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:

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

(1)

and the optimal centroids by:

$$(c_k)_{k=1..K}^* = \text{Argmin}_{c \in \mathbb{R}^d \times K} \Phi(x, c)$$

(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 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 steps:

i. Initialization 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 \(\|x_i - c_k\|^2\)

    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:

\[ \text{KMeans Sequential Cost} = I(n + K)d + IKd \text{ readings} \]
\[ + InKd \text{ soustractions} \]
\[ + InKd \text{ square operations} \]
\[ + InK(d - 1) + I(n - K)d \text{ additions} \]
\[ + IKd \text{ divisions} \]
\[ + 2In + I \ast Kd \text{ writings} \]
\[ + IKd \text{ double comparisons} \]
\[ + I \text{ counts of } K \text{ sets} n(k) = 1 \ldots K \text{ of size } n(k) \]

where \( \sum_{k=1}^{K} n(k) = n \)
KMeans Sequential Time \(= (3Knd + Kn + Kd + nd) \times I \times T^{\text{flop}} \)
\(\simeq 3Knd \times I \times T^{\text{flop}} \)
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

\[ p = \text{GetThisNodeId}() \text{ (from 0 to P-1)} \]
Get same initial centroids \((c_k)_{k=1..K}\) in every node
Load into local memory \(S_p = \{x_i, i = p \times (n/P) .. (p + 1) \times (n/P)\}\)

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

Matthieu Durut  Synchronous and asynchronous clusterings
Distributed K-Means cost is dependant of hardware and how well workers can communicate.

- SMP : Symmetric MultiProcessor (shared memory)

\[
K\text{Means SMP Distributed Cost} = T_P^{\text{comp}} = \frac{(3Knd + Kn + Kd + nd) \times I \times T_f^{\text{flop}}}{P} \leq \frac{3Knd \times I \times T_f^{\text{flop}}}{P}
\]
KMeans DMM Distributed Cost

\[ T_P^{\text{comp}} + T_P^{\text{comm}} = \frac{(3Knd + Kn + Kd + nd) \cdot I \cdot T^{\text{flop}}}{P} + T_P^{\text{comm}} \]

\[ \approx \frac{3Knd \cdot I \cdot T^{\text{flop}}}{P} + O(\log(P)) \]

\( T_P^{\text{comm}} = O(\log(P)) \) 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
- $T_{\text{flop}} = \frac{1}{1000000000}$ seconds
On SMP architecture (RAM limitations are not respected), we would get:

\[ T_{P,SMP}^{\text{comp}} = 235066 \text{seconds} \]

\[ T_{P,SMP}^{\text{comm}} \approx 0 \text{seconds} \]
Case study on DMM using MPI

On DMM architecture, we get:

\[ T_{comp}^{P,DMM} = 235066\text{seconds} \]

For communication between 2 nodes, we can suppose:

\[ \text{Centroids broadcast between 2 processors time} = I \times Kd \times \text{sizeof 1 value} \]
\[ = \frac{I \times Kd \times \text{sizeof 1 value}}{\text{bandwith}} \]
\[ = I \times \frac{5977\text{Mbytes}}{20\text{Mbytes/second}} = 29800\text{seconds} \]

\[ \text{Centroids merging time} = I \times kd \times T^{flop} \times 5\text{operations : (2 multiplications, 2 additions, 1 division)} \]
\[ = 195.87\text{seconds} \]
Communicating through Binary Tree

Matthieu Durut

Synchronous and asynchronous clusterings
if MPI binary tree topology, $T_{P,DMM}^{comm}$ becomes:

$$
T_{P,DMM}^{comm} = (\text{Centroids broadcast} + \text{Centroids merging time}) \times \lceil \log_2(P) \rceil
$$

$$
= (\frac{l \times Kd \times \text{sizeof one value}}{\text{bandwidth}} + 5 \times l \times Kd \times T_{flop}) \times \lceil \log_2(P) \rceil
$$

$\approx 420000 \text{ seconds}$
Estimating when communication is a bottleneck

\[ T_P^{\text{comm}} \leq T_P^{\text{comp}} \]

\[ \left( \frac{l \times Kd \times \text{sizeof}\,\text{1\,value}}{\text{bandwidth}} + 5 \times l \times Kd \times T^{\text{flop}} \right) \cdot \lceil \log_2(P) \rceil \leq \frac{(3nKd) \times l \times T^{\text{flop}}}{P} \]

\[ \frac{n}{P \cdot \lceil \log_2(P) \rceil} \geq \frac{\text{sizeof}\,\text{1\,value}}{\text{bandwidth}} + 5 \times T^{\text{flop}} \]

\[ \frac{n}{P \cdot \lceil \log_2(P) \rceil} \geq 255 \]
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), $N=\text{from 30000 to 2500000}$, 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
1. All data must transit through storage
2. Storage bandwidth is limited
3. Bandwidth, 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

\( p = \text{GetThisNodeId()} \) (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 \( S_p = \{x_i, i = p \cdot (n/P) \ldots (p + 1) \cdot (n/P)\} \)

repeat
  for \( x_i \in S_p \) do
    for \( k = 1 \) to \( K \) do
      Compute \( ||x_i - c_k||^2 \)
    end for
    Find the closest centroid \( c_k^*(i) \) to \( x_i \)
  end for
  for \( k = 1 \) to \( K \) do
    \( c_{k,p} = \frac{1}{\#\{i, x_i \in S_p \text{ & } k^*(i) = k\}} \sum \{i, x_i \in S_p \text{ & } k^*(i) = k\} \cdot x_i \)
  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, ...