Clustering algorithms distributed over a Cloud Computing Platform.

SEPTEMBER 28TH 2012

Ph. D. thesis supervised by Pr. Fabrice Rossi.
Outline.

1. Introduction to Cloud Computing
2. Context
3. Distributed Batch K-Means
4. Distributed Vector Quantization algorithms
Outline

1. Introduction to Cloud Computing
2. Context
3. Distributed Batch K-Means
4. Distributed Vector Quantization algorithms
Introduction to Cloud Computing

What is Cloud Computing?

Some Features

1. Abstraction of commodity hardware that can be rent on-demand on a hourly basis.
2. Quasi-infinite hardware scale-up.
3. Virtualization, that makes web-applications maintenance easier.

Grid vs Cloud

- Ownership.
- Intensive use of Virtual Machines (VM).
- Elasticity.
- Hardware administration and maintenance.
Everything a as Service

1. Software as a Service (SaaS) : Gmail, Salesforce, Lokad API, etc.
2. Platform as a Service (PaaS) : Azure, Amazon S3, etc,
3. Infrastructure as a Service (IaaS) : Amazon EC2, etc.

Stack of Azure

- Storage Level : BlobStorage, TableStorage, QueueStorage, SQLAzure.
- Execution Level : Dryad.
- Domain Specific Language Level : DryadLinq, Scope.
**Figure**: Illustration of the Google, Hadoop and Microsoft technology stacks for cloud applications building.
The Windows Azure Storage (WAS)

**BlobStorage**
- **Key-value pair** (blobname/blob) storage.
- **No more ACID.**
- **But atomicity, strong persistency and strong consistency** per blob.
- Optimistic Read-Modify-Write primitive (RMW).

**QueueStorage**
- Set of scalable queues.
- Asynchronous Message Delivery mechanism.
- Approximately FIFO.
- Messages returned at least once => **Idempotency.**
Elements of Azure applications architecture

- No communication framework such as MPI.
- WAS used as a shared memory abstraction.
- No affinity between storage and processing units.
- Task agnosticism of workers (at least in the beginning).
- Idempotence.
Outline

1. Introduction to Cloud Computing
2. Context
3. Distributed Batch K-Means
4. Distributed Vector Quantization algorithms
Why clustering?

- One of the Lokad’s abilities is to deal with large scale data.
- Need to group client data (clustering) to extract information from complex objects (e.g. time series seasonality).

Problem Set-up

- Data set is composed of \( N \) points \( \{ z_t \}^N_{t=1} \) in \( \mathbb{R}^\kappa \).
- Clustering POV: find a simplified representation with \( \kappa \) vectors of \( \mathbb{R}^d \).
- These vectors will be called prototypes/centroids and gathered in a quantization scheme \( w = (w_1, \ldots, w_{\kappa}) \in (\mathbb{R}^d)^\kappa \).
Objective

Clustering challenge can be expressed as a minimization of the empirical distortion $C_N$, where

$$C_N(w) = \sum_{t=1}^{N} \min_{\ell=1,\ldots,\kappa} \|z_t - w_\ell\|^2, \quad w \in (\mathbb{R}^d)^\kappa.$$ 

Initial challenge

Exact minimization is computationally intractable.

Some approximative algorithms

- Batch K-Means
- Vector Quantization (Online K-Means)
- Neural Gas
- Kohonen Maps
Why distributed?

- A suitable way to allow more computing resources. Faster **serial computers**: increasingly expensive + physical limits.

- Cloud computing: adopted by Lokad (**MS Azure**). Early 2012, all apps on Cloud and scale-up $\sim 300$VMs.

- Consequences: **communication delays** and the lack of **efficient shared memory** $\rightarrow$ **asynchronous** schemes.
Outline

1. Introduction to Cloud Computing
2. Context
3. Distributed Batch K-Means
4. Distributed Vector Quantization algorithms
Sequential Batch K-Means

Algorithm 1 Sequential Batch K-Means

Select $\kappa$ initial prototypes $(w_k)_{k=1}^{\kappa}$

repeat

for $t = 1$ to $N$ do

for $k = 1$ to $\kappa$ do

compute $||z_t - w_k||_2^2$

end for

find the closest centroid $w_{k^*}(t)$ from $z_t$;

end for

for $k = 1$ to $\kappa$ do

$w_k = \frac{1}{\#\{t, k^*(t) = k\}} \sum_{\{t, k^*(t) = k\}} z_t$

end for

until the stopping criterion is met
Characteristics

- Relatively fast: $Batch_{seq}^{Walltime} = (3N\kappa d + N\kappa + Nd + \kappa d) IT^{flop}$, where $I$ refers to the number of iterations and $T^{flop}$ refers to the time for a floating point operation to be evaluated.
- Deterministic.
- Easy to set-up.
- Results stationary from a certain iteration.

Suitable for parallelization?

- Obvious data-level Parallelism.
- Same result than sequential.
- Excellent speed-up efficiency already achieved.
Distribution Scheme

- Data-level parallelism suggests iterated Map-Reduce distribution.
- Data set \( \{z_t\}_{t=1}^N \) is homogeneously split into \( M \) chunks (one per processing unit): \( S^i, i \in \{1..M\} \).
- The processing unit \( i \) computes the distance \( ||z_t^i - w_k||_2^2 \) for \( z_t^i \in S^i \) and \( k \in \{1..\kappa\} \) (Map phase).
- Then the new prototypes version is recomputed by one or several machines (Reduce phase).
Batch K-Means distributed over a DMM architecture
Wall Time

$$Batch_{DMM}^{Wall Time} = T_{M}^{comp} + T_{M}^{comm},$$

where $T_{M}^{comp}$ refers to the wall time of the assignment phase and $T_{M}^{comm}$ refers to the wall time of the recalculation phase (mostly spent in communications).

Assignment phase

$$T_{M}^{comp} = \frac{3IN_{κ}dT^{flop}}{M}.$$
Recalculation phase - DMM architecture with MPI

\[ T_{M}^{comm} = \left\lceil \log_2(M) \right\rceil \frac{l_{K}dS}{B}, \]

where \( S \) refers to the size of a double in memory (8 bytes in the following) and \( B \) refers to the communication bandwidth per machine.

Wall time - DMM architecture with MPI

\[ \text{Batch}_{DMM}^{Wall Time} = \frac{3IN_{K}dT^{flop}}{M} + \left\lceil \log_2(M) \right\rceil \frac{l_{K}dS}{B}. \]
Speed-up - DMM architecture with MPI

\[
\text{SpeedUp}_{DMM}(M, N) = \frac{3NT^{\text{flop}}}{3NT^{\text{flop}}_M + \frac{S}{B} \lceil \log_2(M) \rceil}.\]

Optimal number of processing units

\[
M^*_DMM = \frac{3NT^{\text{flop}}B}{S}.\]
Batch K-Means distributed over Azure
Figure: Distribution scheme of our cloud Batch K-Means.
Communication Modeling

\[ T_M^{\text{comm}} = I \sqrt{M \kappa dS(2T_{\text{read}}^{\text{Blob}} + T_{\text{write}}^{\text{Blob}})}, \]

where \( T_{\text{read}}^{\text{Blob}} \) (resp. \( T_{\text{write}}^{\text{Blob}} \)) refers to the time needed by a given processing unit to download (resp. upload) a blob from (resp. to) the storage per memory unit.

Speed-up - Cloud architecture

\[ \text{SpeedUp}(M, N) = \frac{3NT^{\text{flop}}}{\frac{3NT^{\text{flop}}}{M} + \sqrt{MS(2T_{\text{read}}^{\text{Blob}} + T_{\text{write}}^{\text{Blob}})}}. \]

Optimal number of workers

\[ M^*(N) = \sqrt[2/3]{\frac{6NT^{\text{flop}}}{S(2T_{\text{read}}^{\text{Blob}} + T_{\text{write}}^{\text{Blob}})}}. \]
Figure: Time to execute the Reduce phase per unit of memory $(2T_{read}^{Blob} + T_{write}^{Blob})$ in $10^{-7}$ sec/Byte in function of the number of communicating units.
Figure: Charts of speedup performance curves with different data set size.
### Table:
Comparison between the effective optimal number of processing units $M_{\text{eff}}^*$ and the theoretical optimal number of processing units $M^*$ for different data set size.

<table>
<thead>
<tr>
<th>Exp.</th>
<th>$N$</th>
<th>$M_{\text{eff}}^*$</th>
<th>$M^*$</th>
<th>Wall Time</th>
<th>Sequential theoretic time</th>
<th>Effective Speedup</th>
<th>Theoretical Speedup ($= \frac{M^*}{3}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62500</td>
<td>27</td>
<td>28</td>
<td>264</td>
<td>2798</td>
<td>10.6</td>
<td>9.34</td>
</tr>
<tr>
<td>2</td>
<td>125000</td>
<td>45</td>
<td>45</td>
<td>306</td>
<td>5597</td>
<td>18.29</td>
<td>14.84</td>
</tr>
<tr>
<td>3</td>
<td>250000</td>
<td>78</td>
<td>71</td>
<td>384</td>
<td>11194</td>
<td>29.15</td>
<td>23.55</td>
</tr>
<tr>
<td>4</td>
<td>500000</td>
<td>95</td>
<td>112</td>
<td>521</td>
<td>22388</td>
<td>43.0</td>
<td>37.40</td>
</tr>
</tbody>
</table>
Figure: Charts of speedup performance curves with different number of processing units. For each value of $M$, the value of $N$ is set accordingly so that the processing units are heavy loaded with data and computations.
Figure: Distribution of the processing time (in second) for multiple runs of the same computation task for multiple VM.
Outline

1. Introduction to Cloud Computing
2. Context
3. Distributed Batch K-Means
4. Distributed Vector Quantization algorithms
Asynchronous clustering: motivation
Joint work with Benoit Patra

Every actions should be accounted once
- No calculation should be discarded.
- No calculation should be used more than once.
- All the writes should result into prototypes update everywhere.
- All the reads should be used locally.

*On War* from Clausewitz
- Saturate bandwidth, memory, CPU, etc.
- \( \Rightarrow \) Asynchronism
- \( \Rightarrow \) Online or at least mini-batch (no more batch)
Sequential VQ algorithm

- Consists in **incremental updates** of the \((\mathbb{R}^d)^\kappa\)-valued prototypes \(\{w(t)\}_{t=0}^{\infty}\).
- Initiated from a random initial \(w(0) \in (\mathbb{R}^d)^\kappa\).
- Given a series of positive steps \((\varepsilon_t)_{t>0}\), it produces a series of \(w(t)\) by updating \(w\) at each step with a “descent term”.

\[
H(z, w) = \left((w_\ell - z) \mathbb{1}_{\{l = \text{argmin}_{i=1,\ldots,\kappa} \|z - w_i\|_2\}}\right)_{1 \leq \ell \leq \kappa}.
\]

\[
w(t + 1) = w(t) - \varepsilon_{t+1} H(z_{\{t+1 \mod n\}}, w(t)), \quad t \geq 0.
\]
**Algorithm 2** Sequential VQ algorithm

Select $\kappa$ initial prototypes $(w_k)_{k=1}^\kappa$

Set $t=0$

repeat
  for $k = 1$ to $\kappa$ do
    compute $||z_{(t+1 \mod n)} - w_k||_2^2$
  end for
  Deduce $H(z_{(t+1 \mod n)}, w)$
  Set $w(t+1) = w(t) - \varepsilon_{t+1} H(z_{(t+1 \mod n)}, w(t))$
increment $t$
until the stopping criterion is met
Our context

- We assume that a satisfactory VQ implementation has been found but too slow.
- We will not be concerned with optimization of the several parameters (initialization, sequence of steps etc.)
- We have access to a finite dataset: \( \{ z^i_t \}_{t=0}^n, i \in \{ 1, \ldots, M \} \) distributed over \( M \) processing units.
- When does a distributed VQ implementation perform better than the corresponding sequential one?
Definition of Speed-up for VQ algorithms

- A “reference” prototypes version is made available in the shared-memory (BlobStorage), referred to as the prototypes shared version: $w^{srd}$.
- Performance is measured with the corresponding empirical distortion: for all $w \in (\mathbb{R}^d)^\kappa$,

$$L_N(w) = \frac{1}{nM} \sum_{i=1}^{M} \sum_{t=1}^{n} \min_{\ell=1,\ldots,\kappa} \|z^i_t - w_{\ell}\|^2$$

- After any $t$ wall time seconds, the empirical distortion of the prototypes shared version should be lower than for the prototypes version produced by sequential algorithm.
Previous work

- VQ as stochastic gradient descent method
- Shared-Memory: interleaving the prototypes version updates
- No Shared-Memory but Loss Convexity: averaging the prototypes versions

In our case

- No efficient shared-memory
- No convexity of the loss function

Organization of our work

- Simulated distributed architecture on a single machine.
- Then cloud implementation
First distributed scheme

All the versions are set equal at time $t = 0$, $w^1(0) = \ldots = w^M(0)$. For all $i \in \{1, \ldots, M\}$ and all $t \geq 0$, we have the following iterations:

$$\begin{cases} 
    w^i_{\text{temp}} = w^i(t) - \varepsilon_{t+1} H \left( z^i_{\{t+1 \mod n\}}, w^i(t) \right) \\
    w^i(t + 1) = w^i_{\text{temp}} & \text{if } t \mod \tau \neq 0 \text{ or } t = 0, \\
    w^{\text{srd}} = \frac{1}{M} \sum_{j=1}^{M} w^j_{\text{temp}} & \text{if } t \mod \tau = 0 \text{ and } t \geq \tau, \\
    w^i(t + 1) = w^{\text{srd}} 
\end{cases}$$
A first basic parallelization scheme

Figure: A simple (and synchronous) scheme: whenever $\tau$ points are processed an averaging phase occurs.
A first basic parallelization scheme

Figure: Charts of performance with different number of computing entities: $M = 1, 2, 10$ and $\tau = 10$. 
A comparison between the previous parallel scheme and the sequential VQ

For \( t \mod \tau = 0 \) and \( t > 0 \). Then, for all \( i \in \{1, \ldots, M\} \),

\[
\begin{align*}
\dot{w}^i(t+1) &= \dot{w}^i(t-\tau+1) - \sum_{t'=\text{mod}(t+1, \tau)} \epsilon_{t'+1} \left( \frac{1}{M} \sum_{j=1}^M H \left( z_{t'+1}^j, w^j(t') \right) \right) \quad \text{(parallel)} \\
\dot{w}(t+1) &= \dot{w}(t-\tau+1) - \sum_{t'=\text{mod}(t+1, \tau)} \epsilon_{t'+1} H \left( \mathbf{z}_{\text{mod}(t'+1, n)}, \mathbf{w}(t') \right) \quad \text{(sequential)}
\end{align*}
\]

Terms in blue are estimators of the gradient.
Two SGD algorithms with the same sequence of steps then, they have similar convergence speed.

Sequence of steps → learning rate → trade-off exploration/convergence.

Introducing displacement/descent terms

For all $j \in \{1, \ldots, M\}$ and $t_2 \geq t_1 \geq 0$ set

$$\Delta^j_{t_1 \rightarrow t_2} = \sum_{t' = t_1 + 1}^{t_2} \varepsilon_{t' + 1} H \left( z^j_{\{t' + 1 \mod n\}}, w^j(t') \right).$$

corresponds to the displacement of the prototypes computed by $j$ during $(t_1, t_2)$,
Second distributed scheme

\[
\begin{align*}
  \left\{\begin{array}{l}
  w^i_{\text{temp}} = w^i(t) - \varepsilon_{t+1} H \left( z^i_{\{t+1 \mod n\}}, w^i(t) \right) \\
  w^i(t + 1) = w^i_{\text{temp}} \\
  w^{\text{srd}} = w^{\text{srd}} - \sum_{j=1}^{M} \Delta^j_{t-\tau \rightarrow t} \\
  w^i(t + 1) = w^{\text{srd}} 
  \end{array}\right. \\
  \text{if } t \mod \tau \neq 0 \text{ or } t = 0, \\
  \text{if } t \mod \tau = 0 \text{ and } t \geq \tau.
\end{align*}
\]
Figure: Illustration of the parallelization scheme of VQ procedures described by equations (43).
Figure: Charts of performance curves for a reviewed scheme $M = 1, 2, 10$ and $\tau = 10$. 
Delayed distributed scheme

\[
\begin{align*}
    w^i_{\text{temp}} &= w^i(t) - \varepsilon_{t+1} H\left(z^i_{\{t+1 \mod n\}}, w^i(t)\right) \\
    w^i(t + 1) &= w^i_{\text{temp}} \quad \text{if } t \mod \tau \neq 0 \text{ or } t = 0, \\
    w^{srd} &= w^{srd} - \sum_{j=1}^{M} \Delta^j_{t-2\tau \rightarrow t-\tau} \quad \text{if } t \mod \tau = 0 \text{ and } t \geq 2\tau, \\
    w^i(t + 1) &= w^{srd} - \Delta^i_{t-\tau \rightarrow t} \quad \text{if } t \mod \tau = 0 \text{ and } t \geq \tau.
\end{align*}
\]
Figure: Illustration of the parallelization scheme described by equations (46). The reducing phase is only drawn for processor 1 where $t = 2\tau$ and processor 4 where $t = 4\tau$. 

\[ w^1 = w^{strd} - \Delta^1_{\tau \to 2\tau} \]
\[ w^2 = w^{strd} - \Delta^2_{0 \to \tau} - \Delta^3_{0 \to 3\tau} - \Delta^4_{0 \to 4\tau} \]
\[ w^3 = w^{strd} - \Delta^3_{3\tau \to 4\tau} \]
\[ w^4 = w^{strd} - \Delta^1_{2\tau \to 3\tau} - \Delta^2_{2\tau \to 3\tau} - \Delta^3_{2\tau \to 3\tau} - \Delta^4_{2\tau \to 3\tau} \]
Distributed Vector Quantization algorithms

Figure: Charts of performance curves for iterations (46) with different numbers of computing entities, $M = 1, 2, 10$ and $\tau = 10$. 
**Motto:** summing displacement term rather than averaging versions.

**Experimental results**

- Satisfactory speed-ups are recovered for the later simulated parallel schemes.
- Delays (determinist + random) are also studied: reasonable [random] delays do not have sever impact on the convergence.
- Good perspectives for a true implementation on a cloud computing platform.
The **CloudDALVQ** project

- Scientific project for testing new large scale clustering/quantization algorithms distributed on a Cloud Platform (**MS Azure**).
Figure: Distribution scheme of our cloud VQ implementation.
Distributed Vector Quantization algorithms

BlobStorage

ProcessService

read buffer
local version
displacement
term
write buffer
shared version
process thread
data
process action 1
process action 2
process action 3
local version
displacement term
write buffer
process thread
displacement
term
BlobStorage

Matthieu Durut (Telecom/Lokad)
Figure: Normalized quantization curves with $M = 1, 2, 4, 8, 16$. Troubles appear with $M = 16$ because the ReduceService is overloaded.
Figure: Normalized quantization curves with $M = 8, 16, 32, 64$ with an extra layer for the so called “reducing task”.
**Figure**: This chart reports on the competition between our cloud DAVQ algorithm and the cloud Batch K-Means. The graph shows the empirical distortion of the algorithms over the time.