A Discussion on Parallelization Schemes for Stochastic Vector Quantization Algorithms

Matthieu Durut[1,3], Benoit Patra[2,3], Fabrice Rossi[4]

1- Telecom ParisTech - INFRES, 46 rue Barrault - Paris - France

2- Université Pierre et Marie Curie - LSTA, 4 place Jussieu - Paris - France

3- Lokad , 10 rue Philippe de Champaigne - Paris - France

4- SAMM, Université Paris I Panthéon-Sorbonne - 90 rue de Tolbiac - Paris - France

Abstract. This paper studies parallelization schemes for stochastic Vector Quantization algorithms in order to obtain time speed-ups using distributed resources. We show that the most intuitive parallelization scheme does not lead to better performances than the sequential algorithm. Another distributed scheme is therefore introduced which obtains the expected speed-ups. Then, it is improved to fit implementation on distributed architectures where communications are slow and inter-machines synchronization too costly. The schemes are tested with simulated distributed architectures and, for the last one, with Microsoft Windows Azure platform obtaining speed-ups up to 32 Virtual Machines.

1 Introduction

Motivated by the problem of executing clustering algorithms on very large datasets, we investigate parallelization schemes of the stochastic Vector Quantization (VQ) method (also called online k-means). This procedure is known for its good statistical properties but it does not exhibit the embarrassing parallelism of the (batch) k-means. Given a satisfactory sequential implementation of the VQ algorithm, we aim at speeding up its execution through a parallel implementation: the ultimate goal is to reduce the wall time used by the method on a given dataset, that is the time needed to reach some performance threshold, using more than one computing unit. Theoretical parallel VQ algorithms are studied in [1]. The aim of the present paper is to derive actual real world implementations.

The VQ technique computes a summary of a dataset $\{\mathbf{z}_t\}_{t=1}^n$ of d dimensional samples with κ prototypes, $w = (w_1, \ldots, w_\kappa) \in \left(\mathbb{R}^d\right)^\kappa$. Starting from a random initial $w(0) \in \left(\mathbb{R}^d\right)^\kappa$ and given a series of positive *steps* $(\varepsilon_t)_{t>0}$, VQ produces a series of $w(t)$ by updating w prototype by prototype. More precisely, with $l(t) = \operatorname{argmin}_{i=1,\ldots,\kappa} \left\| \mathbf{z}_{\{t+1 \bmod n\}} - w_i(t) \right\|^2$, we have

$$
w(t+1)_i = \begin{cases} w(t)_i & \text{when } i \neq l(t) \\ w(t)_i - \varepsilon_{t+1}(w(t)_i - \mathbf{z}_{\{t+1 \bmod n\}}) & \text{when } i = l(t) \end{cases}, \qquad (1)
$$

where the mod operator stands for the remainder of an integer division operation. A theorem about almost sure convergence of the VQ procedure is proved in [2]. It is well known that the VQ algorithm belongs to the class of stochastic gradient descent algorithms (see [2] for instance).

This paper follows the VQ ideas presented in [1]. We assume having access to M computing entities, each of them executing concurrent VQ procedures. These executions are performed on a dataset, split among the local memory of the computing instances, and represented by the sequences $\{\mathbf{z}_t^i\}_{t=1}^n$, $i \in \{1, \ldots, M\}$. The prototype iterations computed by the VQ techniques on each node are denoted by $\{w^i(t)\}_{t=0}^\infty$ and called *versions*. We use the following normalized criterion to measure the speed-up ability of our investigated schemes.

$$C_{n,M}(w) = \frac{1}{nM} \sum_{i=1}^{M} \sum_{t=1}^{n} \min_{\ell=1,\ldots,\kappa} \left\| \mathbf{z}_t^i - w_\ell \right\|^2, \quad w \in \left(\mathbb{R}^d\right)^\kappa. \tag{2}$$

The rest of the paper is organized as follows. First, Section 2 provides empirical evidences that the most simple scheme cannot bring speed-ups. Then, some insights to explain the previous non satisfactory situation are provided in Section 3. Consequently, we design a new scheme and prove by practice its ability to bring speed-ups. Finally, in Section 4, we present an asynchronous adaptation of this latter scheme which fits better slow communication architectures such as Cloud Computing.

Notice that the proposed algorithms are tested using simulated distributed architecture and synthetic vector data[1], but our conclusions are more sensitive to the loss function smoothness and convexity than to the data choice.

2 A first distributed scheme

Our investigation starts with the most intuitive parallelization scheme. Each computing resource starts with the same initial prototypes (a.k.a., versions): $w^1(0) = \ldots = w^M(0)$. Then each machine applies the sequential VQ to its subset of the dataset. Once in a while, prototypes are synchronized: when τ data points have been processed by each concurrent processor, a shared version of the prototypes is computed as follows (here for the first synchronization event):

$$w^{srd} = \frac{1}{M} \sum_{j=1}^{M} w^j(\tau). \tag{3}$$

The shared version is then broadcasted to each processing unit. In the case of a smooth convex loss function, distributed stochastic gradient descent algorithms with averaging of local results provide a speed-up in comparison of the sequential algorithm (see [3]). Figure 1 shows a typical evolution through wall time of the quantization error, obtained with an execution of the scheme on a simulated parallel implementation in which communications are instantaneous. It shows up that in our non smooth and convex loss function case, multiple resources do not bring speed-ups for convergence. Even if more data are processed, no gain in term of wall clock time is provided using this parallel scheme.

---

[1]The source code is available at `http://code.google.com/p/clouddalvq/`. Details about the artificial data generator are available in Section 4.2 of `http://www.lsta.upmc.fr/doct/patra/publications/PhDMain.pdf`.
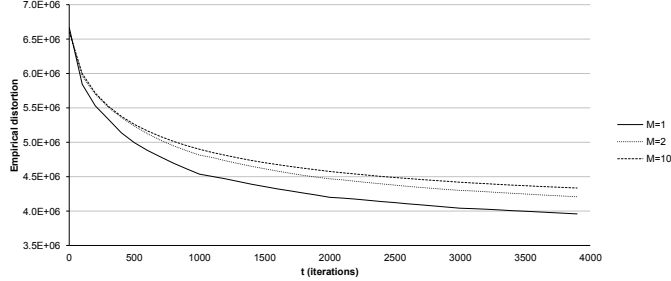
Figure 1: Charts of performance curves for iterations (3) with $\tau = 10$ and different number of computing entities: M = 1, 2, 10.

3 Towards a better scheme

The investigation of the previous non-satisfactory result starts by rewriting both the sequential and the distributed scheme. Let us first introduce $H(\mathbf{z}, w)$ defined by

$$H(\mathbf{z}, w) = \left((w_\ell - \mathbf{z}) \mathbb{1}_{\left\{ l = \mathrm{argmin}_{i=1,\dots,\kappa} \|\mathbf{z} - w_i\|^2 \right\}} \right)_{1 \le \ell \le \kappa}. \tag{4}$$

Then, a series of the sequential VQ iterations (1) can be rewritten:

$$w(t+1) = w(t - \tau + 1) - \sum_{t'=t-\tau+1}^{t} \varepsilon_{t'+1} H\left(\mathbf{z}_{\{t'+1 \bmod n\}}, w(t') \right), \quad t \ge \tau. \tag{5}$$

Then, just after a synchronization (defined by $t \bmod \tau = 0$ and $t > 0$), for all $i \in \{1, \dots, M\}$, the sequential VQ iterations on each computational resource can be rewritten as follows

$$w^i(t+1) = w^i(t - \tau + 1) - \sum_{t'=t-\tau+1}^{t} \varepsilon_{t'+1} \left(\frac{1}{M} \sum_{j=1}^{M} H\left(\mathbf{z}^j_{\{t'+1 \bmod n\}}, w^j(t') \right) \right). \tag{6}$$

Assuming that $w^j(t') \approx w^i(t')$, for all $(i, j) \in \{1, \dots, M\}^2$ and $t' \ge 0$, the mean in parenthesis is an estimator of the gradient of the distortion at $w^i(t')$. Consequently, the two algorithms induced by iterations (5) and (6) can be thought as stochastic gradient descent procedures with different estimators of the gradient but driven by the same learning rate which is given by the sequence $(\varepsilon_t)_{t>0}$.

The convergence speed of a non-fixed step gradient descent procedure is essentially driven by the decreasing speed of the sequence of steps. The choice of this sequence is subject to an exploration/convergence trade-off. Since the two procedures above share the same learning rate with respect to the iterations $t \ge 0$, they share the same convergence speed with respect to the wall clock time (time measured by an exterior observer). Yet, the distributed scheme of Section 2

has a much lower learning rate with respect to the number of samples processed, favoring exploration to the detriment of the convergence. The multiple resources therefore lead to better exploration but to similar convergence speed with respect to wall clock time.

As we assume to have a satisfactory VQ implementation, the series of steps $(\varepsilon_t)_{t>0}$ is supposed to be adapted to the dataset. Consequently we should seek for a distributed scheme that have the same learning rate evolution in term of processed samples and which convergence speed with respect to iterations is accelerated. Denoting

$$\Delta^j_{t_1 \to t_2} = \sum_{t'=t_1+1}^{t_2} \varepsilon_{t'+1} H\left(\mathbf{z}^j_{\{t'+1 \bmod n\}}, w^j(t')\right), \; j \in \{1, \ldots, M\} \text{ and } t_2 > t_1 \geq 0.$$

(7)

At time $t = 0$, $w^1(0) = \ldots = w^M(0) = w^{srd}$. For all $i \in \{1, \ldots, M\}$ and all $t \geq 0$, consider the distributed scheme given by

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

(8)

The main difference between the two parallel schemes consists in the way results are merged in the reducing phase (described by the braced inner equations): here we apply the translation calculated by each parallel VQ to the current shared version of the prototypes, rather than averaging this translation. The results of a typical application of this scheme are displayed in the charts of Figure 2. The charts show that substantial speed-ups are obtained with distributed resources. The acceleration is greater when the reducing phase is frequent. Indeed, if τ is large then more autonomy has been granted to the concurrent executions, they could be attracted to different regions that would slow down the consensus and the convergence.

4 A model with stochastic delays

The previous parallelization schemes do not deal with communication costs introduced by update exchanges between machines. In the context of cloud computing, no efficient shared memory is available and these costs introduce delays. The effect of delays for parallel stochastic gradient descent has already been studied (see for instance [4]) but for a computing architecture endowed with an efficient shared memory. Moreover, the unreliability of the cloud computing hardware introduces strong straggler issues and makes the synchronization process inappropriate. In this subsection, we improve the model of iterations (8) with random communication costs that follow a geometric distribution and we remove the synchronization process of reducing phase, resulting in the more realistic
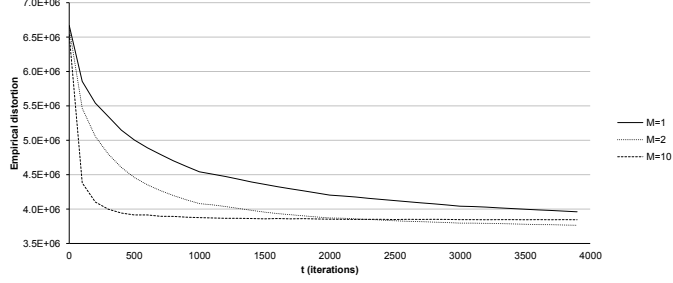
Figure 2: Charts of performance curves for iterations (8) with $\tau = 10$ and different number of computing entities: M = 1, 2, 10.

iterations (9) below. For each time $t \geq 0$, let $\tau^i(t)$ be the latest time before t when the unit i finished to send its updates and received the shared version. At time $t = 0$ we have $w^1(0) = \ldots = w^M(0) = w^{srd}$, and for all $i \in \{1, \ldots, M\}$ and all $t \geq 0$,

$$
\begin{cases}
w^i_{temp} = w^i(t) - \varepsilon_{t+1} H\left(\mathbf{z}^i_{\{t+1 \bmod n\}}, w^i(t)\right) & \\
w^i(t+1) = w^i_{temp} & \text{if } t \neq \tau^i(t) \\
w^i(t+1) = w^{srd}(\tau^i(t-1)) - \Delta^i_{\tau^i(t-1) \to t} & \text{if } t = \tau^i(t) \\
w^{srd}(t+1) = w^{srd}(t) - \sum_{j,t=\tau^j(t)} \Delta^j_{\tau^j(\tau^j(t-1)-1) \to \tau^j(t-1)} &
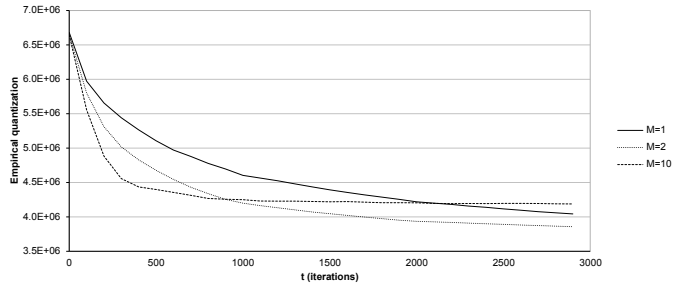\end{cases}
\tag{9}
$$

Figure 3: Charts of performance curves for iterations (9) with $\tau = 10$ and different number of computing entities: M = 1, 2, 10.

There are no more synchronization between processing units: each machine uploads its updates and downloads the shared version as soon as its previous uploads and downloads are completed. A dedicated unit permanently modifies

the shared version with the latest updates received from the other machines without any synchronization barrier. The Figure 3 shows that the introduction of small delays and asynchronism only slightly impacts performances, compared to the scheme given by equations (8). The Figure 4 shows the results obtained by our cloud implementation[2] of the iterations (9) using 32 real processing units. A future paper will describe more precisely this cloud implementation.
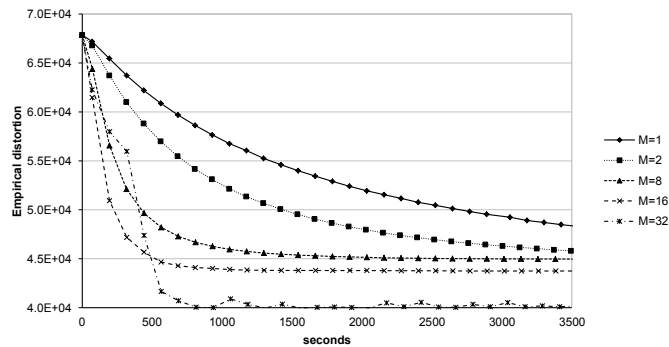

Figure 4: Charts of performance curves for iterations (9) on our cloud implementation and different number of computing entities.

5 Conclusion

In this paper we show that the naive parallelization scheme proposed in Section 2 does not provide better performance than the sequential scheme. This surprising result derives from the fact that our first parallel scheme leads to a decrease of the learning rate per data points processed. We therefore propose a new parallelization scheme relying on asynchronous updates of a common "shared version". This latter algorithm is very well suited for parallel computation on slow communication networks such as cloud computing platforms. Our implementation on Azure shows significant scale-up, up to 32 machines.

References

[1] Benoît Patra. Convergence of distributed asynchronous learning vector quantization algorithms. *Journal of Machine Learning Research 12 (2011)*, pages 3431–3466, 2011.

[2] Gilles Pagès. A space vector quantization for numerical integration. *Journal of Applied and Computational Mathematics*, 89:1–38, 1997.

[3] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *CoRR*, abs/1012.1367, 2010.

[4] Martin Zinkevich, Alex Smola, and John Langford. Slow learners are fast. In *Advances in Neural Information Processing Systems 22*, pages 2331–2339. 2009.

---

[2]`http://code.google.com/p/clouddalvq/`