Clustering algorithms distributed over a Cloud Computing Platform.

SEPTEMBER 28 TH 2012

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

Outline.

Outline

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.

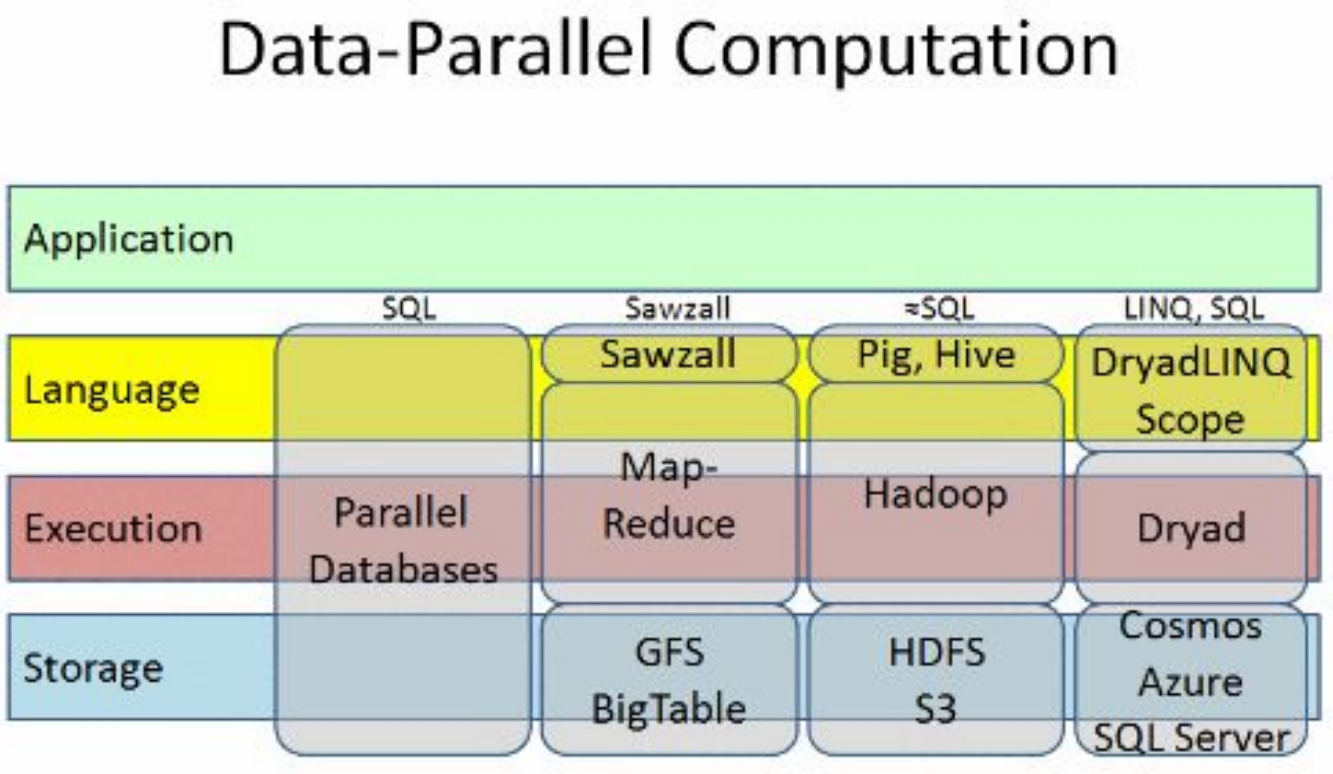


Figure : Illustration of the Google, Hadoop and Microsoft technology stacks for cloud applications building.

MapReduce

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 agnosticity of workers (at least in the beginning).
- Idempotence.

Outline

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 $\{\mathbf{z}_t\}_{t=1}^{N}$ in $\mathbb{R}^{\kappa}$.
- Clustering POV: find a simplified representation with κ vectors of $\mathbb{R}^{d}$.
- These vectors will be called prototypes/centroids and gathered in a quantization scheme $w = (w_1, \ldots, w_\kappa) \in \left(\mathbb{R}^{d}\right)^{\kappa}$.

Objective

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

$$C_N(w) = \sum_{t=1}^{N} \min_{\ell=1,\dots,\kappa} \|\mathbf{z}_t - w_\ell\|^2, \quad w \in \left(\mathbb{R}^d\right)^\kappa.$$

Initial challenge

Exact minimization is computationnaly intractable.

Some approximative algorithms

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

Architecture Context

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 $\sim$ 300VMs.
- Consequences: communication delays and the lack of efficient shared memory $\rightarrow$ asynchronous schemes.

Outline

Sequencial Batch K-Means

Algorithm 1 Sequential Batch K-Means

Select κ initial prototypes $(w_k)_{k=1}^{\kappa}$
repeat
  for $t = 1$ to N **do**
    for $k = 1$ to κ **do**
      compute $||\mathbf{z}_t - w_k||_2^2$
    end for
    find the closest centroid $w_{k^*(t)}$ from $\mathbf{z}_t$;
  end for
  for $k = 1$ to κ **do**

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

  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.
- Deterministe.
- Easy to set-up.
- Results stationary from a certain iteration.

Suited 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 $\{\mathbf{z}_t\}_{t=1}^{N}$ is homogeneously split into M chunks (one per processing unit): $S^i, i \in \{1..M\}$.
- The processing unit i computes the distance $||\mathbf{z}_t^i - w_k||_2^2$ for $\mathbf{z}_t^i \in S^i$ 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

$$Batch_{DMM}^{WallTime} = T_M^{comp} + T_M^{comm},$$

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

$$T_M^{comp} = \frac{3INKdT^{flop}}{M}.$$

Recalculation phase - DMM architecture with MPI

$$T_M^{comm} = \lceil log_2(M) \rceil \frac{I \kappa d S}{B},$$

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

$$Batch_{DMM}^{WallTime} = \frac{3IN\kappa d T^{flop}}{M} + \lceil log_2(M) \rceil \frac{I \kappa d S}{B}.$$

Speed-up - DMM architecture with MPI

$$SpeedUp_{DMM}(M,N) = \frac{3NT^{flop}}{\frac{3NT^{flop}}{M} + \frac{S}{B}\lceil log_2(M)\rceil}.$$

Optimal number of processing units

$$M^*_{DMM} = \frac{3NT^{flop}B}{S}.$$

Batch K-Means distributed over Azure

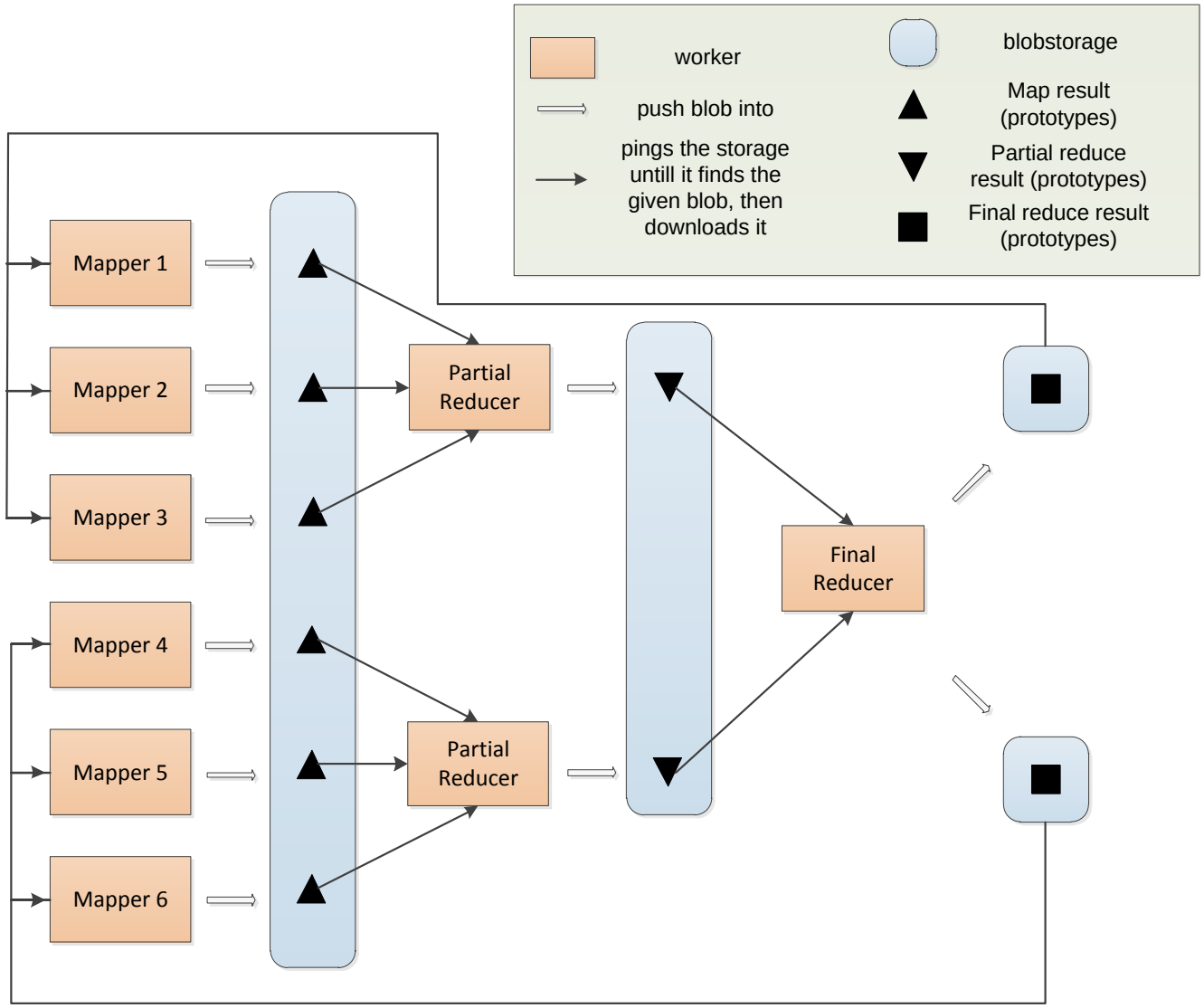


Figure : Distribution scheme of our cloud Batch K-Means.

Communication Modeling

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

where T_{Blob}^{read} (resp. T_{Blob}^{write}) 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

$$SpeedUp(M,N) = \frac{3NT^{flop}}{\frac{3NT^{flop}}{M} + \sqrt{M}S(2T_{Blob}^{read} + T_{Blob}^{write})}.$$

Optimal number of workers

$$M^*(N) = \sqrt[2/3]{\frac{6NT^{flop}}{S(2T_{Blob}^{read} + T_{Blob}^{write})}}.$$

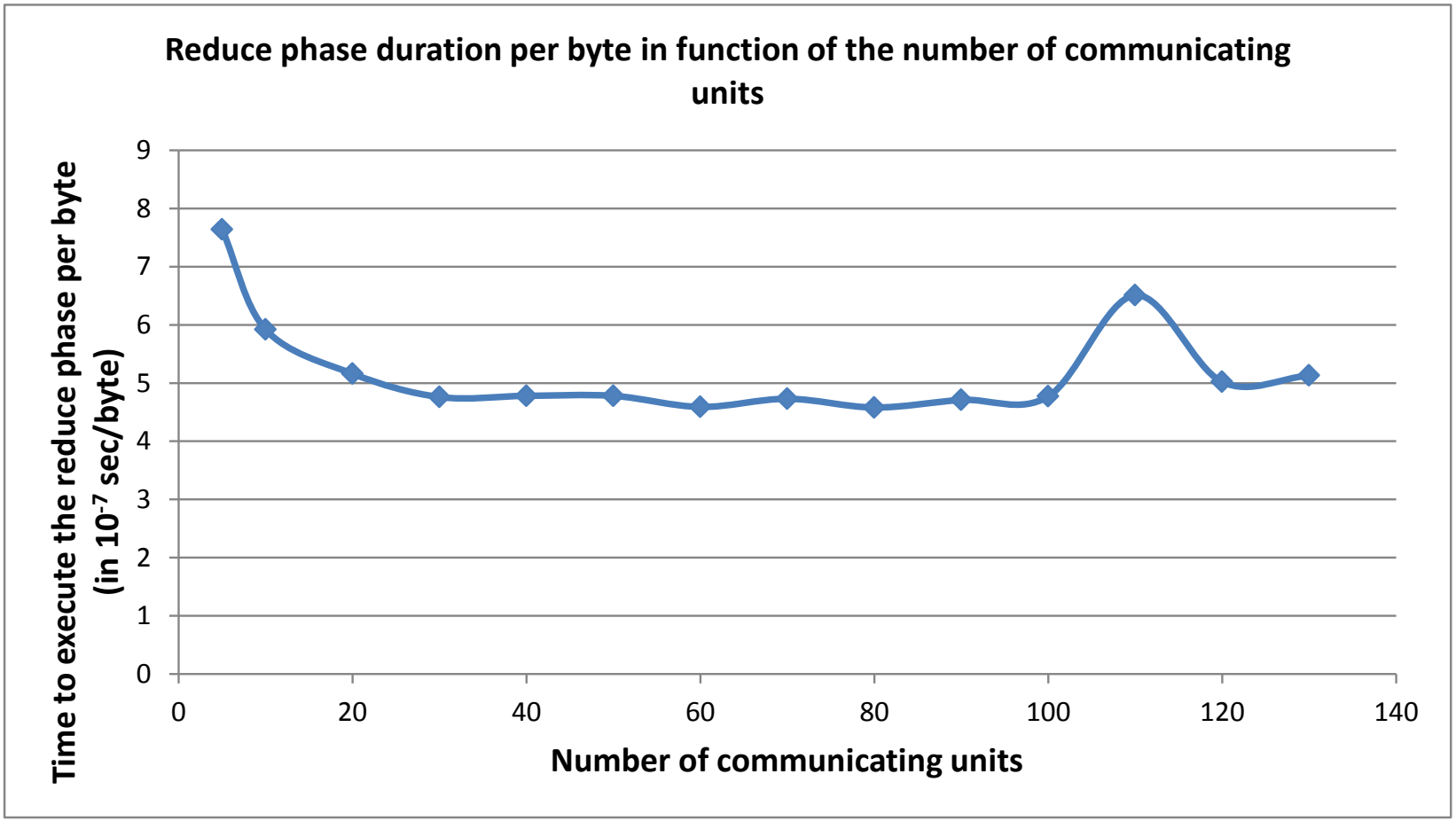


Figure : Time to execute the Reduce phase per unit of memory $(2T^{read}_{Blob} + T^{write}_{Blob})$ in 10^{-7}sec/Byte in function of the number of communicating units.

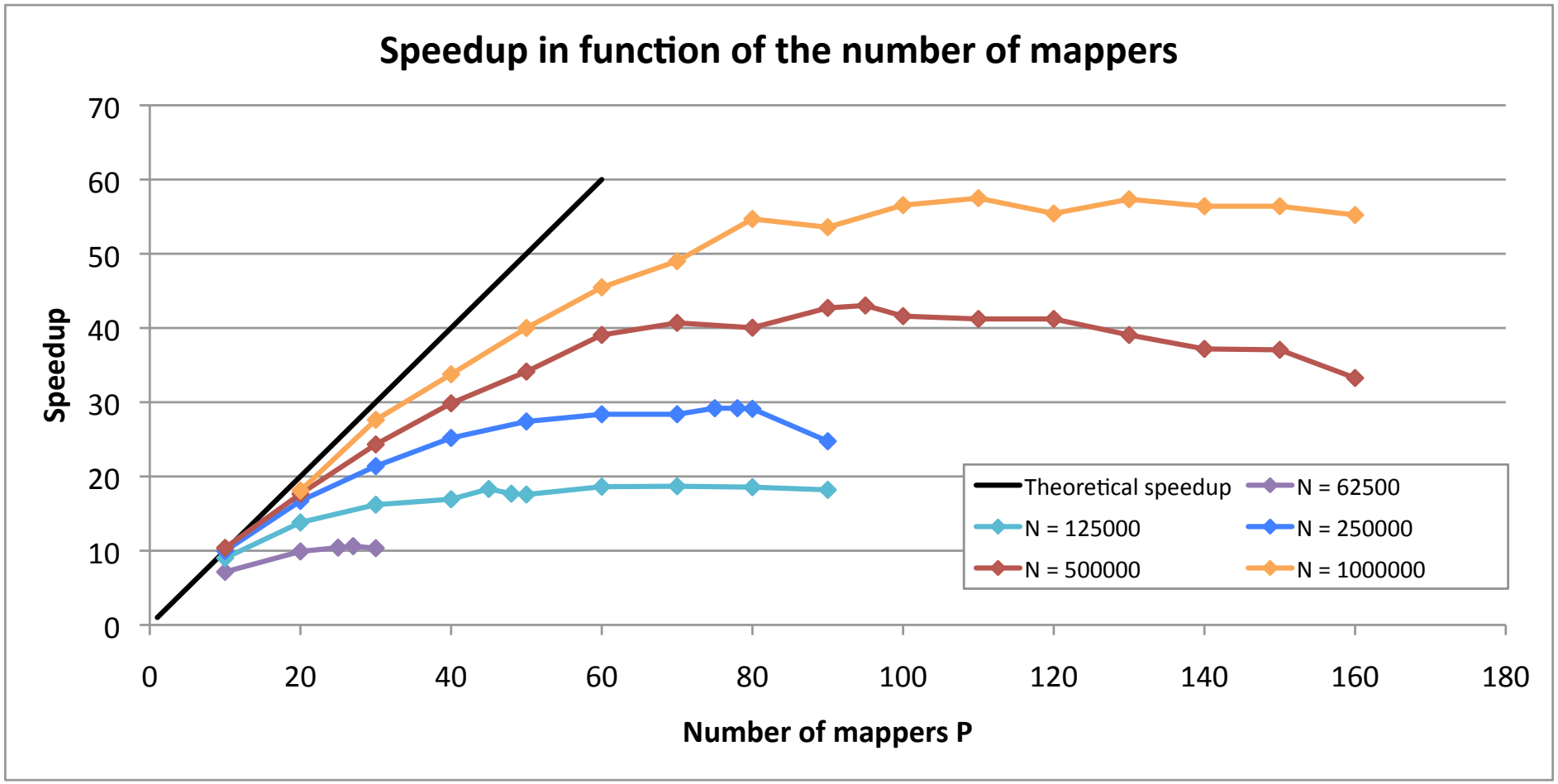


Figure : Charts of speedup performance curves with different data set size.

	N	M^*_{eff}	M^*	Wall Time	Sequential theoretic time	Effective Speedup	Theoretical Speedup ($= \frac{M^*}{3}$)
Exp. 1	62500	27	28	264	2798	10.6	9.34
Exp. 2	125000	45	45	306	5597	18.29	14.84
Exp. 3	250000	78	71	384	11194	29.15	23.55
Exp. 4	500000	95	112	521	22388	43.0	37.40

Table : Comparison between the effective optimal number of processing units M^*_{eff} and the theoretical optimal number of processing units M^* for different data set size.

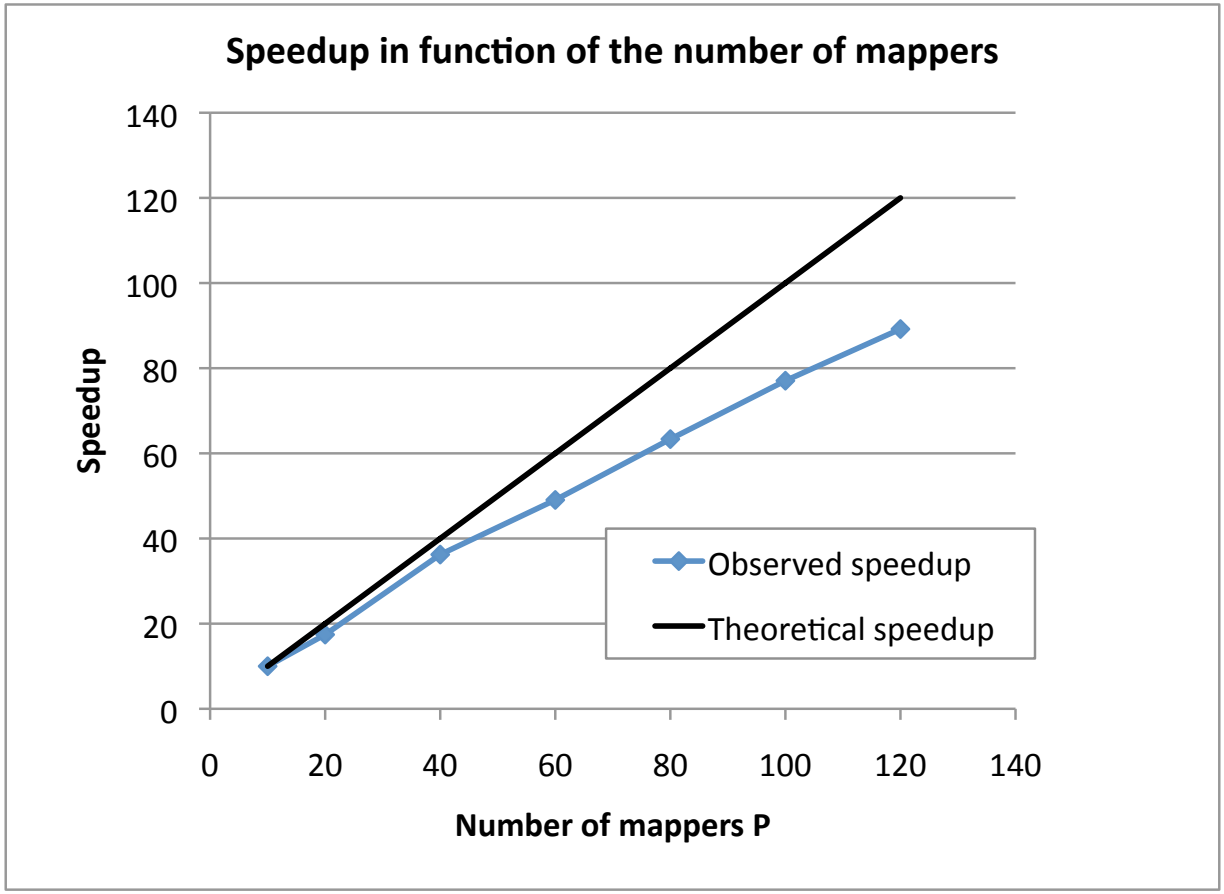


Figure : Charts of speedup performance curves with different number of processing units. For each value of M, the value of N is set accordingly so that the processing units are heavy loaded with data and computations.

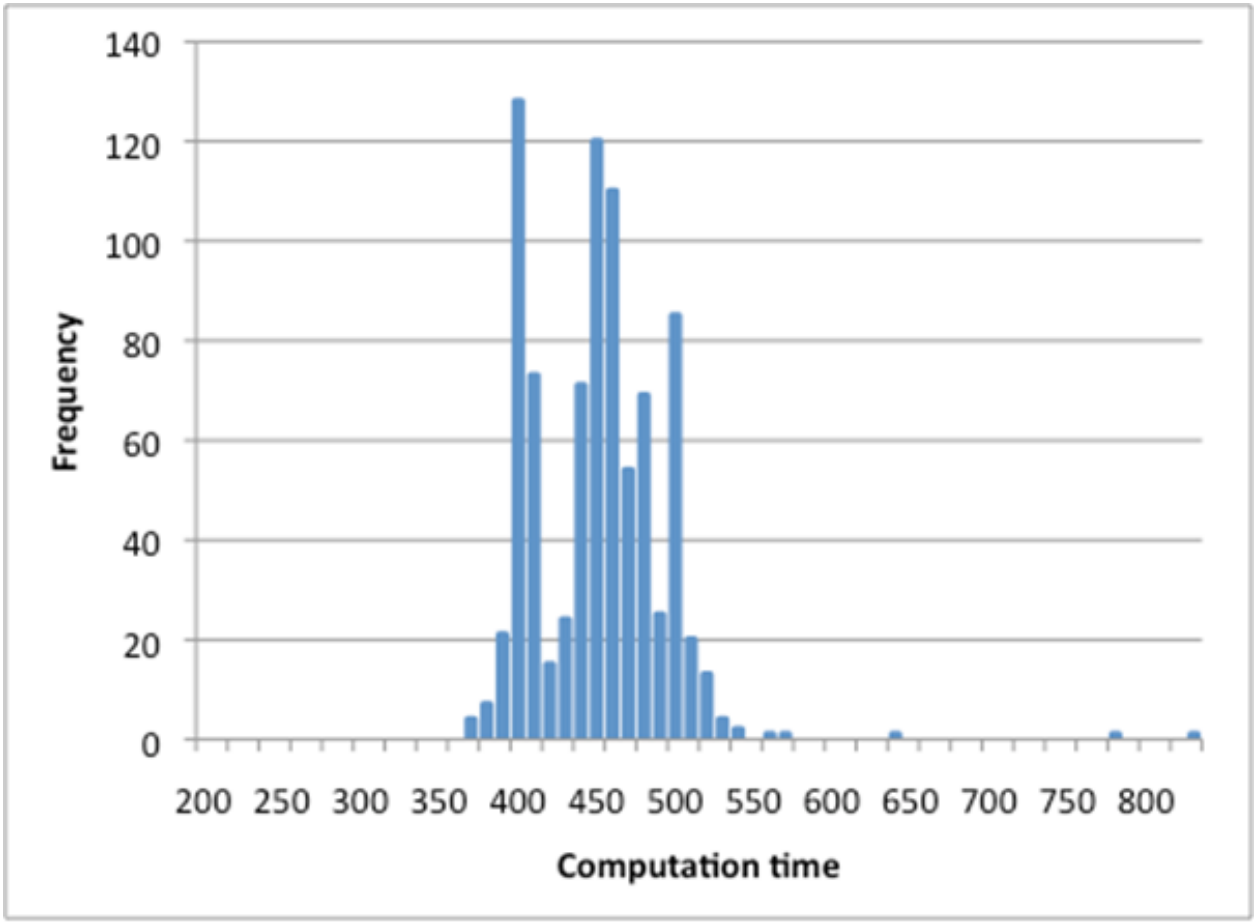


Figure : Distribution of the processing time (in second) for multiple runs of the same computation task for multiple VM.

Outline

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.
- $\Longrightarrow$ Asynchronism
- $\Longrightarrow$ Online or at least mini-batch (no more batch)

Sequential VQ algorithm

- Consists in incremental updates of the $\left(\mathbb{R}^d\right)^\kappa$-valued prototypes $\{w(t)\}_{t=0}^{\infty}$.
- Initiated from a random initial $w(0) \in \left(\mathbb{R}^d\right)^\kappa$.
- 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".

$$H(\mathbf{z}, w) = \left((w_\ell - \mathbf{z}) \, \mathbb{1}_{\left\{ l = \text{argmin}_{i=1,\ldots,\kappa} \|\mathbf{z} - w_i\|^2 \right\}} \right)_{1 \leq \ell \leq \kappa}.$$

$$w(t+1) = w(t) - \varepsilon_{t+1} H\left(\mathbf{z}_{\{t+1 \bmod n\}}, w(t)\right), \quad t \geq 0.$$

Algorithm 2 Sequential VQ algorithm

Select κ initial prototypes $(w_k)_{k=1}^{\kappa}$
Set t=0
repeat
  for $k = 1$ to κ **do**
    compute $||\mathbf{z}_{\{t+1 \bmod n\}} - w_k||_2^2$
  end for
  Deduce $H(\mathbf{z}_{\{t+1 \bmod n\}}, w)$
  Set $w(t+1) = w(t) - \varepsilon_{t+1} H\left(\mathbf{z}_{\{t+1 \bmod n\}}, w(t)\right)$
  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: $\{\mathbf{z}_t^i\}_{t=0}^n$, $i \in \{1, \ldots, M\}$ 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 $w \in \left(\mathbb{R}^d\right)^\kappa$,

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

- 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 : interlaying 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:

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

A first basic parallelization scheme

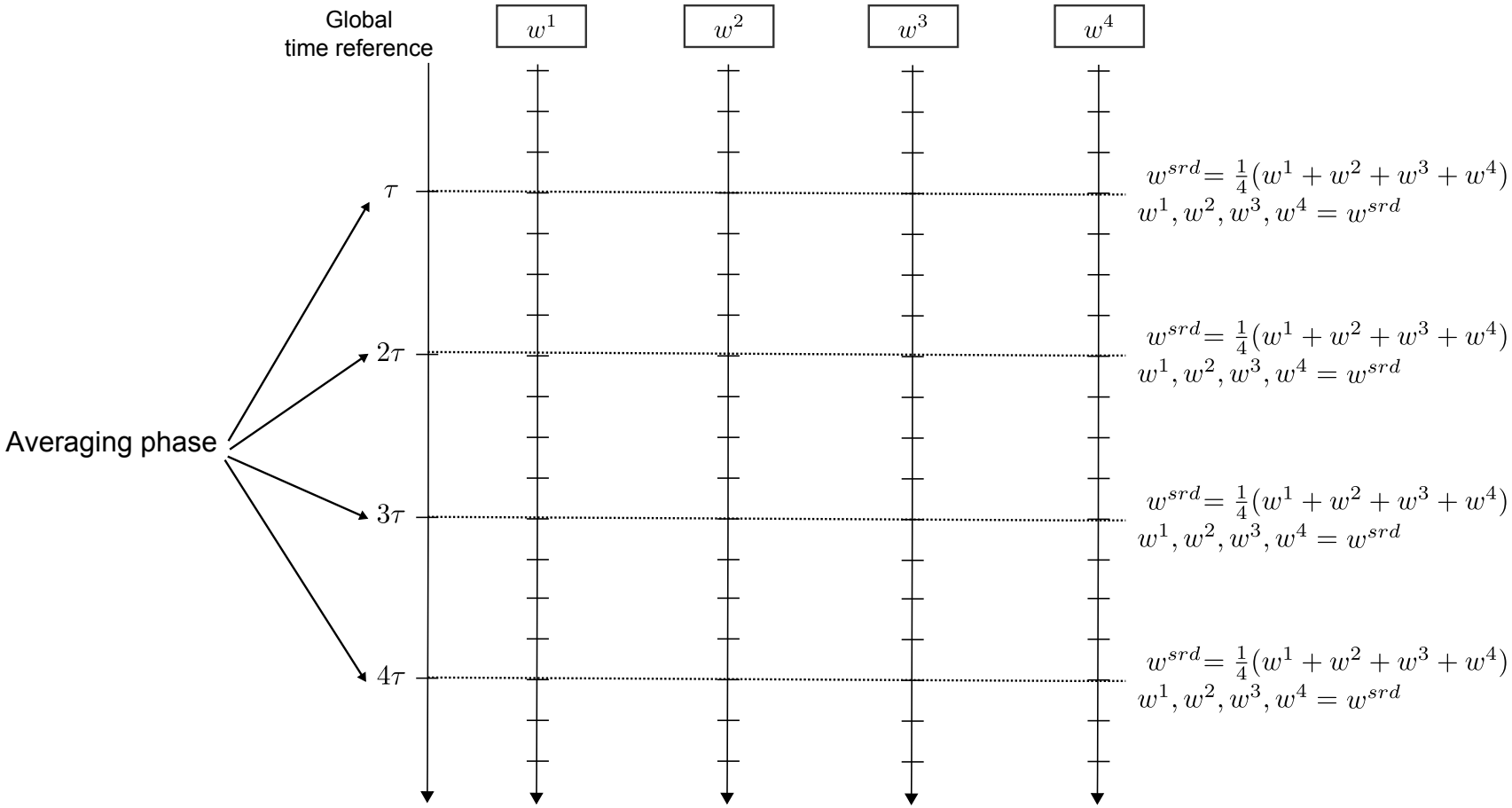


Figure : A simple (and synchronous) scheme: whenever τ points are processed an averaging phase occurs.

A first basic parallelization scheme

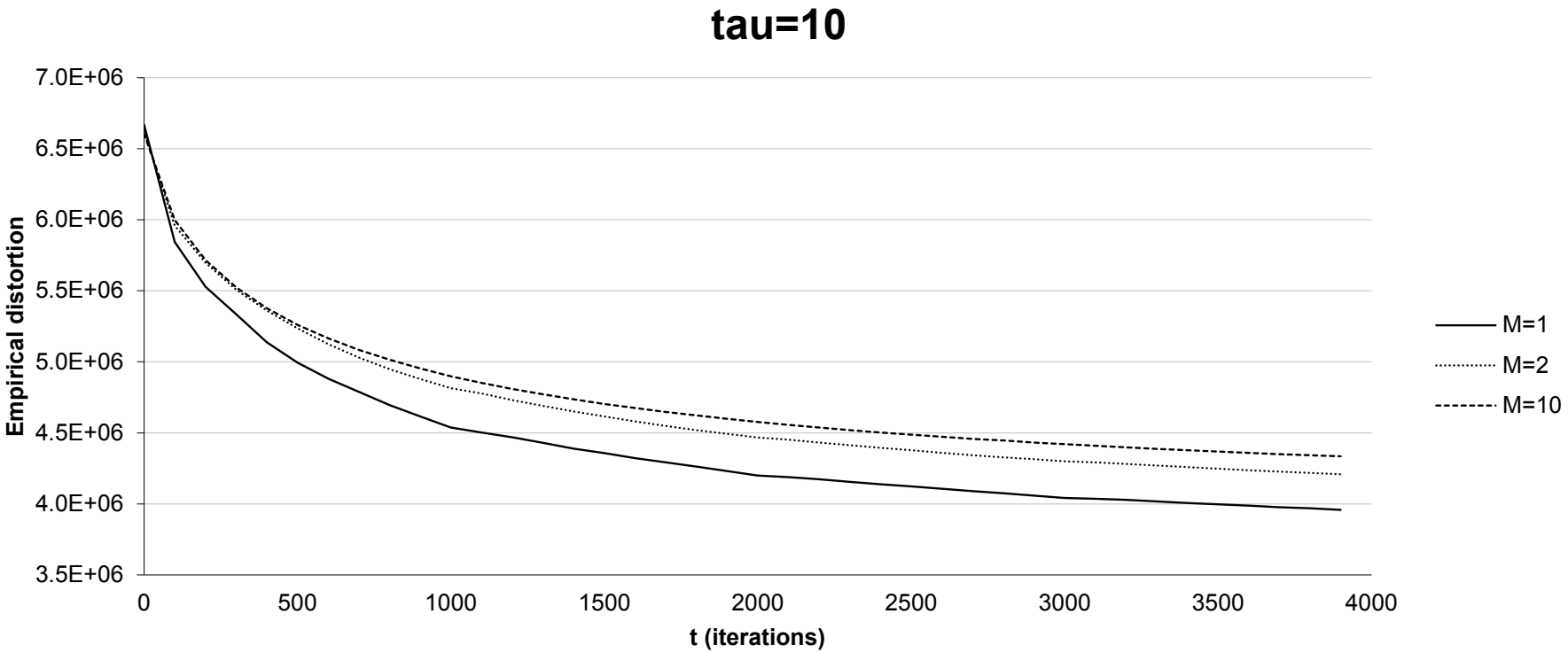


Figure : Charts of performance with different number of computing entities: $M = 1, 2, 10$ and $\tau = 10$.

A comparison between the previous parallel scheme and the sequential VQ

For $t \bmod \tau = 0$ and $t > 0$. Then, for all $i \in \{1, \ldots, M\}$, $w^i(t+1) = w^i(t-\tau+1) - \sum_{t'=t-\tau+1}^{t} \varepsilon_{t'+1} \left(\frac{1}{M} \sum_{j=1}^{M} H\left(\mathbf{z}^j_{t'+1}, w^j(t') \right) \right)$ (parallel)

$w(t+1) = w(t-\tau+1) - \sum_{t'=t-\tau+1}^{t} \varepsilon_{t'+1} H\left(\mathbf{z}_{\{t'+1 \bmod n\}}, w(t') \right)$ (sequential)

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 $\rightarrow$ learning rate $\rightarrow$ trade-off exploration/convergence.

Introducing displacement/descent terms

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

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

corresponds to the displacement of the prototypes computed by j during (t_1, t_2),

Second distributed scheme

$$\begin{cases} w^i_{temp} = w^i(t) - \varepsilon_{t+1} H\left(\mathbf{z}^i_{\{t+1 \bmod n\}}, w^i(t)\right) \\ w^i(t+1) = w^i_{temp} \\ \begin{cases} w^{srd} = w^{srd} - \sum_{j=1}^{M} \Delta^j_{t-\tau \to t} \\ w^i(t+1) = w^{srd} \end{cases} \end{cases} \quad \begin{array}{l} \text{if } t \bmod \tau \neq 0 \text{ or } t = 0, \\ \\ \text{if } t \bmod \tau = 0 \text{ and } t \geq \tau. \end{array}$$

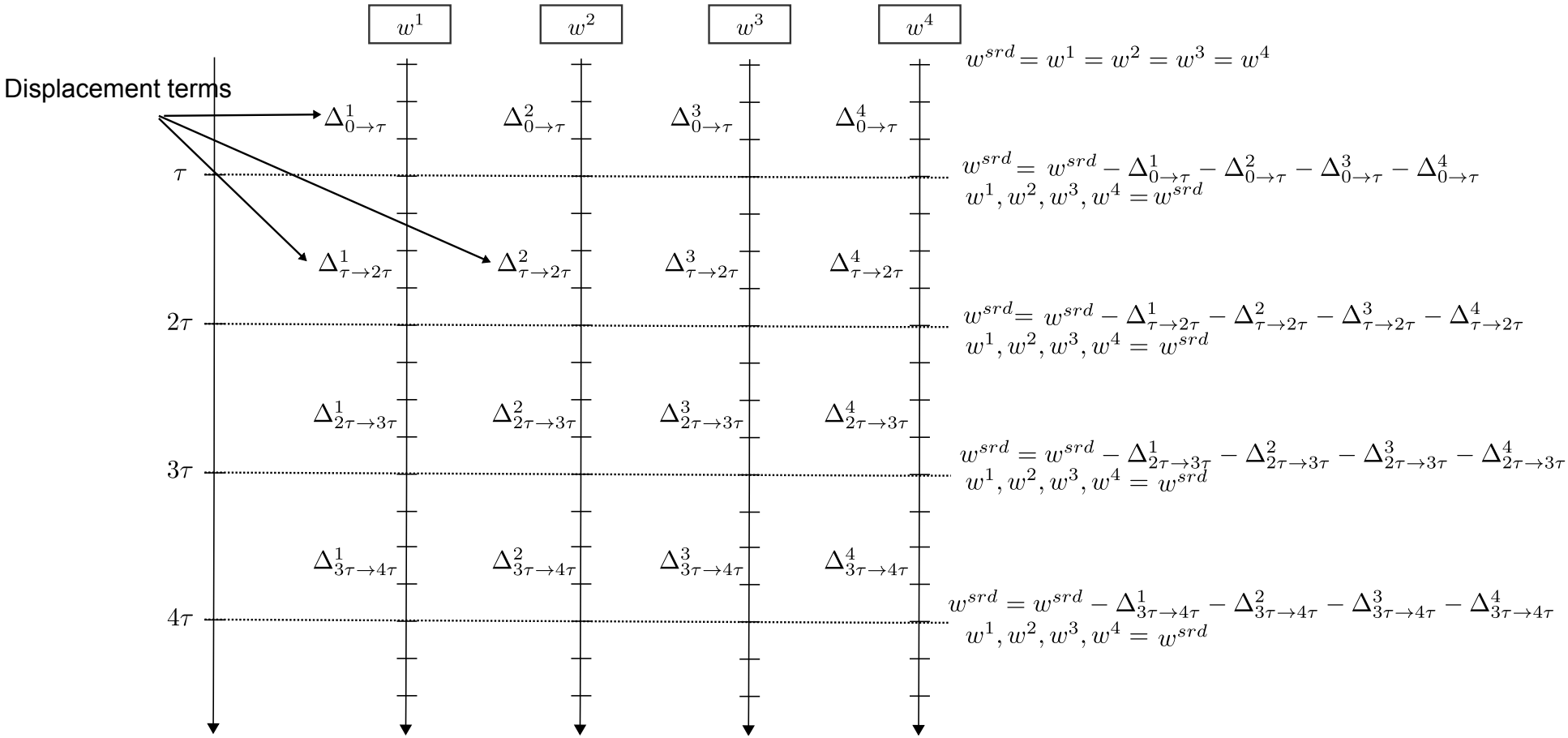


Figure : Illustration of the parallelization scheme of VQ procedures described by equations (43).

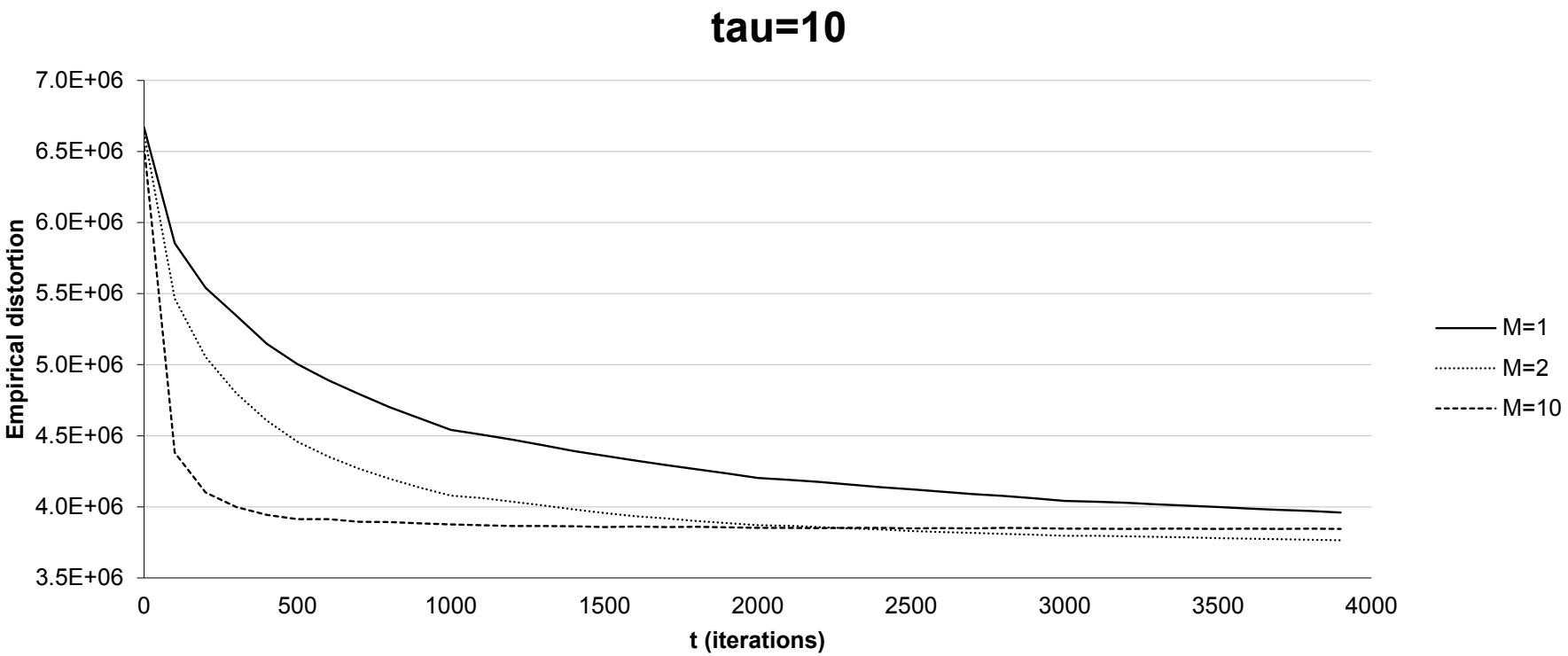


Figure : Charts of performance curves for a reviewed scheme $M = 1, 2, 10$ and $\tau = 10$.

Delayed distributed scheme

$$\begin{cases} w^i_{temp} = w^i(t) - \varepsilon_{t+1} H\left(\mathbf{z}^i_{\{t+1 \bmod n\}}, w^i(t)\right) & \\ w^i(t+1) = w^i_{temp} & \text{if } t \bmod \tau \neq 0 \text{ or } t = 0, \\ \begin{cases} w^{srd} = w^{srd} - \sum_{j=1}^{M} \Delta^j_{t-2\tau \to t-\tau} & \text{if } t \bmod \tau = 0 \text{ and } t \geq 2\tau, \\ w^i(t+1) = w^{srd} - \Delta^i_{t-\tau \to t} & \text{if } t \bmod \tau = 0 \text{ and } t \geq \tau. \end{cases} & \end{cases}$$

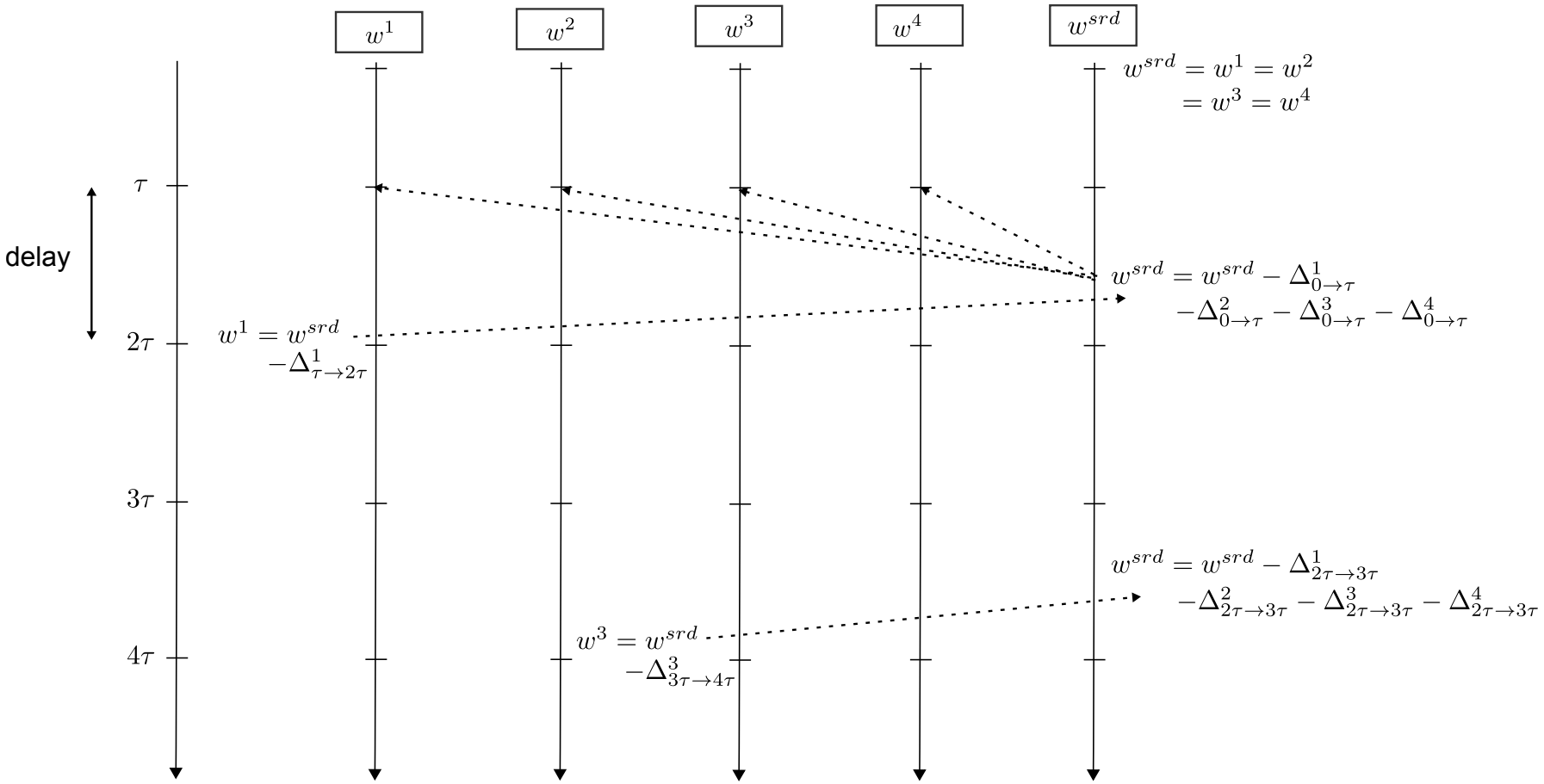


Figure : Illustration of the parallelization scheme described by equations (46). The reducing phase is only drawn for processor 1 where $t = 2\tau$ and processor 4 where $t = 4\tau$.

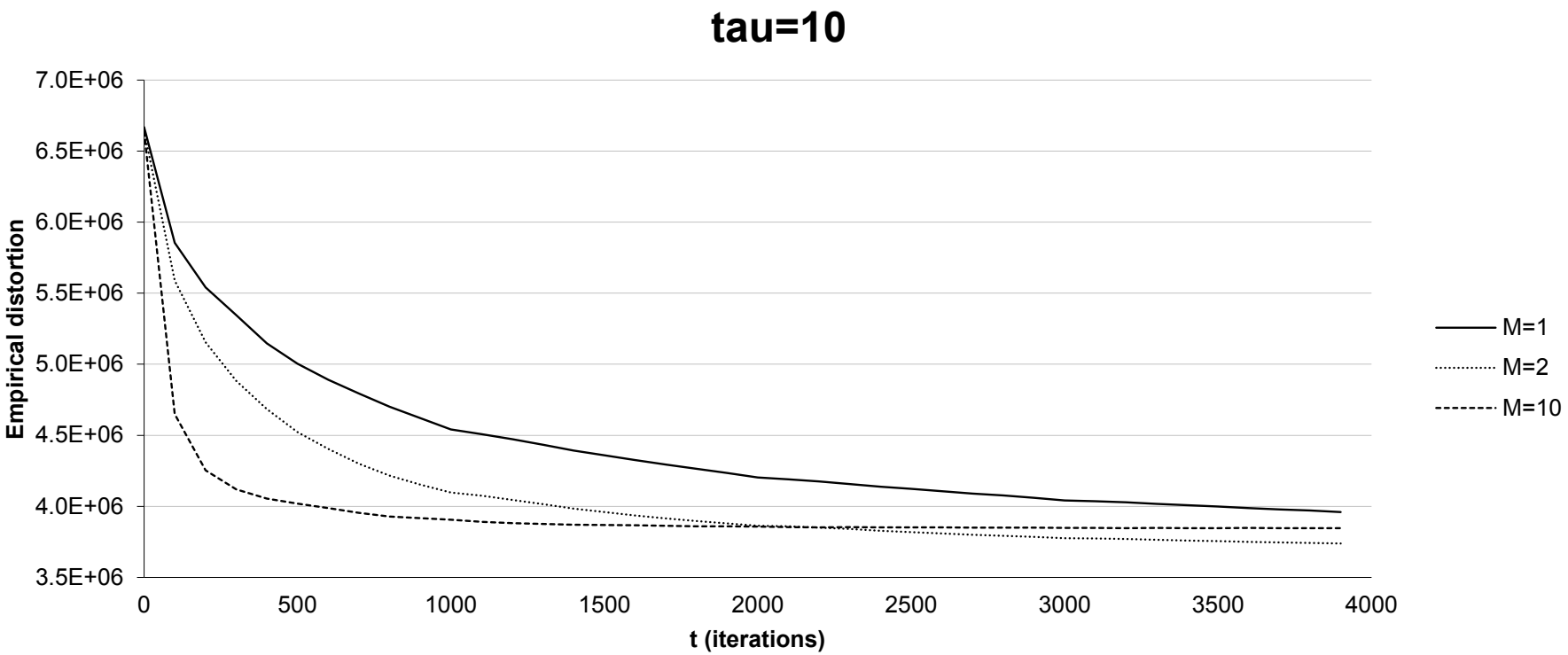


Figure : Charts of performance curves for iterations (46) with different numbers of computing entities, $M = 1, 2, 10$ and $\tau = 10$.

Simulated parallelization schemes first conclusions

Motto: summing displacement term rather than averaging versions.

Experimental results

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

The CloudDALVQ project

- Scientific project for testing new large scale clustering/quantization algorithms distributed on a Cloud Platform (MS Azure).
- Open source written in C#.NET released under new BSD Licence.
  http://code.google.com/p/clouddalvq/

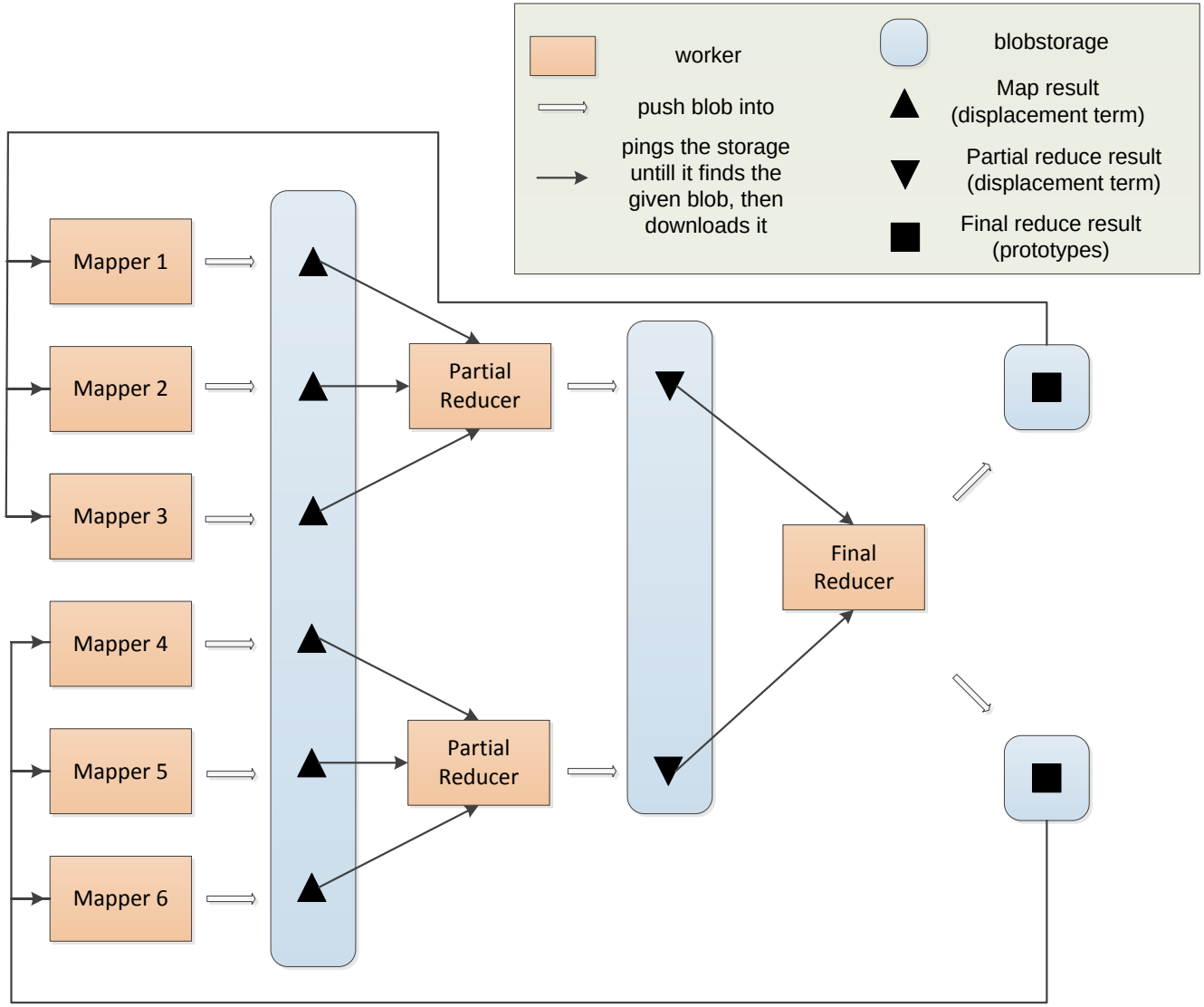
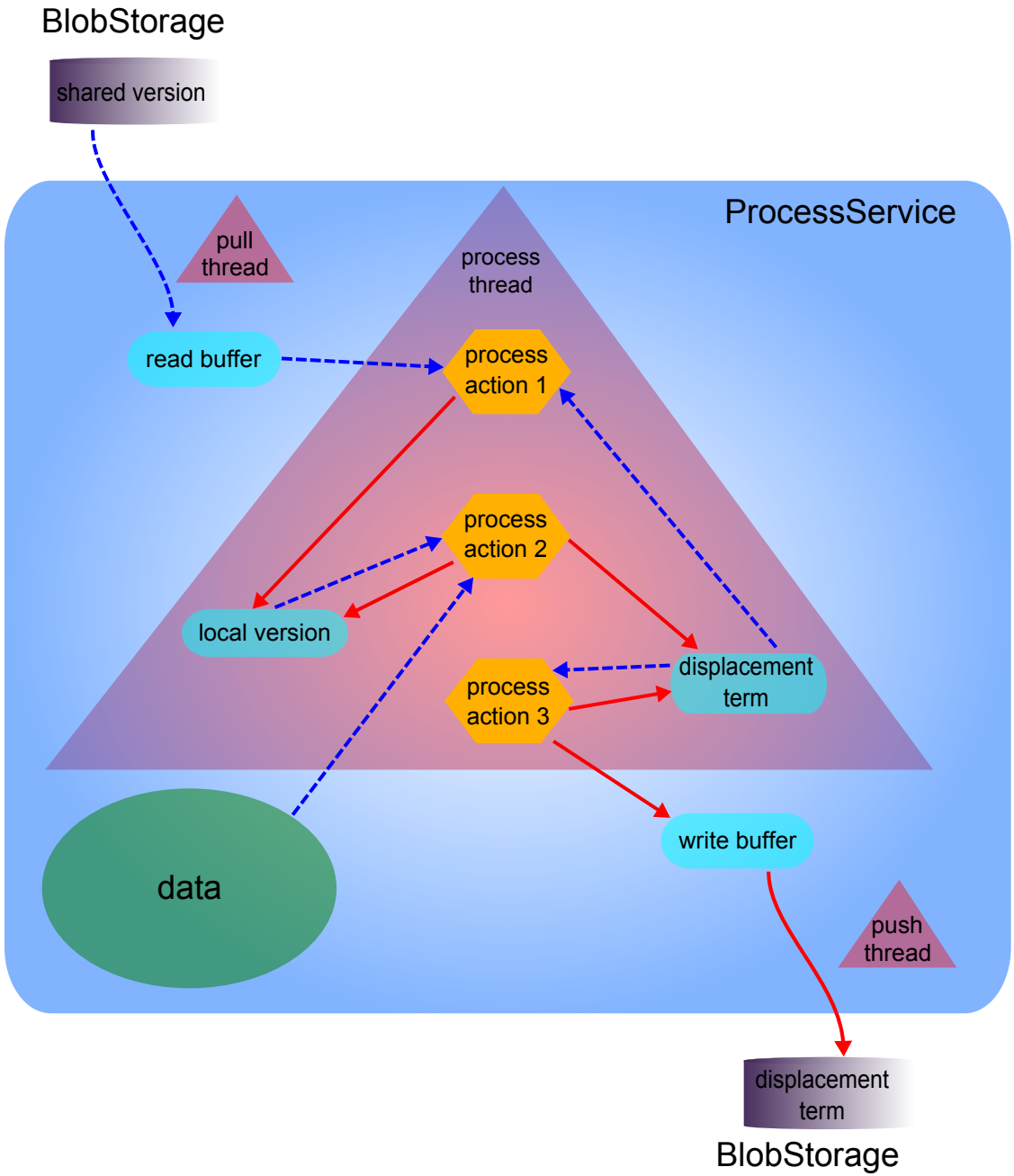


Figure : Distribution scheme of our cloud VQ implementation.


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

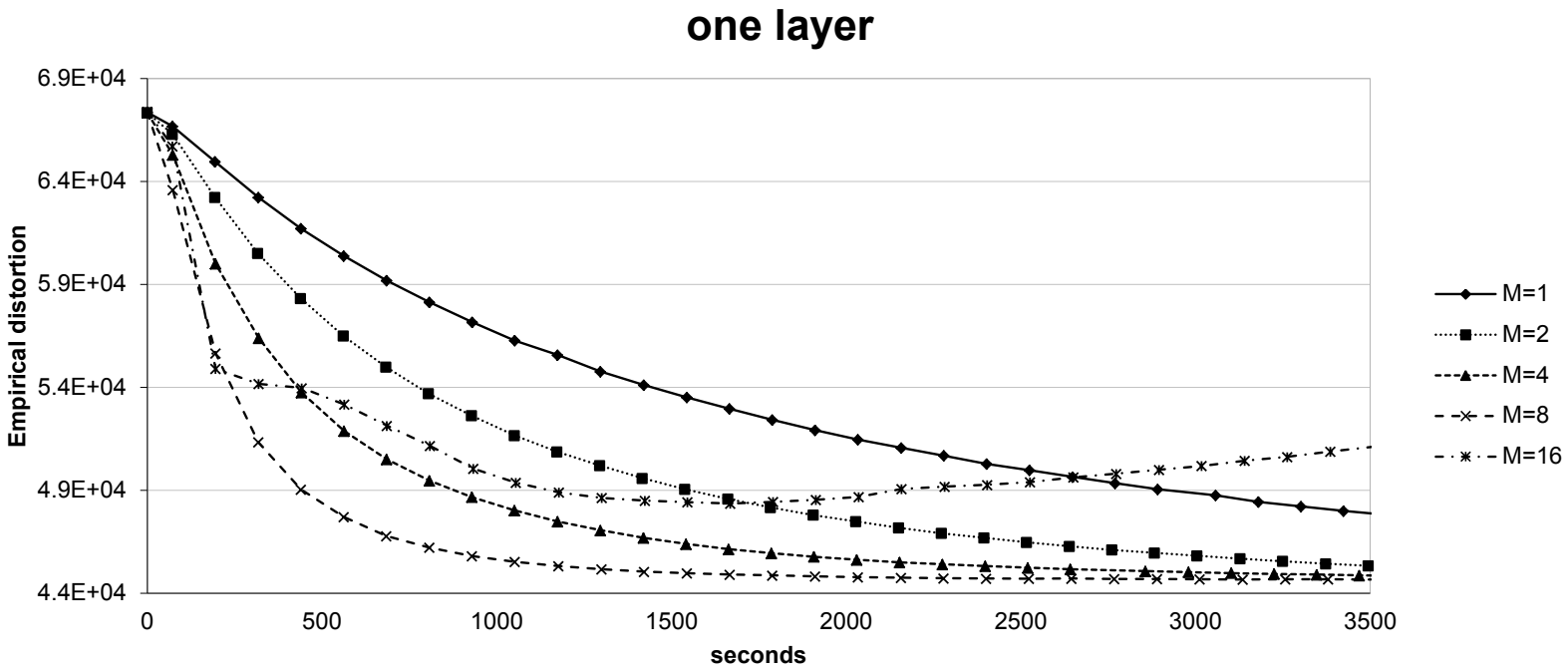


Figure : Normalized quantization curves with $M = 1, 2, 4, 8, 16$. Troubles appear with $M = 16$ because the ReduceService is overloaded.

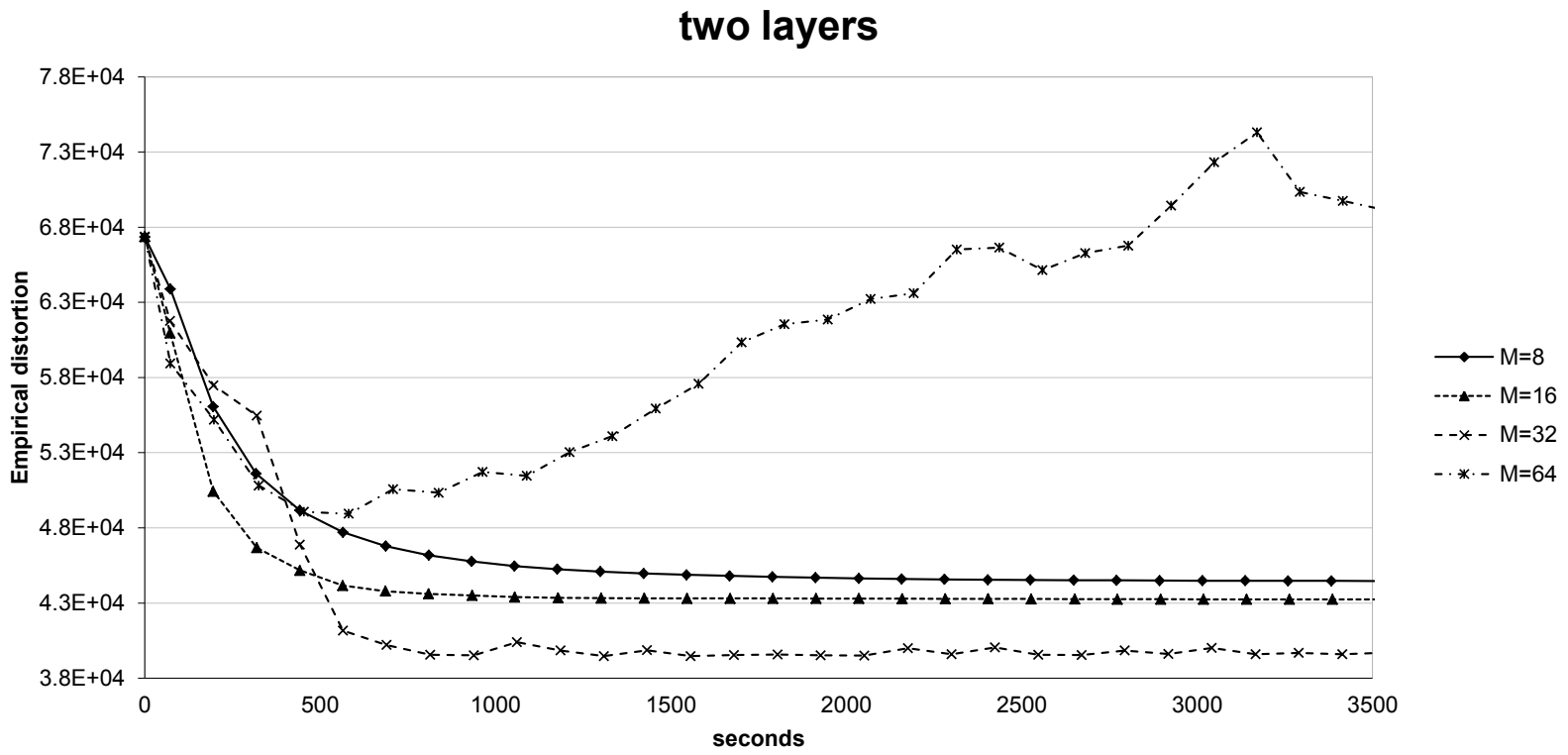


Figure : Normalized quantization curves with $M = 8, 16, 32, 64$ with an extra layer for the so called “reducing task”.

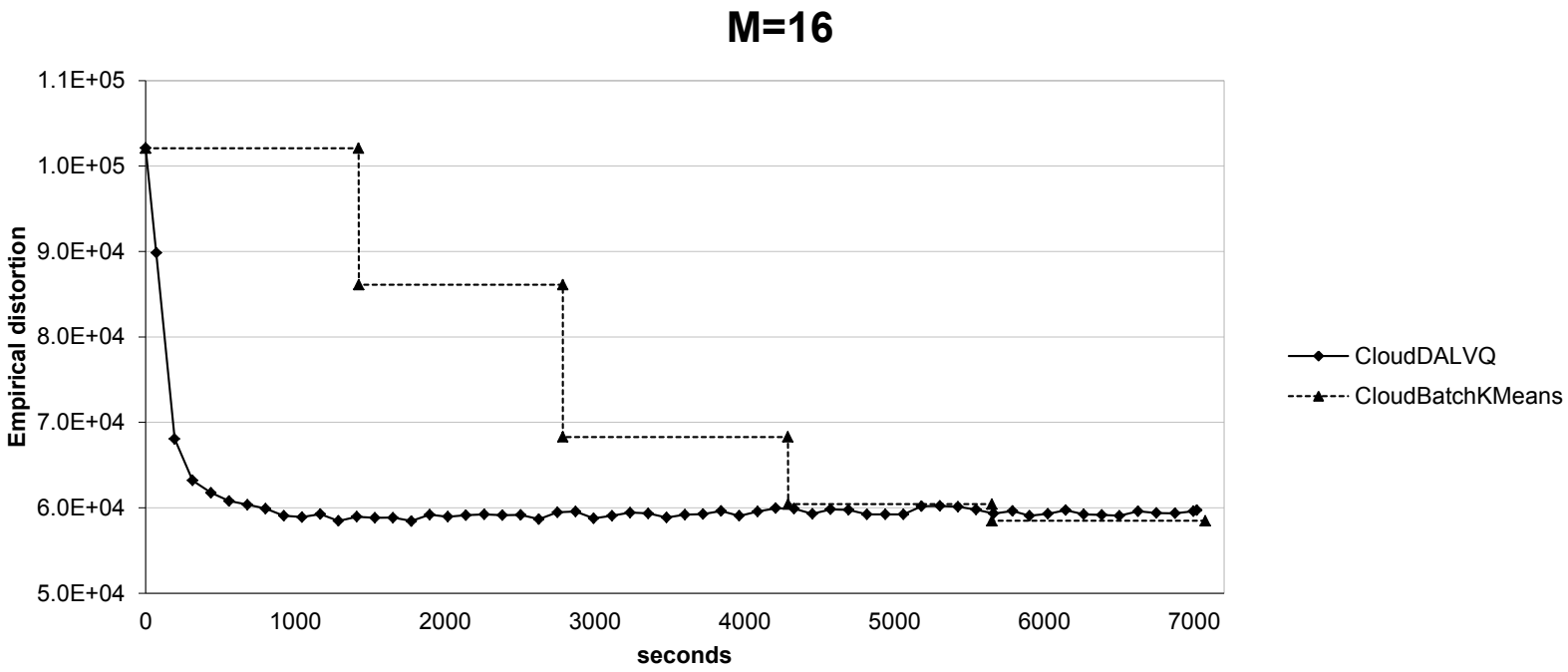


Figure : This chart reports on the competition between our cloud DAVQ algorithm and the cloud Batch K-Means. The graph shows the empirical distortion of the algorithms over the time.