$

- $T_P^{comm} = O(log(P))$ comes from MPI according to Dhillon.
- Issue : the constant is far greater than log(P) for reasonable P.

Case Study : EDF load curves.

- n = 20 000 000 series
- d = 87600 (10 years of hourly series)
- K = $\sqrt{n}$ = 4472 clusters
- P = 10000 processors
- I = 100 iterations
- $T^{flop} = \frac{1}{1000000000}$ seconds

Case study on SMP

On SMP architecture (RAM limitations are not respected), we would get :

$$T^{comp}_{P,SMP} = 235066 seconds$$

$$T^{comm}_{P,SMP} \simeq 0 seconds$$

Case study on DMM using MPI

On DMM architecture, we get :

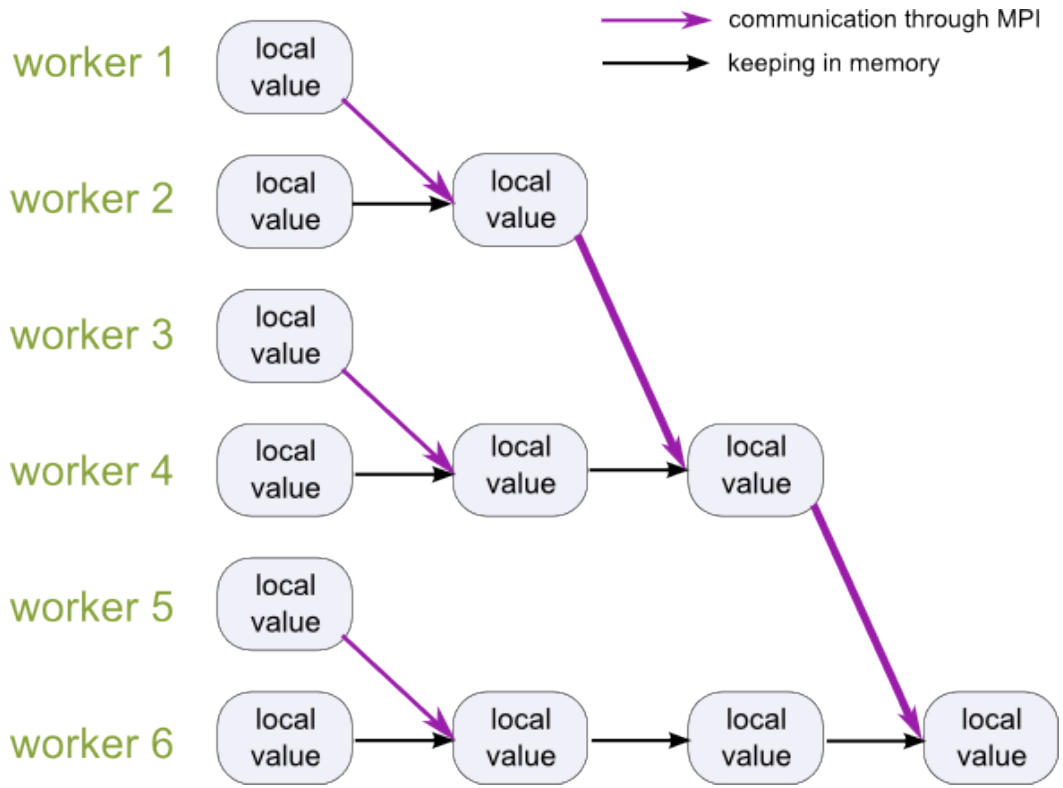
$$T^{comp}_{P,DMM} = 235066 seconds$$

For communication between 2 nodes, we can suppose :

$$\begin{aligned} &Centroids\ broadcast\ between\ 2\ processors\ time \\ &= \frac{I * Kd * sizeof 1 value}{bandwith} \\ &= I * \frac{5977 Mbytes}{20 Mbytes/second} = 29800 seconds \end{aligned}$$

$$\begin{aligned} &Centroids\ merging\ time \\ &= I * kd * T^{flop} * 5 operations : (2 multiplications, 2 additions, 1 division) \\ &= 195.87 seconds \end{aligned}$$

Communicating through Binary Tree



Estimation of $T^{comm}_{P,DMM}$

if MPI binary tree topology, $T^{comm}_{P,DMM}$ becomes :

$$
\begin{aligned}
& T^{comm}_{P,DMM} \\
& = (Centroids\ broadcast + Centroids\ merging\ time) * \lceil log_2(P) \rceil \\
& = (\frac{I * Kd * sizeof1value}{bandwith} + 5 * I * Kd * T^{flop}) * \lceil log_2(P) \rceil \\
& \simeq 420000 seconds
\end{aligned}
$$

Estimating when communication is a bottleneck

$$T_P^{comm} <= T_P^{comp}$$

$$(\frac{I * Kd * sizeof1value}{bandwith} + 5 * I * Kd * T^{flop}) * \lceil log_2(P) \rceil <= \frac{(3nKd) * I * T^{flop}}{P}$$

$$\frac{n}{P\lceil log_2(P) \rceil} >= \frac{\frac{sizeof1value}{bandwith} + 5 * T^{flop}}{3T^{flop}}$$

$$\frac{n}{P\lceil log_2(P) \rceil} >= 255$$

Empirical speed-up already observed

- (Kantabutra, Couch) 2000, clustering with affinity : P=4 (workstations with ethernet) , D=2, K =4, N=900000, best speed-up of 2.1, concludes they have a O(K/2) speed-up.
- (Kraj, Sharma, Garge, ...) 2008, (1 master, 7 nodes dualcore 3Ghz), D=200, K = 20, N=10000 genes, best speed-up 3
- (Chu, Kim, Lin, Yu,...) 2006, (1 sun workstation, 16 nodes), N=from 30000 to 2500000, speed-up from 8 to 12.
- (Dhillon, Modha) 1998, (1 IBM PowerParallel SP2 16 nodes (160Mhz)), D=8, K = 16, N= 2000000 then speed-up of 15.62 on 16 nodes, N = 2000, speed-up of 6 on 16 nodes

Cloud Computing

- Hardware resources on-demand for storage and computation

Clustering on the cloud

1. All data must transit through storage
2. Storage bandwith is limited
3. Bandwith, CPU power, latency are guaranteed on average only
4. Workers are likely to fail

- Workers shouldn't wait for each other

Algorithm 3 Asynchronous Distributed K-Means without affinity

p = GetThisNodeId() (from 0 to P-1)
Get same initial centroids $(c_k)_{k=1..K}$ in every node. Persist them on the Storage
Load into local memory $S_p = \{x_i, i = p * (n/P)..(p+1) * (n/P)\}$
repeat
  for $x_i \in S_p$ **do**
    for $k = 1$ to K **do**
      Compute $||x_i - c_k||_2^2$
    end for
    Find the closest centroid $c_{k^*(i)}$ to x_i
  end for
  for $k = 1$ to K **do**
    $c_{k,p} = \frac{1}{\#\{i,\ x_i \in S_p\ \&\ k^*(i)=k\}} \sum_{\{i,\ x_i \in S_p\ \&\ k^*(i)=k\}} x_i$
  end for
  Don't wait for other processors to finish the for loops.
  Retrieve centroids $(c_k)_{k=1..K}$ from the storage
  for $k = 1$ to K **do**
    Update c_k using $c_{k,p}$
  end for
  Update storage version of the centroids.
until empirical loss stabilizes

Current work

1. Synchronous K-Means
2. Asynchronous K-Means
3. Getting a speed-up (hopefully)

Present technical difficulties of coding on the cloud

- Code Abstractions : Inversion of Control, SOA, Storage Garbage Collection, ...
- Debugging the cloud : Mock Providers, Reporting System, ...
- Profiling the cloud : no release date
- Monitoring the cloud : Counting workers, Measuring utilization levels, ...