Université Pierre et Marie Curie – Paris VI

T H È S E

Présentée pour obtenir

Le titre de docteur en sciences de
l'université Pierre et Marie Curie – Paris VI

Spécialité : Mathématiques

Ecole Doctorale : Sciences Mathématiques de Paris Centre

Apprentissage à « grande échelle » : contribution à l'étude d'algorithmes de clustering répartis asynchrones.

par

Benoît PATRA

Soutenue le 5 mars 2012 devant le jury composé de :

M. Gérard BIAU	Professeur, Université Paris VI	Directeur de thèse
M. Léon BOTTOU	Scientist partner, Microsoft	Rapporteur
Mme Barbara HAMMER	Professeur, Université de Clausthal	Rapporteur
M. Gilles PAGÈS	Professeur, Université Paris VI	Président
M. Fabrice ROSSI	Professeur, Université Paris I	Examinateur
M. Joannès VERMOREL	Fondateur de la société Lokad	Examinateur

Rapporteurs :

M. Léon BOTTOU	Scientist partner, Microsoft	Rapporteur
Mme Barbara HAMMER	Professeur, Université de Clausthal	Rapporteur

Résumé

Les thèmes abordés dans ce manuscrit de thèse sont inspirés de problématiques de recherche rencontrées par la société Lokad, qui sont résumées dans le premier chapitre. Le Chapitre 2 est consacré à l'étude d'une méthode non paramétrique de prévision des quantiles d'une série temporelle. Nous démontrons, en particulier, que la technique proposée converge sous des hypothèses minimales. La suite des travaux porte sur des algorithmes de clustering répartis et asynchrones (DALVQ). Ainsi, le Chapitre 3 propose tout d'abord une description mathématique de ces modèles précédent, et se poursuit ensuite par leur étude théorique. Notamment, nous démontrons l'existence d'un consensus asymptotique et la convergence presque sûre de la procédure vers des points critiques de la distortion. Le chapitre suivant propose des réflexions ainsi que des expériences sur les schémas de parallélisation à mettre en place pour une réalisation effective des algorithmes de type DALVQ. Enfin, le cinquième et dernier chapitre présente une implémentation de ces méthodes sur la plate-forme de *Cloud Computing* `Microsoft Windows Azure`. Nous y étudions, entre autres thèmes, l'accélération de la convergence de l'algorithme par l'augmentation de ressources parallèles. Nous le comparons ensuite avec la méthode dite de Lloyd, elle aussi répartie et déployée sur `Windows Azure`.

Mots clés : séries temporelles, prévision quantile, perte pinball, agrégation d'experts, k-means, quantification vectorielle, calcul réparti, asynchronisme, consensus réparti, Cloud Computing, Windows Azure.

"Large scale" learning : a contribution to distributed asynchronous clustering algorithms analysis.

Abstract

The subjects addressed in this thesis manuscript are inspired from research problems encountered by the company Lokad, which are summarized in the first chapter. Chapter 2 deals with a nonparametric method for forecasting the quantiles of a real-valued time series. In particular, we establish a consistency result for this technique under minimal assumptions. The remainder of the dissertation is devoted to the analysis of distributed asynchronous clustering algorithms (DALVQ). Chapter 3 first proposes a mathematical description of the models and then offers a theoretical analysis, where the existence of an asymptotical consensus and the almost sure convergence towards critical points of the distortion are proved. In the next chapter, we propose a thorough discussion as well as some experiments on parallelization schemes to be implemented for a practical deployment of DALVQ algorithms. Finally, Chapter 5 contains an effective implementation of DALVQ on the *Cloud Computing* platform `Microsoft Windows Azure`. We study, among other topics, the speed ups brought by the algorithm with more parallel computing ressources, and we compare this algorithm with the so-called Lloyd's method, which is also distributed and deployed on `Windows Azure`.

Keywords: time series, quantile forecasting, pinball loss, experts aggregation, k-means, vector quantization, distributed computing, asynchronous, distributed consensus, Cloud Computing, Windows Azure.

Remerciements

Je tiens tout d'abord à remercier mon directeur de thèse Gérard Biau. Il a été à la fois un mentor scientifique et un précieux conseillé dans de nombreuses situations. J'ai eu le privilège de recevoir de mon directeur un encadrement idéal ce qui n'est pas chose aisée dans le contexte particulier de la thèse CIFRE. Gérard a su se montrer toujours disponible dans la bonne humeur et est parvenu à trouver le subtil équilibre entre un encadrement proche et une liberté d'initiative dans la recherche.

Je remercie Joannès Vermorel fondateur de la société Lokad qui m'a appris énormément sur le plan professionnel notamment dans le secteur du logiciel. Joannès a totalement accepté les règles du jeu d'une thèse CIFRE et m'a laissé le temps nécessaire pour le déroulement des travaux tout en me promulgant des conseils avisés. Une partie de cette thèse est aussi le fruit des efforts de Matthieu Durut, ami, collègue et collaborateur scientifique hors pair. Les travaux présentés ont aussi bénéficié des précieux conseils de Fabrice Rossi et de Jean-Claude Fort.

Au moment de soutenir cette thèse, j'éprouve une forte reconnaissance pour mon premier maître mathématique, Denis Choimet, alors professeur de mathématiques en classe préparatoire au lycée Clémenceau. Je remercie également les professeurs de l'antenne de Bretagne de l'ENS Cachan en particulier Arnaud Debussche, Michel Pierre et Gregory Vial. Parmi les rennais, j'ai une pensée pour Florent Malrieu qui a su me donner le goût des mathématiques de l'aléatoire.

La recherche en mathématiques, malgré l'image qu'elle peut véhiculer, est une activité sociale. J'ai donc été naturellement influencé par mes camarades d'école avec qui j'ai travaillé : Brot, Guillaume, Gwen, Pitroll, Léo, Anne et Maher. Je réserve une mention spéciale à mes "colloc' stateux", Benjamin Favetto et Adrien Saumard, qui m'ont aussi aidé sur certains points techniques de cette thèse.

Il est temps pour moi de remercier mes proches : ma compagne Nina ainsi que mon père, ma mère, ma grand mère, Danièle, mes frères Arnaud et Nicolas et ma soeur Axelle. Je remercie aussi mon oncle François et mes tantes Dominique et Catherine ainsi que mes cousins.

Je tiens aussi à écrire que j'ai reçu le soutien de mes amis "non matheux" : Brice, Clémence, Sophie, Rémi, Gregoire, Goustan, Matthieu, Alexandre, Benjamin, Antoine, Olivier, Simon, Foucault, Marina, Robby et tous les autres.

Table des matières

1 Introduction et synthèse des résultats

1.1 Contexte scientifique et professionnel de la thèse

Cette thèse est le fruit d'un partenariat entre la PME Lokad et le Laboratoire de Statistique Théorique et Appliquée (LSTA) de l'Université Pierre et Marie Curie (UPMC). Lokad est une société éditrice de logiciels spécialisée dans les prévisions statistiques à grande échelle. Les applications proposées aux entreprises clientes sont des applications web en mode SaaS, c'est-à-dire accessibles directement depuis l'internet. Les secteurs d'activités auxquels s'adressent les applications Lokad sont variés. Par exemple, l'application `Salescast` est utilisée pour l'optimisation des stocks de détaillants, grossistes ou revendeurs e-commerce, tandis que `CallCalc` est dédiée à l'optimisation du personnel des centres d'appels et donc utile pour les banques, les assurances, les opérateurs télécoms etc. Les applications développées par Lokad fonctionnent de manière entièrement automatisée, si bien qu'aucune expertise statistique n'est nécessaire pour leur utilisation. Pour parvenir à cette automatisation Lokad importe, via le réseau internet, les données et réalise d'importants calculs sur ses propres serveurs avant de renvoyer les résultats à travers ses applications. Ces aspects des services proposés par Lokad se résument bien dans la devise commerciale de la société : *Your data our burden*[1].

De manière générale, les données des entreprises clientes sont gérées sous forme de séries temporelles. Lokad est donc doté d'un important moteur de prévision de séries temporelles entièrement automatisé. A titre d'exemple, dans le cadre de l'optimisation des stocks, pour chaque unité de vente, il existe une série chronologique de l'historique des réserves du dépôt. Il n'est donc pas rare qu'un jeu de données soit composé de plusieurs centaines de milliers de séries temporelles. Ainsi, ce moteur se doit non seulement de fournir des prévisions de bonne qualité, mais également de manipuler de grandes quantités de données dans un laps de temps ne dépassant pas une heure. Il a donc fallu adapter les ressources informatiques pour pouvoir réaliser de très gros calculs en parallèle. C'est donc naturellement que l'entreprise Lokad s'est tournée vers des technologies de type *Cloud Computing*. En un mot,

1. Vos données notre fardeau.

il s'agit de services proposant des accès à des machines distantes, dont le nombre peut être rapidement adapté. En outre, ces services reposent le plus souvent sur une facturation dite « à l'utilisation ». Depuis la fin de l'année 2010, l'intégralité des applications Lokad, à usage public ou interne, sont en déploiement sur la plate-forme de *Cloud Computing* `Microsoft Windows Azure`. L'élasticité de ce type de technologie permet de répondre à d'importants besoins de calcul ponctuels. De cette manière, Lokad, PME du secteur de l'informatique décisionnelle, peut en quelques minutes avoir accès à des moyens de calcul qui n'étaient auparavant accessibles qu'aux seules grandes institutions. Cette flexibilité dans l'allocation des ressources permet en outre de ne pas se voir facturer celles-ci lorsqu'elles ne sont pas utilisées.

Les thèmes abordés dans ce manuscrit sont tous profondément inspirés de problématiques de recherche et de développement nées chez Lokad. C'est ainsi que la première partie de la thèse porte sur l'analyse mathématique et expérimentale d'une méthode de prévision de quantiles de séries temporelles. La prévision quantile dans le cadre de séries chronologiques est un thème important pour Lokad car elle permet de nombreuses optimisations fines souvent négligées. Par exemple, l'optimisation des stocks dans les entrepôts peut être réalisée à partir de prévisions quantiles afin de garantir une disponibilité des produits avec un certain niveau de confiance. Citons également le cas des centres d'appels, où la gestion des équipes peut être effectuée grâce à des prévisions quantiles permettant alors un ajustement fin des taux de service. Ces premiers travaux ont donné lieu à une publication avec Gérard Biau [15] dans le journal *IEEE Transactions on Information Theory* et font l'objet du Chapitre 2 du présent manuscrit.

La suite des travaux présentés dans ce document de synthèse sont le résultat d'une longue réflexion débutée au sein de l'entreprise autour de problématiques liée au calcul réparti [2]. En effet, pour améliorer ses prévisions de séries temporelles, Lokad utilise des méthodes dites « multi-séries ». Ces modèles ont la particularité de s'entraîner sur des blocs de séries dans le but d'affiner la qualité des prévisions. En exploitant des composantes saisonnières calculées de cette manière, nous sommes parvenus à améliorer fortement les prévisions de Lokad dans le secteur de la grande distribution. Cependant, ces groupements de séries doivent être réalisés de manière pertinente, au travers d'algorithmes dits de *clustering*. On comprend donc que, parmi les problématiques scientifiques soulevées par Lokad, ces algorithmes tiennent une place de choix. Nous avons donc naturellement été amenés à réfléchir à la parallélisation de ces algorithmes sur des architectures de type *Cloud Computing*. Certaines réalisations antérieures, comme le projet `Cloudster` [3], mon-

2. Traduction de l'anglais *distributed computing*.
3. Projet logiciel démarré en 2009 par les étudiants de première année du département d'in-

traient des goulots d'étranglement provoqués par les phases de synchronisation du célèbre algorithme de *batch k-means*. Ces problèmes nous ont alors conduit à amorcer une réflexion théorique sur des algorithmes de *clustering* parallèles et asynchrones. Cette étude a finalement abouti à la mise en oeuvre de méthodes que nous avons baptisées de l'acronyme DALVQ (*Distributed Asynchronous Learning Vector Quantization*). Nous avons naturellement commencé par poser les bases mathématiques de ces algorithmes, tout en poursuivant leur analyse, notamment par l'étude de leur convergence presque sûre. Ces travaux théoriques sont regroupés dans le Chapitre 3 de la thèse.

Par la suite nous avons voulu étudier et tester le comportement de nos algorithmes DALVQ sur les architectures pour lesquels ils avaient été pensés à l'origine, à savoir les plate-formes de type *Cloud Computing*. Ces expériences ont donné naissance à un projet logiciel, `CloudDALVQ`[4], distribué sous licence libre `new BSD`. Il a été développé en collaboration avec Matthieu Durut, doctorant au sein de la même société. `CloudDALVQ` est conçu pour être déployé sur `Microsoft Windows Azure` et est écrit principalement dans le langage informatique `C#/.NET`. Sa description ainsi que ses résultats expérimentaux font l'objet du Chapitre 5.

Lors de la réalisation du project `CloudDALVQ` nous avons beaucoup travaillé sur certains aspects pratiques non abordés dans la partie théorique décrite dans le Chapitre 3. Ces questions étant indépendantes des aspects de génie logiciel, nous les discutons dans un Chapitre 4, intermédiaire entre l'analyse mathématique d'une part et le projet logiciel `CloudDALVQ` d'autre part. Dans ce chapitre, nous introduisons également les générateurs de données artificielles utilisés dans la plupart de nos expériences sur les algorithmes DALVQ.

Dans la suite de cette introduction, nous présentons brièvement le contenu de chaque chapitre du manuscrit.

1.2 Présentation des travaux

1.2.1 Chapitre 2 - Prévisions non paramétriques des quantiles de séries temporelles

Ces dernières années ont témoigné d'un regain d'intérêt pour le domaine de la prévision des valeurs futures d'une série temporelle. Les champs d'applications sont

formatique de l'École Normale Supérieure de la rue d'Ulm sous la direction de Joannès Vermorel. Voir `http://cloudster.sourceforge.net/`.

4. `http://code.google.com/p/clouddalvq/`

vastes dans la mesure où la prévision des séries temporelles s'applique à de nombreux domaines de la vie courante et du monde industriel comme, par exemple, la génétique, les diagnostics médicaux, la prévision des pics de pollution etc., mais aussi, comme dans le cas de Lokad, à la prévision des ventes, à l'optimisation des stocks ou encore à la gestion du personnel dans les centres d'appels.

Afin de définir le context mathématique de notre problème, supposons qu'à chaque instant $n \geq 1$, le prévisionniste (ou prédicteur) est interrogé sur la valeur future y_n de la suite de nombres réels $y_1, y_2, \ldots$ Notons y_1^{n-1} (resp. y_1^0) le n-uplet $(y_1, \ldots, y_n)$ (resp. la suite vide). Formellement, la stratégie du prédicteur peut être définie mathématiquement par une suite $g = \{g_n\}_{n=1}^{\infty}$ de fonctions de prévision de la forme

$$g_n : \mathbb{R}^{n-1} \longrightarrow \mathbb{R}, \tag{1.1}$$

et la prévision au temps n est alors simplement donnée par $g_n(y_1^{n-1})$. Dans tout ce chapitre, nous supposons que $y_1, y_2, \ldots$ sont des réalisations de variables aléatoires $Y_1, Y_2, \ldots$ et que le processus stochastique associé $\{Y_n\}_{-\infty}^{\infty}$ est stationnaire et ergodique.

La plupart des techniques statistiques utilisées pour la prévision de séries temporelles s'apparentent à de la régression basée sur la populaire méthode des moindres carrés. Ces types de prévision reposent sur la recherche d'une fonction g_n telle que la prévision $g_n(Y_1^{n-1})$ soit une estimation (ou une quantité voisine) de l'espérance de Y_n, conditionnellement au passé Y_1^{n-1}. De très nombreuses techniques ont été développées sur ce schéma, allant des approches paramétriques telles que les processus AR(p) et ARMA(p,q) (voir par exemple Brockwell et Davies [26]) à des méthodes récentes non paramétriques (voir par exemple Györfi et al. [59] et Bosq [18] pour un tour d'horizon et une liste de références à ce sujet). Les estimations de ces espérances conditionnelles sont certes pertinentes. Néanmoins, il existe également de nombreuses situations où le prévisionniste est plus intéressé par l'estimation de quantiles conditionnels et d'intervalles de confiance. Ces dernières quantités permettent, en effet, de mieux connaître certains aspects de la loi du futur du phénomène conditionnellement à son passé. Porté notamment par les nombreuses applications en finance, la littérature traitant de la régression quantile est en plein essor. L'ouvrage de Koenker [64] constitue une référence dans ce domaine. En outre, la prévision d'une série temporelle peut aussi bien reposer sur l'estimation d'une médiane conditionnelle. Les méthodes de prévision basées sur la médiane conditionnelle jouissent (entres autres avantages) d'excellentes propriétés de robustesse, comme expliqué par Hall et al. dans [61]. Elles rejoignent ainsi les questions de prévision de quantiles, puisque la médiane n'est autre que le quantile d'ordre 1/2.

Ainsi motivés par de nombreuses applications, nous étudions dans ce Chapitre 2 le problème de la prévision des quantiles d'une série temporelle à valeurs réelles. Notre approche est non paramétrique par essence et se distingue des approches plus traditionnelles par trois aspects. Premièrement, nous nous affranchissons des hypothèses habituelles telles que le caractère borné, auto-régressif ou Markovien du processus $\{Y_n\}_{-\infty}^{\infty}$. Notre objectif est, en effet, de montrer des résultats de convergence en supposant seulement que le processus est stationnaire et ergodique. Deuxièmement, le modèle construit dans ce chapitre se base sur des méthodes développées ces dernières années pour la prévision de suites individuelles. Nous présentons un modèle séquentiel de prévision de quantiles basé sur une combinaison de prédicteurs élémentaires de type « plus proches voisins » appelés « experts ». Le paradigme de prévision par agrégat d'experts a été introduit au début des années 90 en théorie du *Machine Learning* comme faisant partie des problèmes d'apprentissage en ligne. Il a été intensément étudié depuis et nous recommandons l'ouvrage de Cesa-Bianchi et Lugosi [30] pour une introduction complète à cette problématique. Enfin, par opposition aux approches non paramétriques plus standards, nous étudions ce problème en exploitant intensivement la structure quantile vue comme un minimiseur de la fonction de perte *pinball* (voir Koenker et Basset [65]). Comme cette remarque est coeur de notre approche, nous l'explicitons davantage dans les lignes qui suivent.

Le quantile d'ordre $\tau \in (0,1)$ d'une variable aléatoire réelle Y est communément définit de la façon suivante :

$$q_\tau = \inf\{t \in \mathbb{R} : F_Y(t) \geq \tau\},$$

où la fonction de répartition F_Y est définie, pour tout $t \in \mathbb{R}$, par

$$F_Y(t) = \mathbb{P}\left[Y \leq t\right].$$

Cependant, sous certaines hypothèses portant sur la régularité de la loi de Y, le quantile d'ordre τ est solution (parfois unique) du problème de minimization suivant :

$$q_\tau \in \operatorname*{argmin}_{q \in \mathbb{R}} \mathbb{E}\left[\rho_\tau(Y-q)\right],$$

dans lequel ρ_τ désigne la fonction *pinball*, linéaire par morceaux et définie comme le montre la Figure 1.1.

Dans le contexte qui est le nôtre, l'objectif est de prévoir la valeur future, au temps n du quantile conditionnel de la variable Y_n connaissant le passé Y_1^{n-1} :

$$q_\tau(Y_1^{n-1}) = \inf\{t \in \mathbb{R} : F_{Y_n|Y_1^{n-1}}(t) \geq \tau\}.$$

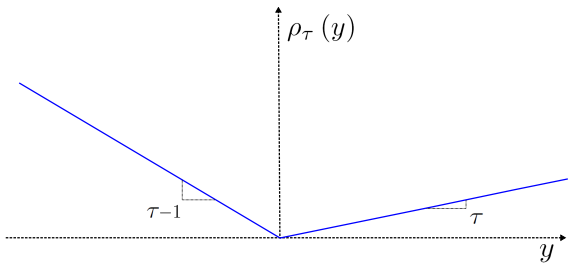

FIGURE 1.1: La fonction de perte *pinball* ρ_τ.

Il est ainsi naturel de mesurer les performances d'une stratégie de prévision quantile $g = \{g_n\}_{n=1}^{\infty}$, où les fonctions g_n sont du type de celle de la formule (1.1), à l'aide d'un critère cumulatif faisant intervenir la fonction de perte *pinball*. Ce critère s'écrit donc

$$L_n(g) = \frac{1}{n}\sum_{t=1}^{n} \rho_\tau\left(Y_t - g_t(Y_1^{t-1})\right).$$

Par conséquent, une stratégie sera d'autant plus précise et satisfaisante que le critère d'erreur ci-dessus sera petit.

La stratégie de prévision quantile introduite dans ce chapitre repose sur une agrégation d'experts. Sans rentrer dans les détails, chacun de ces prédicteurs fondamentaux est fondé sur la technique des plus proches voisins. Les prédictions sont agrégés avec des poids dépendant directement de leurs performances passées. Plus précisément, le poids d'un expert dans l'agrégat évolue au fur et à mesure du temps et il est d'autant plus important que les prévisions passées de l'expert considéré ont été satisfaisantes.

Notre principal résultat établit que cette stratégie de prévision quantile g est, sous certaines hypothèses très générales, universellement convergente. Cela signifie que pour tout processus stationnaire et ergodique on a la convergence suivante :

$$\lim_{n\to\infty} L_n(g) = L^\star, \quad \text{presque sûrement,}$$

où $L^\star$ est la plus petite perte asymptotique moyenne possible pour une stratégie de prévision quantile.

De cette stratégie de prévision quantile décrite mathématiquement, nous avons extrait et codé un véritable modèle opérationnel intégré dans le moteur de prévision de Lokad. Nous détaillons l'implémentation pratique d'un tel modèle et nous l'expérimentons dans la suite du chapitre sur un jeu de données de type « centre

d'appels » composé de 21 séries temporelles mesurées à la journée (en moyenne 760 jours par série). Nous présentons d'abord les performances de ce nouveau modèle en tant que méthode de prévision, en utilisant une estimation de mettant l'accent sur la médiane, c'est-à-dire $\tau = 0.5$. Ces résultats sont comparés à ceux de certaines techniques classiques déjà implémentées dans le système de prévision de Lokad. Nous montrons que notre nouveau modèle donne d'excellents résultats et ceci suivant divers critères d'erreurs. Nous poursuivons l'étude expérimentale de notre procédure de prévision quantile avec d'autres valeurs de τ, à savoir $\tau = 0.1$ et $\tau = 0.9$. La Figure 1.2 montre un graphique de ce type de prévisions. De plus, une comparaison avec le modèle le plus connu de la régression quantile, le modèle $\texttt{QAR(p)}_\tau$ (voir Koenker [64]) est menée en utilisant deux critères d'erreur pertinents dans ce cadre (dont la perte *pinball*). Une fois de plus, le modèle d'agrégation d'experts fonctionne de manière surprenante en dépassant dans bien des cas le modèle $\texttt{QAR(p)}_\tau$.

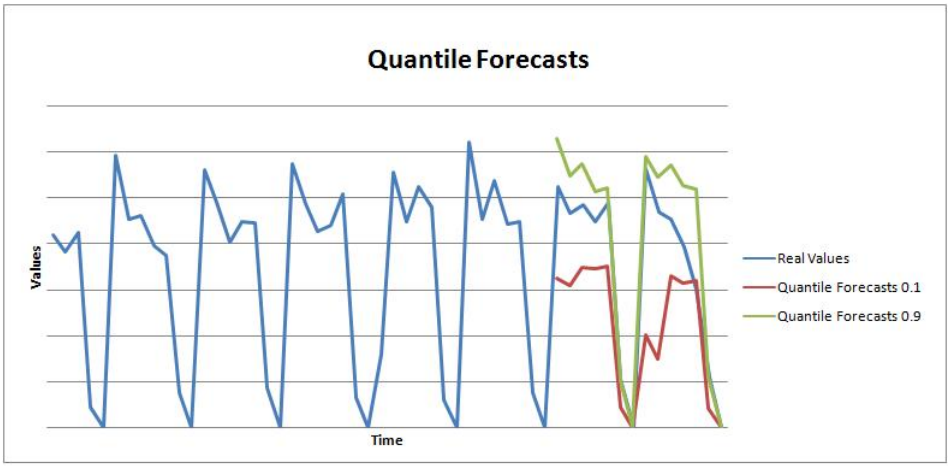

FIGURE 1.2: Prévisions quantiles d'une série temporelle réalisées par la méthode d'agrégation d'experts introduite dans le Chapitre 2.

1.2.2 Chapitre 3 - Etude de la convergence des algorithmes DALVQ

Les algorithmes répartis sont aujourd'hui fréquemment utilisés dans le domaine des télécommunications, du calcul scientifique, de l'informatique décisionnelle et bien d'autres secteurs encore. La parallélisation des unités de calculs apparaît de

nos jours comme la voie la plus prometteuse pour accroître encore davantage les ressources informatiques disponibles pour une tâche donnée. En effet, la construction de « supers processeurs » semble de moins en moins prisée et rencontre également des limites physiques de miniaturisation et de cadence d'exécution. L'un des grands défis adressé au *Machine Learning* consiste désormais à bâtir des applications dites « *scalable* », c'est-à-dire capables de gérer et de traiter intelligemment des quantités de données toujours plus importantes.

Les algorithmes de *clustering* sont parmi les outils les plus importants de l'apprentissage non supervisé. D'un point de vue pratique, ils tiennent une place de choix dans l'analyse de données. C'est le cas notamment en *Text Mining*, en bio-informatique ou encore, par certains aspects, dans l'étude des séries temporelles. Le *clustering* peut être défini, en un mot, comme la séparation (suivant un certain critère de similarité) d'un ensemble de données en sous-groupes. Il permet donc de représenter les données par le groupe qui les contient, ces groupes étant alors appelés *clusters*[5]. Cette simplification a pour conséquence d'effacer certains détails fins. Ainsi, la compréhension et la manipulation des grosses quantités de données est facilitée. La représentation à l'aide des *clusters* peut être également utile en soi. Comme par exemple dans le cas de Lokad, où les séries temporelles montrant un profil de saisonnalité commun sont regroupées pour une meilleure estimation de cette composante. Notons par ailleurs que le *clustering* peut aussi permettre d'accélérer l'exécution d'autres procédures statistiques en réduisant la complexité des calculs.

Le célèbre algorithme de CLVQ (*Competitive Learning Vector Quantization*) permet de calculer des *clusters* pertinents déterminés par des points de référence appelés prototypes ou centroïdes (voir par exemple Gersho et Gray [48]). Bottou et Bengio remarquent dans [21] que l'algorithme de CLVQ peut être vu comme la version « en-ligne » de la méthode de Lloyd, appelée également *batch k-means* (voir Lloyd [73] pour une définition). Par certains aspects, le CLVQ appartient à la grande famille des algorithmes de descente de gradient stochastique. C'est le point de vue que nous adopterons lors de son étude mathématique.

Le clustering peut être considéré comme la version statistique du problème probabiliste de la quantification. Ce problème consiste à proposer une description simplifiée d'une mesure de probabilité inconnue μ, que nous supposons à valeurs dans $\mathbb{R}^d$. Nous cherchons ici à représenter cette mesure à l'aide de $\kappa \geq 1$ vecteurs

5. Traduction du terme français « grappes ». Nous mettons en garde le lecteur que, dans un souci de clarté, nous n'utiliserons à aucun moment le terme *cluster* dans le sens de *computer cluster*, qui désigne fréquemment un groupe d'ordinateurs reliés par un réseau local.

de $\mathbb{R}^d$, les prototypes. Ces derniers définissant les *clusters* et sont donnés par un vecteur $w \in \left(\mathbb{R}^d\right)^\kappa$, dont la pertinence est mesurée par le critère de distortion :

$$C_\mu(w) = \frac{1}{2} \int_{\mathbb{R}^d} \min_{1 \leq \ell \leq \kappa} \|\mathbf{z} - w_\ell\|^2 \, d\mu(\mathbf{z}).$$

La distortion est un fonction continûment différentiable pour certaines valeurs de w. Cela signifie qu'il existe une fonction $H : \mathbb{R}^d \times \left(\mathbb{R}^d\right)^\kappa \longrightarrow \left(\mathbb{R}^d\right)^\kappa$ satisfaisant pour ces valeurs de w et pour tout $\mathbf{z} \in \left(\mathbb{R}^d\right)^\kappa$,

$$\nabla C(w) = \int_{\mathbb{R}^d} H(\mathbf{z}, w) d\mu(\mathbf{z}).$$

Le fait que la fonction de distortion soit non différentiable sur tout l'espace induit des difficultés qui sont discutées dans le chapitre.

Les travaux de cette thèse portent sur les méthodes effectives permettant de calculer de bons prototypes au regard du critère introduit ci-dessus. Dans le problème qui est le nôtre, la mesure μ est inconnue et n'est visible qu'à travers un échantillon représenté par les variables aléatoires $\mathbf{z}_1, \mathbf{z}_2, \ldots$ Cependant, pour une valeur de $w \in \left(\mathbb{R}^d\right)^\kappa$ et une donnée $\mathbf{z} \in \mathbb{R}^d$, une « observation » du gradient $H(\mathbf{z}, w)$ est très simplement accessible par un simple calcul de distance. Ainsi, l'algorithme de CLVQ est naturellement donné par une procédure de type descente de gradient stochastique qui s'écrit

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

où $\{\varepsilon_t\}_{t=1}^\infty$ est une suite de réels positifs, appelée suite des pas. La convergence presque sûre de cet algorithme est étudiée par Pagès dans [82]. Cet auteur obtient, de notre point de vue, les résultats les plus satisfaisants sur la question. Pagès [82] montre en particulier que, sous une hypothèse asymptotique portant sur les trajectoires de l'algorithme, la converge suivante à lieu :

$$\operatorname{dist}(w(t), \Xi_\infty) \xrightarrow[t \to \infty]{} 0, \quad \text{presque sûrement},$$

où Ξ_∞ est une composante connexe de l'ensemble $\{\nabla C = 0\}$ et la notation dist représente la fonction de distance entre un point et un ensemble.

L'analyse des procédures de type descente de gradient stochastique en parallèle dans un context de *Machine Learning* a fait l'objet d'une attention particulière ces dernières années, comme le montrent, par exemple, les travaux de Langford et al. [96], Mac Donald et al. [77], Louppe et Geurts [74] et Zinkevich et al. [98].

Notre objectif dans ce chapitre est de réunir l'algorithme de CLVQ (à travers les méthodes développées dans [82]) et la théorie des algorithmes linéaires parallèles asynchrones. Cette dernière théorie a été développée sous l'influence de Tsitsiklis dans la série de publications Tsitsiklis [93], Tsitsiklis et al. [94], Bertsekas et Tsitsiklis [11]. Nous avons baptisé le modèle resultant de ce rapprochement sous le sigle DALVQ (*Distributed Asynchronous Learning Vector Quantization*). En un mot, ces procédures réalisent sur différentes instances de calculs plusieurs exécutions de l'algorithme de CLVQ. Les résultats intermédiaires de ces techniques sont alors diffusés efficacement et de manière asynchrone à travers le réseau reliant les instances de calculs.

Sans entrer dans les détails, voici le fonctionnement global de ces modèles. Nous supposons que nos ressources informatiques reposent sur une architecture distribuée composée de M instances de calculs. Les algorithmes de DALVQ se définissent par un système d'équations similaire à celui introduit dans [11] reliant les différentes valeurs des prototypes que nous appelons alors les versions, et qui seront notées $\{w^i(t)\}_{t=0}^{\infty}$, où $i \in \{1, \ldots, M\}$. Plus précisément, pour chaque version i, au temps t, $w^i(t)$ est égal à la somme d'une combinaison convexe des autres versions w^j et d'un terme de déplacement correspondant à une itération du type « CLVQ » calculée localement. En outre, l'asynchronisme est modélisé par des délais : la combinaison convexe fait intervenir les autres versions retardées, qui s'écrivent alors $w^j(\tau^{i,j}(t))$, où $0 \leq \tau^{i,j}(t) \leq t$. Ainsi, la quantité $t - \tau^{i,j}(t)$ peut être vue comme la durée nécessaire à l'observation de w^j par l'instance i. Cette modélisation est illustrée par le schéma de la Figure 1.3. Remarquons par ailleurs que, pour la version i, on a $\tau^{i,i}(t) = t$ et que, parfois, la combinaison convexe peut être réduite à une combinaison triviale où seul le poids de la version i vaut 1 les autres étant nuls. Dans ce cas, les différentes instances de calculs exécutent la plupart du temps des itérations locales du CLVQ et ne font pas systématiquement intervenir les autres versions dans leurs calculs.

Le chapitre 3 reprend, dans une partie dédiée, les grandes lignes des travaux de Blondel et al. [17] portant sur les problèmes de consensus des algorithmes linéaires parallèles et asynchrones. Les hypothèses pertinentes sont présentées à cette occasion. Le premier résultat mathématique important sur le DALVQ que nous démontrons dans ce chapitre concerne l'existence d'un consensus asymptotique. Plus précisément, presque sûrement, les différentes versions vont coïncider asymptotiquement. Ce type de résultat constitue une véritable nécessité pour obtenir un algorithme parallèle satisfaisant. Dans notre cas cela s'écrit

$$w^i(t) - w^j(t) \xrightarrow[t\to\infty]{} 0, \quad (i,j) \in \{1, \ldots, M\}^2 \text{ presque sûrement.}$$

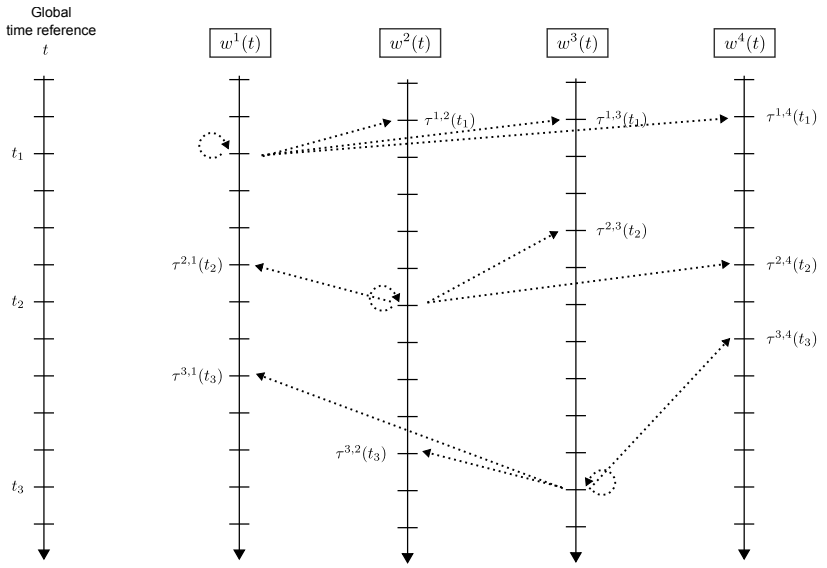

FIGURE 1.3: Ce schéma illustre les délais introduits dans les modèles DALVQ. Dans la situation du dessin, il y a $M = 4$ différentes instances de calculs avec leur propre version w^i, où $i \in \{1, 2, 3, 4\}$. Trois valeurs de temps ont été choisies arbitrairement dans le but de ne pas surcharger la figure : t_1, t_2 et t_3. Sur ce dessin, les flèches partent au temps courant d'une version considérée et pointent vers la version visible d'une autre instance.

Pour terminer nous montrons un résultat central, dans l'esprit de Pagès [82] que, sous certaines hypothèses, les différentes versions s'approchent d'un ensemble de points critiques de la distortion, c'est-à-dire

$$\text{dist}\left(w^i(t), \Xi_\infty\right) \xrightarrow[t\to\infty]{} 0, \quad i \in \{1, \ldots, M\} \text{ presque sûrement,}$$

où Ξ_∞ est une composante connexe de l'ensemble $\{\nabla C = 0\}$.

Nos algorithmes DALVQ sont capables de visiter beaucoup plus de données, au cours d'une certaine durée d'exécution, que la procédure de CLVQ sur une seule machine. Dans de telles circonstances, l'utilisateur s'attend à un *speed up*, c'est-à-dire à un gain de temps comparativement à une réalisation séquentielle. De plus, la définition du DALVQ introduit de l'asynchronisme, ce qui signifie que les diverses exécutions locales n'ont pas besoin d'attendre certains moments spécifiques pour communiquer. Ceci autorise donc des instances de calculs à être plus performantes que d'autres. Le modèle DALVQ supporte des délais de communication pouvant

être importants et imprévisibles. Nous insistons sur le fait que ce modèle est particulièrement adapté à la description d'implémentations sur des plate-formes de type *Cloud Computing* où les communications passent le plus souvent à travers un stockage distant.

L'asynchronisme apporte, par ailleurs, deux avantages supplémentaires. Premièrement, on constate une réduction des temps d'attente et donc un potentiel gain de temps par rapport aux algorithmes comportant certaines phases de synchronisation. Deuxièmement, les implémentations basées sur des comportements asynchrones sont souvent plus tolérantes aux problèmes pratiques, comme les pannes où d'autres phénomènes non déterministes. Notons également que les algorithmes DALVQ sont, de part leur nature « en-ligne », adaptés à des modifications des données pendant leur exécution. Cet aspect, inexistant dans les approches de type *batch*, est très important pour une application en production.

1.2.3 Chapitre 4 - Considérations pratiques pour implémentation des algorithmes DALVQ

Ce chapitre, ainsi que le suivant, présente des travaux réalisés en collaboration avec Matthieu Durut, doctorant au sein de la société Lokad. Nous traitons de certains aspects pratiques des algorithmes de quantification DALVQ introduits dans le chapitre précédent. Rappelons que ces derniers algorithmes sont alors définis par des équations mathématiques générales et peuvent à ce titre couvrir des réalisations effectives très diverses. La plupart des questions soulevées dans les paragraphes à venir ont été abordées pendant l'écriture du projet logiciel `CloudDALVQ`. La réalisation même du projet nous a conduits à mener une réflexion plus théorique sur la mise en place d'algorithmes de DALVQ. Dans ce chapitre, nous étudions les schémas de parallélisation envisageables pour une implémentation efficace du DALVQ. Nous avons donc codé de petits programmes expérimentaux qui simulent des situations de calculs répartis en utilisant des données synthétiques. Ce chapitre se propose donc de jouer un rôle de transition entre l'analyse mathématique des méthodes DALVQ du Chapitre 3 et le véritable projet logiciel `CloudDALVQ` déployé sur `Microsoft Windows Azure` et présenté au Chapitre 5.

Rappelons tout d'abord que nos algorithmes de DALVQ sont basés sur la procédure de CLVQ. Ces algorithmes exécutent en parallèle des méthodes de CLVQ en utilisant des ressources de calculs réparties, qui peuvent donc se situer sur des machines physiques distinctes. Les résultats intermédiaires (les versions) sont diffusées à travers le réseau de manière asynchrone. Dans le Chapitre 3 nous avons expliqué que l'algorithme de CLVQ appartenait à la grande famille des descentes

de gradient stochastique. Nos algorithmes de DALVQ peuvent donc être perçus comme des algorithmes de descente de gradient stochastique asynchrones et répartis. Ces derniers ont été le sujet d'études récentes comme celles de Langford et al. [96] et de Louppe et Geurts [74]. Cependant, dans le context des travaux de ces auteurs, les programmes sont exécutés par des machines disposant d'une mémoire commune à accès rapide (du type RAM). Dans notre situation nous souhaitons, à termes, déployer nos algorithmes sur des plate-formes de *Cloud Computing*, qui ne possèdent pas de telles mémoires partagées. Par ailleurs, des algorithmes de descente de gradient stochastique en parallèle sans mémoire partagée sont proposés par Zinkevich et al. dans [98]. Malheureusement, ces derniers ne sont pas adaptés à la situation de la quantification principalement à cause de la non convexité de la fonction de distortion.

Nous démarrons le chapitre par une présentation des générateurs aléatoires qui nous serviront pour évaluer nos méthodes de parallélisation. En effet, nous nous proposons de générer des données artificielles de grandes dimensions, ce qui fait sens dans notre contexte d'apprentissage à grande échelle. Plus précisément, nous avons cherché à travailler sur de fonctions échantillonnées. Ce context est un cas d'étude fréquent en statistique, se rapprochant non seulement de l'étude des séries temporelles, mais encore, par exemple, de celui des données météorologiques ou spectrométriques. Dans une première section nous présentons les générateurs aléatoires de mélanges de fonctions B-splines que nous avons codés et qui seront utilisés dans ce chapitre ainsi que dans le suivant.

Dans les travaux présentés dans la suite, nous abordons les algorithmes de DALVQ du point de vue pratique. Rappelons que dans la modélisation du DALVQ, chaque instance de calculs a accès dans sa mémoire à un ensemble contenant n données de dimension d à composantes réelles, notées $\{\mathbf{z}_t\}_{t=1}^{n}$ ($d \geq 1$, $n \geq 1$). L'algorithme a pour objectif de trouver une « bonne quantification » pour ce jeu de données, obtenue à travers un calcul de prototypes pertinents. Les itérations $\{w(t)\}_{t=0}^{\infty}$ de la méthode CLVQ, détaillées dans la partie 3.2, s'écrivent avec ce nouveau point de vue de la manière suivante :

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

où $w(0) \in \left(\mathbb{R}^d\right)^{\kappa}$ et $\{\varepsilon_t\}_{t=1}^{\infty}$ est une suite de réels décroissante. Le symbole *mod* désigne le reste dans la division euclidienne d'entiers. Ces itérations font des passes (ou des cycles) sur les données jusqu'à ce qu'un critère d'arrêt soit rencontré.

Les algorithmes de *clustering* ont pour but de résumer les données. Par conséquent, il est naturel d'utiliser un critère de type empirique pour évaluer les pro-

cédures. Remarquons également que l'ensemble des données est séparé de manière égale à travers les M instances de calculs, chacune possédant n points. Cet ensemble est donc décrit par les M suites $\{\mathbf{z}_t^i\}_{t=1}^n$, $i \in \{1, \ldots, M\}$. Le critère de distortion que nous utilisons ici est normalisé, dans le but de pouvoir faire évoluer M, et donc de comparer les résultats du *clustering* sur des jeux de données de tailles différentes. Il s'écrit pour un vecteur de prototypes $w \in \left(\mathbb{R}^d\right)^\kappa$,

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

Remarque : les diverses techniques d'optimisation du CLVQ, comme le choix de la procédure d'initialisation ou celui de la suite des pas $\{\varepsilon_t\}_{t=1}^\infty$, ne sont pas discutés dans ces travaux. En effet, nous supposons disposer d'une implémentation pertinente du CLVQ bien adapté à notre problème. Rappelons que notre objectif principal vise à accélérer cet algorithme à travers une parallélisation efficace du calcul.

Pour commencer notre étude, nous proposons de suivre l'évolution du critère de distortion dans un schéma de parallélisation très simple, donné par le système d'équations ci-dessous. Par hypothèse, les versions initiales $w^i(0)$ sont toutes égales. Le système décrit donc M exécutions parallèles du type CLVQ avec mise en accord. Précisément, dès lors que τ points ont été visités pour chaque instance de calculs, les différentes versions sont toutes affectées à une même valeur, la version partagée w^{srd}. Dans le cas présent w^{srd} est égal à la moyenne des autres versions. Pour tout $t \geq 0$, nous considérons les itérations

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

L'étude de ce schéma de parallélisation synchrone, très naturel, nous apporte beaucoup en termes de compréhension. Dans ce chapitre, nous expliquons que le développement des algorithmes en ligne parallèles basés sur une moyenne des versions ne permet pas d'apporter un *speed up* même si plus de données sont utilisées. Ce phénomène est dû à une modification du paramètre d'apprentissage, caractérisé par la suite $\{\varepsilon_t\}_{t=1}^\infty$, et dont le choix apporte un compromis entre exploration et convergence. Dans le cas présent, le taux d'apprentissage, comparativement à l'exécution séquentielle, est diminué et donc l'exploration est augmentée au détriment de la convergence. Cependant, nous montrons qu'une alternative peut être mise en place, en conservant à la fois les versions dans les mémoires locales des

unités de calculs mais également les termes de déplacements entre deux mises à jour t_1 et t_2 ($t_2 > t_1$). Ces déplacements sont définis comme

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

Nous montrons ensuite, dans une série d'experiences, que des accélérations peuvent être apportées par ces nouveaux schémas de parallélisation. Nous proposons ainsi un modèle basé sur ce principe que nous testons à l'aide d'un petit programme simulant une situation répartie. Enfin, nous terminons le chapitre par une amélioration de ce modèle en introduisant des délais, permettant un calcul de la version partagée réalisé de manière asynchrone .

1.2.4 Chapitre 5 - Le projet `CloudDALVQ`

Dans ce cinquième et dernier chapitre nous présentons le projet logiciel `CloudDALVQ`[6]. Le projet a été conçu pour être déployé sur la plate-forme de *Cloud Computing* `Microsoft Windows Azure`. Il se propose d'être une implémentation des algorithmes de DALVQ présentés théoriquement au Chapitre 3. Le chapitre comporte des descriptions sont d'ordre plutôt théoriques qui résident bien souvent dans des équations mathématiques générales. En effet, les modèles DALVQ ont été pensés dès leur conception pour être déployés sur des plate-formes du type *Cloud Computing*. A titre d'exemple, citons les délais de communications imprévisibles et non négligeables introduits dans la modélisation DALVQ, et qui font partie des aspects essentiels de ce type d'architecture répartie. Le Chapitre 4 a été écrit dans le but d'établir un pont entre les questions mathématiques et les problèmes de génie logiciel que nous évoquons dans ce cinquième chapitre. L'implémentation de `CloudDALVQ` utilise de manière fondamentale les résultats et les remarques apportées dans les travaux du Chapitre 4.

`CloudDALVQ` est distribué sous license libre `new BSD` et est écrit principalement en `C#/.NET 4.0`. Il repose également sur un autre projet libre, `Lokad.Cloud`[7], contenant des fonctionnalités d'accès au système de stockage Azure `BlobStorage` et un *framework* d'exécution simplifié à l'aide d'abstractions facilitant la manipulation de certains aspects bas niveau de Azure.

Tout d'abord, nous proposons une rapide introduction au concept de *Cloud Computing* en mettant en valeur un aspect fondamental : l'élasticité des ressources

6. `http://code.google.com/p/clouddalvq/`
7. `http://lokad.github.com/lokad-cloud/`

informatiques. Nous décrivons les différents types de solutions principalement proposées par les acteurs du marché (en tout cas, au moment de l'écriture de cette thèse). La solution `Windows Azure` fournie par Microsoft fait partie d'une offre dite de PaaS (*Plateform as a Service*). Ce modèle consiste essentiellement à offrir aux clients un environnement *middleware* immédiatement disponible où l'infrastructure est masquée. Dans la suite du chapitre, nous effectuons une présentation de l'environnement logiciel et des abstractions proposé par `Azure` et `Lokad.Cloud`. Nous expliquons en particulier comment le paradigme `Azure/Lokad.Cloud` permet de concevoir des applications réparties à l'aide d'un système de traitement de messages et de queues parallèles. Plus précisément, une application est un ensemble de routines du type `QueueService` qui sont déployées sur divers *workers*, les unités de calcul `Azure`. Ces `QueueServices` sont parfois utilisées à la manière d'un SPMD *Single Program Multiple Data* : la même routine est exécutée avec diverses instructions contenues dans les messages rassemblés dans des queues. Nous présentons ensuite succinctement le `Azure BlobStorage`. Il s'agit d'un système de stockage grande échelle pour données non structurées (les blobs). Ce `BlobStorage` sert en particulier de système de communication entre les workers puisque des connections directes ne sont pas recommandées sur ce type d'architecture.

Les spécificités de l'implémentation de `CloudDALVQ` sont ensuite détaillées dans une section dédiée. Nous débutons par une description du plus important `QueueService`, à savoir le `ProcessService`, déployé sur plusieurs *workers*. Il s'agit du coeur de l'algorithme, dans la mesure où chaque instance réalise alors une exécution du CLVQ. Le `ProcessService` utilise de manière centrale les résultats du Chapitre 4, car la mise en accord est faite au travers de la somme des termes de descentes $\Delta^j_{\cdot\to\cdot}$. Notons par ailleurs que les données sont directement chargées dans les mémoires des machines virtuelles. En effet, dans ce chapitre nous utilisons de nouveau les données synthétiques introduites dans le chapitre précédent, ce qui permet de pouvoir les générer localement sur les *workers*.

La difficulté du code du `ProcessService` provient de l'utilisation intensive du *multi-threading* au sein même de la machine virtuelle dédiée au *worker*. Ceci nous permet d'exploiter au maximum les propriétés asynchrones de notre algorithme : les envois des termes de déplacements et les réceptions de la version partagée se font à l'aide de tâches en parallèles, les *threads*. Ainsi, les opérations d'*upload* et de *download* ne bloquent par le principal *thread* responsable des calculs sur les données et, par conséquent, ne sont pas sources de ralentissement.

Nous présentons également le `ReduceService`, un autre `QueueService` chargé de calculer la version partagée, qui constitue la référence commune pour les diverses

instances du `ProcessService`. Nous terminons cet aperçu de l'implémentation de `CloudDALVQ` en expliquant le système d'évaluation mis en place et indispensable à l'étude de l'algorithme. Dans cette situation, il n'est pas possible d'évaluer la performance du *clustering* en temps réel avec le critère de distortion empirique. Ceci aurait pour conséquence de ralentir l'algorithme et donc de fausser ses performances. Pour contourner ces difficultés, nous mettons en place un système reposants sur des *snapshots*[8]. Pour mettre en place ce procédé, un nouveau *worker* est déployé hébergeant un `QueueService` chargé de réaliser des copies régulières de la version partagée. Ces dernières seront par la suite évaluées, une fois l'algorithme terminé, en générant à nouveau les mêmes données artificielles et en utilisant les *workers* libérés (et d'autres si besoin). Notons au passage qu'un tel processus d'évaluation serait très compliqué à envisager sans données synthétiques.

Nous présentons deux types d'expériences réalisées sur `CloudDALVQ`. Nous commençons par présenter des résultats d'expériences de type *scale up*, similaires à celles menées dans le Chapitre 4. Ces expériences ont donc pour but de montrer que l'augmentation de ressources de calcul et des données en parallèle permet d'accélérer la convergence de l'algorithme tout en gardant de bons niveaux de quantification. Nous montrons que `CloudDALVQ` possède de bonne propriétés jusqu'à 8 instances de `ProcessService` ; ensuite le `ReduceService` se retrouve surchargé et l'algorithme est paralysé. Nous réussissons à contourner cette difficulté en répartissant la tâche réalisée jusqu'ici par la seule instance du `ReduceService`. Nous parvenons alors à continuer d'apporter des gains de vitesses significatifs jusqu'à 32 instances. Nous présentons également les performances de l'algorithme en nombres de données visitées par unité de temps. Ceci ne permet pas d'évaluer les qualités statistiques de l'implémentation, mais étudie ses capacités purement calculatoires. Les résultats sont très satisfaisants, et l'on obtient alors une fonction du nombre de points visités par nombre d'instances de calculs parfaitement linéaire. Ces analyses montrent que la parallélisation de l'algorithme en ligne CLVQ réalisée par `CloudDALVQ` est efficace.

Les précédents résultats ne prouvent pas que `CloudDALVQ` constitue une bonne méthode de quantification/*clustering* par rapport aux autres techniques existantes. Dans ce but nous avons confronté `CloudDALVQ` à une autre implémentation d'algorithme de *clustering* sur `Windows Azure`. Nous comparons les méthodes apportées par `CloudDALVQ` et `CloudBatchKMeans` sur un même jeu de donnés de grande taille et donc réparti dans la mémoire de diverses machines. Cette seconde méthode est le déploiement de la procédure de Lloyd's (ou *batch k-means*) sur `Windows Azure` par Durut et Rossi [42]. Dans les situations de nos expériences, l'algorithme de

8. Copies instantanées.

`CloudDALVQ` surpasse de manière surprenante cette procédure très populaire et réputée pour son efficacité.

2 Sequential quantile prediction of time series

Ce chapitre à fait l'objet d'une co-publication avec le Professeur Gérard Biau [15].

2.1 Introduction

Forecasting the future values of an observed time series is an important problem, which has been an area of considerable activity in recent years. The application scope is vast, as time series prediction applies to many fields, including problems in genetics, medical diagnoses, air pollution forecasting, machine condition monitoring, financial investments, production planning, sales forecasting and stock controls.

To fix the mathematical context, suppose that at each time instant $n = 1, 2, \ldots$, the forecaster (also called the predictor hereafter) is asked to guess the next outcome y_n of a sequence of real numbers $y_1, y_2, \ldots$ with knowledge of the past $y_1^{n-1} = (y_1, \ldots, y_{n-1})$ (where y_1^0 denotes the empty string). Formally, the strategy of the predictor is a sequence $g = \{g_n\}_{n=1}^{\infty}$ of forecasting functions

$$g_n : \mathbb{R}^{n-1} \longrightarrow \mathbb{R}$$

and the prediction formed at time n is just $g_n(y_1^{n-1})$. Throughout the paper we will suppose that y_1, $y_2, \ldots$ are realizations of random variables Y_1, $Y_2, \ldots$ such that the stochastic process $\{Y_n\}_{-\infty}^{\infty}$ is jointly stationary and ergodic.

Many of the statistical techniques used in time series prediction are those of regression analysis, such as classical least square theory, or are adaptations or analogues of them. These forecasting schemes are typically concerned with finding a function g_n such that the prediction $g_n(Y_1^{n-1})$ corresponds to the conditional mean of Y_n given the past sequence Y_1^{n-1}, or closely related quantities. Many methods have been developed for this purpose, ranging from parametric approaches such as AR(p) and ARMA(p,q) processes (Brockwell and Davies [26]) to more involved

nonparametric methods (see for example Györfi et al. [59] and Bosq [18] for a review and references).

On the other hand, while these estimates of the conditional mean serve their purpose, there exists a large area of problems where the forecaster is more interested in estimating conditional quantiles and prediction intervals, in order to know other features of the conditional distribution. There is now a fast pace growing literature on quantile regression (see Gannoun et al. [47] for an overview and references) and considerable practical experience with forecasting methods based on this theory. Economics makes a persuasive case for the value of going beyond models for the conditional mean (Koenker and Allock [66]). In financial mathematics and financial risk management, quantile regression is intimately linked to the τ-Value at Risk (VaR), which is defined as the $(1-\tau)$-quantile of the portfolio. For example, if a portfolio of stocks has a one-day 5%-VaR of €1 million, there is a 5% probability that the portfolio will fall in value by more than €1 million over a one day period (Duffie and Pan [39]). More generally, quantile regression methods have been deployed in social sciences, ecology, medicine and manufacturing process management. For a description, practical guide and extensive list of references on these methods and related methodologies, we refer the reader to the monograph of Koenker [64].

Motivated by this broad range of potential applications, we address in this paper the quantile prediction problem of real-valued time series. Our approach is nonparametric in spirit and breaks with at least three aspects of more traditional procedures. First, we do not require the series to necessarily satisfy the classical statistical assumptions for bounded, autoregressive or Markovian processes. Indeed, our goal is to show powerful consistency results under a strict minimum of conditions. Secondly, building on the methodology developed in recent years for prediction of individual sequences, we present a sequential quantile forecasting model based on the combination of a set of elementary nearest neighbor-type predictors called "experts". The paradigm of prediction with expert advice was first introduced in the theory of machine learning as a model of online learning in the 1980-early 1990s, and it has been extensively investigated ever since (see the monograph of Cesa-Bianchi and Lugosi [30] for a comprehensive introduction to the domain). Finally, in opposition to standard nonparametric approaches, we attack the problem by fully exploiting the quantile structure as a minimizer of the so-called pinball loss function (Koenker and Basset [65]).

The document is organized as follows. After some basic recalls in Section 2, we present in Section 3 our expert-based quantile prediction procedure and state its

consistency under a minimum of conditions. We perform an in-depth analysis of real-world datasets and show that the nonparametric strategy we propose is faster and generally outperforms traditional methods in terms of average prediction errors (Section 4). Proofs of the results are postponed to Section 5.

2.2 Consistent quantile prediction

2.2.1 Notation and basic definitions

Let Y be a real-valued random variable with distribution function F_Y, and let $\tau \in (0,1)$. Recall that the generalized inverse of F_Y

$$F_Y^{\leftarrow}(\tau) = \inf\{t \in \mathbb{R} : F_Y(t) \geq \tau\}$$

is called the quantile function of F_Y and that the real number $q_\tau = F_Y^{\leftarrow}(\tau)$ defines the τ-quantile of F_Y (or Y). The basic strategy behind quantile estimation arises from the observation that minimizing the ℓ_1-loss function yields the median. Koenker and Basset [65] generalized this idea and characterized the τ-quantile by tilting the absolute value function in a suitable fashion.

Lemma 2.2.1. *Let Y be an integrable real-valued random variable and, for any $\tau \in (0,1)$, let the map*

$$\rho_\tau(y) = y(\tau - \mathbb{1}_{\{y \leq 0\}}).$$

Then the quantile q_τ satisfies the property

$$q_\tau \in \operatorname*{argmin}_{q \in \mathbb{R}} \mathbb{E}\left[\rho_\tau(Y - q)\right]. \tag{2.1}$$

Moreover, if F_Y is (strictly) increasing, then the minimum is unique, that is

$$\{q_\tau\} = \operatorname*{argmin}_{q \in \mathbb{R}} \mathbb{E}\left[\rho_\tau(Y - q)\right].$$

We have not been able to find a complete proof of this result, and we briefly state it in Section 5. The function ρ_τ, shown in Figure 2.1, is called the pinball function. For example, for $\tau = 1/2$, it yields back the absolute value function and, in this case, Lemma 2.2.1 just expresses the fact that the median $q_{1/2} = F^{\leftarrow}(1/2)$ is a solution of the minimization problem

$$q_{1/2} \in \operatorname*{argmin}_{q \in \mathbb{R}} \mathbb{E}\left[|Y - q|\right].$$

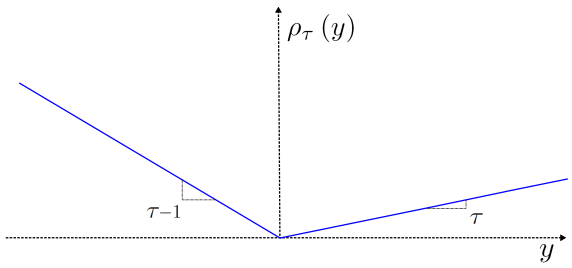

Figure 2.1: Pinball loss function ρ_τ.

These definitions may be readily extended to pairs (X, Y) of random variables with conditional distribution $F_{Y|X}$. In this case, the conditional quantile $q_\tau(X)$ is the measurable function of X almost surely (a.s.) defined by

$$q_\tau(X) = F^{\leftarrow}_{Y|X}(\tau) = \inf\{t \in \mathbb{R} : F_{Y|X}(t) \geq \tau\},$$

and, as in Lemma 2.2.1, it can be shown that for an integrable Y

$$q_\tau(X) \in \underset{q(.)}{\operatorname{argmin}}\, \mathbb{E}_{\mathbb{P}_{Y|X}} \left[\rho_\tau \left(Y - q(X)\right)\right], \tag{2.2}$$

where the infimum is taken over the set of all measurable real-valued functions and the notation $\mathbb{E}_{\mathbb{P}_{Y|X}}$ stands for the conditional expectation of Y with respect to X. We note again that if $F_{Y|X}$ is a.s. increasing, then the solution of (2.2) is unique and equals $q_\tau(X)$ a.s. In the sequel, we will denote by $\mathcal{Q}_\tau(\mathbb{P}_{Y|X})$ the set of solutions of the minimization problem (2.2), so that $q_\tau(X) \in \mathcal{Q}_\tau(\mathbb{P}_{Y|X})$ and $\{q_\tau(X)\} = \mathcal{Q}_\tau(\mathbb{P}_{Y|X})$ when the minimum is unique.

2.2.2 Quantile prediction

In our sequential version of the quantile prediction problem, the forecaster observes one after another the realizations $y_1, y_2, \ldots$ of a stationary and ergodic random process $Y_1, Y_2, \ldots$ At each time $n = 1, 2, \ldots$, before the n-th value of the sequence is revealed, his mission is to guess the value of the conditional quantile

$$q_\tau(Y_1^{n-1}) = F^{\leftarrow}_{Y_n|Y_1^{n-1}}(\tau) = \inf\{t \in \mathbb{R} : F_{Y_n|Y_1^{n-1}}(t) \geq \tau\},$$

on the basis of the previous $n-1$ observations $Y_1^{n-1} = (Y_1, \ldots, Y_{n-1})$ only. Thus, formally, the strategy of the predictor is a sequence $g = \{g_n\}_{n=1}^{\infty}$ of quantile prediction functions

$$g_n : \mathbb{R}^{n-1} \longrightarrow \mathbb{R}$$

and the prediction formed at time n is just $g_n(y_1^{n-1})$. After n time instants, the (normalized) cumulative quantile loss on the string Y_1^n is

$$L_n(g) = \frac{1}{n}\sum_{t=1}^{n} \rho_\tau \left(Y_t - g_t(Y_1^{t-1})\right).$$

Ideally, the goal is to make $L_n(g)$ small. There is, however, a fundamental limit for the quantile predictability, which is determined by a result of Algoet and Cover [5]: for any quantile prediction strategy g and jointly stationary ergodic process $\{Y_n\}_{-\infty}^{\infty}$,

$$\liminf_{n\to\infty} L_n(g) \geq L^\star \quad \text{a.s.}, \tag{2.3}$$

where

$$L^\star = \mathbb{E}\left[\min_{q(.)} \mathbb{E}_{\mathbb{P}_{Y_0|Y_{-\infty}^{-1}}} \left[\rho_\tau \left(Y_0 - q(Y_{-\infty}^{-1})\right)\right]\right]$$

is the expected minimal quantile loss over all quantile estimations of Y_0 based on the infinite past observation sequence $Y_{-\infty}^{-1} = (\ldots, Y_{-2}, Y_{-1})$. Generally, we cannot hope to design a strategy whose prediction error exactly achieves the lower bound $L^\star$. Rather, we require that $L_n(g)$ gets arbitrarily close to $L^\star$ as n grows. This gives sense to the following definition:

Definition 2.2.1. *A quantile prediction strategy g is called consistent with respect to a class $\mathcal{C}$ of stationary and ergodic processes $\{Y_n\}_{-\infty}^{\infty}$ if, for each process in the class,*

$$\lim_{n\to\infty} L_n(g) = L^\star \quad \textit{a.s.}$$

Thus, consistent strategies asymptotically achieve the best possible loss for all processes in the class. In the context of prediction with squared loss, Györfi and Lugosi [56], Nobel [81], Györfi and Ottucsák [54] and Biau et al. [13] study various sequential prediction strategies, and state their consistency under a minimum of assumptions on the collection $\mathcal{C}$ of stationary and ergodic processes. Roughly speaking, these methods consider several "simple" nonparametric estimates (called experts in this context) and combine them at time n according to their past performance. For this, a probability distribution on the set of experts is generated, where a "good" expert has relatively large weight, and the average of all experts' predictions is taken with respect to this distribution. Interestingly, related schemes have been proposed in the context of sequential investment strategies for financial markets. Sequential investment strategies are allowed to use information about the market collected from the past and determine at the beginning of a training period a portfolio, that is, a way to distribute the current capital among the available assets. Here, the goal of the investor is to maximize his wealth in the long run, without knowing the underlying distribution generating the stock prices. For

more information on this subject, we refer the reader to Algoet [4], Györfi and Schäfer [57], Györfi et al. [55], and Györfi et al. [60].

Our purpose in this paper will be to investigate an expert-oriented strategy for quantile forecasting. With this aim in mind, we define in the next section a quantile prediction strategy, called nearest neighbor-based strategy, and state its consistency with respect to a large class of stationary and ergodic processes.

2.3 A nearest neighbor-based strategy

The quantile prediction strategy is defined at each time instant as a convex combination of elementary predictors (the so-called experts), where the weighting coefficients depend on the past performance of each elementary predictor. To be more precise, we first define an infinite array of experts $h_n^{(k,\ell)}$, where k and ℓ are positive integers. The integer k is the length of the past observation vectors being scanned by the elementary expert and, for each ℓ, choose $p_\ell \in (0,1)$ such that

$$\lim_{\ell\to\infty} p_\ell = 0\,,$$

and set

$$\bar{\ell} = \lfloor p_\ell n \rfloor$$

(where $\lfloor . \rfloor$ is the floor function). At time n, for fixed k and ℓ ($n > k + \bar{\ell} + 1$), the expert searches for the $\bar{\ell}$ nearest neighbors (NN) of the last seen observation y_{n-k}^{n-1} in the past and predicts the quantile accordingly. More precisely, let

$$J_n^{(k,\ell)} = \Big\{ k < t < n : y_{t-k}^{t-1} \text{ is among the } \bar{\ell}\text{-NN of } y_{n-k}^{n-1} \text{ in } y_1^k, \ldots, y_{n-k-1}^{n-2} \Big\},$$

and define the elementary predictor $\bar{h}_n^{(k,\ell)}$ by

$$\bar{h}_n^{(k,\ell)} \in \operatorname*{argmin}_{q\in\mathbb{R}} \sum_{t\in J_n^{(k,\ell)}} \rho_\tau(y_t - q)$$

if $n > k + \bar{\ell} + 1$, and 0 otherwise. Next, let the truncation function

$$T_a(z) = \begin{cases} a & \text{if } z > a; \\ z & \text{if } |z| \le a; \\ -a & \text{if } z < -a, \end{cases}$$

and let

$$h_n^{(k,\ell)} = T_{\min(n^\delta,\ell)} \circ \bar{h}_n^{(k,\ell)}, \tag{2.4}$$

where δ is a positive parameter to be fixed later on. We note that the expert $h_n^{(k,\ell)}$ can be interpreted as a (truncated) $\bar{\ell}$-nearest neighbor regression function estimate

drawn in $\mathbb{R}^k$ (Györfi et al. [53]). The proposed quantile prediction algorithm proceeds with an exponential weighting average of the experts. More formally, let $\{b_{k,\ell}\}$ be a probability distribution on the set of all pairs (k,ℓ) of positive integers such that for all k and ℓ, $b_{k,\ell} > 0$. Fix a learning parameter $\eta_n > 0$, and define the weights

$$w_{k,\ell,n} = b_{k,\ell} e^{-\eta_n (n-1) L_{n-1}(h_n^{(k,\ell)})}$$

and their normalized values

$$p_{k,\ell,n} = \frac{w_{k,\ell,n}}{\sum_{i,j=1}^{\infty} w_{i,j,n}}.$$

The quantile prediction strategy g at time n is defined by

$$g_n(y_1^{n-1}) = \sum_{k,\ell=1}^{\infty} p_{k,\ell,n} h_n^{(k,\ell)}(y_1^{n-1}), \qquad n = 1, 2, \ldots \tag{2.5}$$

The idea of combining a collection of concurrent estimates was originally developed in a non-stochastic context for online sequential prediction from deterministic sequences (Cesa-Bianchi and Lugosi [30]). Following the terminology of the prediction literature, the combination of different procedures is sometimes termed aggregation in the stochastic context. The overall goal is always the same: use aggregation to improve prediction. For a recent review and an updated list of references, see Bunea and Nobel [28].

In order to state consistency of the method, we shall impose the following set of assumptions:

($H1$) One has $\mathbb{E}[Y_0^2] < \infty$.

($H2$) For any vector $\mathbf{s} \in \mathbb{R}^k$, the random variable $\|Y_1^k - \mathbf{s}\|$ has a continuous distribution function.

($H3$) The conditional distribution function $F_{Y_0|Y_{-\infty}^{-1}}$ is a.s. increasing.

Condition ($H2$) expresses the fact that ties occur with probability zero. A discussion on how to deal with ties that may appear in some cases can be found in Györfi [58], in the related context of portfolio selection strategies. Condition ($H3$) is mainly technical and ensures that the minimization problem (2.2) has a unique solution or, put differently, that the set $\mathcal{Q}_\tau(\mathbb{P}_{Y_0|Y_{-\infty}^{-1}})$ reduces to the singleton $\{F^{\leftarrow}_{Y_0|Y_{-\infty}^{-1}}(\tau)\}$.

We are now in a position to state the main result of the paper.

Theorem 2.3.1. *Let $\mathcal{C}$ be the class of all jointly stationary ergodic processes $\{Y_n\}_{-\infty}^{\infty}$ satisfying conditions $(H1)$-$(H3)$. Suppose in addition that $n\eta_n \longrightarrow \infty$ and $n^{2\delta}\eta_n \longrightarrow 0$ as $n \to \infty$. Then the nearest neighbor quantile prediction strategy defined above is consistent with respect to $\mathcal{C}$.*

The truncation index T in definition (2.4) of the elementary expert $h_n^{(k,\ell)}$ is merely a technical choice that avoids having to assume that $|Y_0|$ is a.s. bounded. On the practical side, it has little influence on results for relatively short time series. On the other hand, the choice of the learning parameter η_n as $1/\sqrt{n}$ ensures consistency of the method for $0 < \delta < \frac{1}{4}$.

2.4 Experimental results

2.4.1 Algorithmic settings

In this section, we evaluate the behavior of the nearest neighbor quantile prediction strategy on real-world datasets and compare its performances to those of standard families of methods on the same datasets.

Before testing the different procedures, some precisions on the computational aspects of the presented method are in order. We first note that infinite sums make formula (2.5) impracticable. Thus, for practical reasons, we chose a finite grid $(k, \ell) \in \mathcal{K} \times \mathcal{L}$ of experts (positive integers), let

$$g_n(y_1^{n-1}) = \sum_{k\in\mathcal{K},\ell\in\mathcal{L}} p_{k,\ell,n} h_n^{(k,\ell)}(y_1^{n-1}), \qquad n = 1, 2, \ldots \tag{2.6}$$

and fixed the probability distribution $\{q_{k,\ell}\}$ as the uniform distribution over the $|\mathcal{K}| \times |\mathcal{L}|$ experts. Observing that $h_n^{(k,\ell_1)} = h_n^{(k,\ell_2)}$ and $b_{k,\ell_1} = b_{k,\ell_2}$ whenever $\bar{\ell}_1 = \bar{\ell}_2$, formula (2.6) may be more conveniently rewritten as

$$g_n(y_1^{n-1}) = \sum_{k\in\mathcal{K},\bar{\ell}\in\bar{\mathcal{L}}} p_{k,\bar{\ell},n} h_n^{(k,\bar{\ell})}(y_1^{n-1}),$$

where $\bar{\mathcal{L}} = \{\bar{\ell} : \ell \in \mathcal{L}\}$. In all subsequent numerical experiments, we chose $\mathcal{K} = \{1, 2, 3, \ldots, 14\}$ and $\bar{\mathcal{L}} = \{1, 2, 3, \ldots, 25\}$.

Next, as indicated by the theoretical results, we fixed $\eta_n = \sqrt{1/n}$. For a thorough discussion on the best practical choice of η_n, we refer to [13]. To avoid numerical instability problems while computing the $p_{k,\bar{\ell},n}$, we applied if necessary a simple linear transformation on all $L_n(h_n^{(k,\bar{\ell})})$, just to force these quantities to belong to

an interval where the effective computation of $x \mapsto \exp(-x)$ is numerically stable.

Finally, in order to deal with the computation of the elementary experts (2.4), we denote by $\lceil . \rceil$ the ceiling function and observe that if $m \times \tau$ is not an integer, then the solution of the minimization problem

$$\operatorname*{argmin}_{b \in \mathbb{R}} \sum_{i=1}^{m} \rho_\tau (y_i - b)$$

is unique and equals the $\lceil m \times \tau \rceil$-th element in the sorted sample list. On the other hand, if $m \times \tau$ is an integer then the minimum is not unique, but the $m \times \tau$-th element in the sorted sequence may be chosen as a minimizer. Thus, practically speaking, each elementary expert is computed by sorting the sample. The complexity of this operation is $O(\bar{\ell} \log(\bar{\ell}))$—it is almost linear and feasible even for large values of $\bar{\ell}$. For a more involved discussion, we refer the reader to Koenker [64].

All algorithms have been implemented using the oriented object language C# 3.0 and .NET Framework 3.5.

2.4.2 Datasets and results

We investigated 21 real-world time series representing daily call volumes entering call centers. Optimizing the staff level is one of the most difficult and important tasks for a call center manager. Indeed, if the staff is overdimensioned, then most of the employees will be inactive. On the other hand, underestimating the staff may lead to long waiting phone queues of customers. Thus, in order to know the right staff level, the manager needs to forecast the call volume series and, to get a more accurate staff level planning, he has to forecast the quantiles of the series.

In our dataset the series had on average 760 points, ranging from 383 for the shortest to 826 for the longest. Four typical series are shown in Figure 2.2.

We used a set $\mathcal{D}$ of selected dates $m < n$ and, for each method and each time series $(y_1, \ldots, y_n)$ (here, $n = 760$ on average), we trained the models on the pruned series $(y_1, \ldots, y_m)$ and predicted the τ-quantile at time $m + 1$. The set $\mathcal{D}$ is composed of 91 dates, so that all quality criteria used to measure the proximity between the predicted quantiles and the observed values y_{m+1} were computed using $91 \times 21 = 1911$ points. The 21 times series and the set $\mathcal{D}$ are available at the address `http://www.lsta.upmc.fr/doct/patra/` .

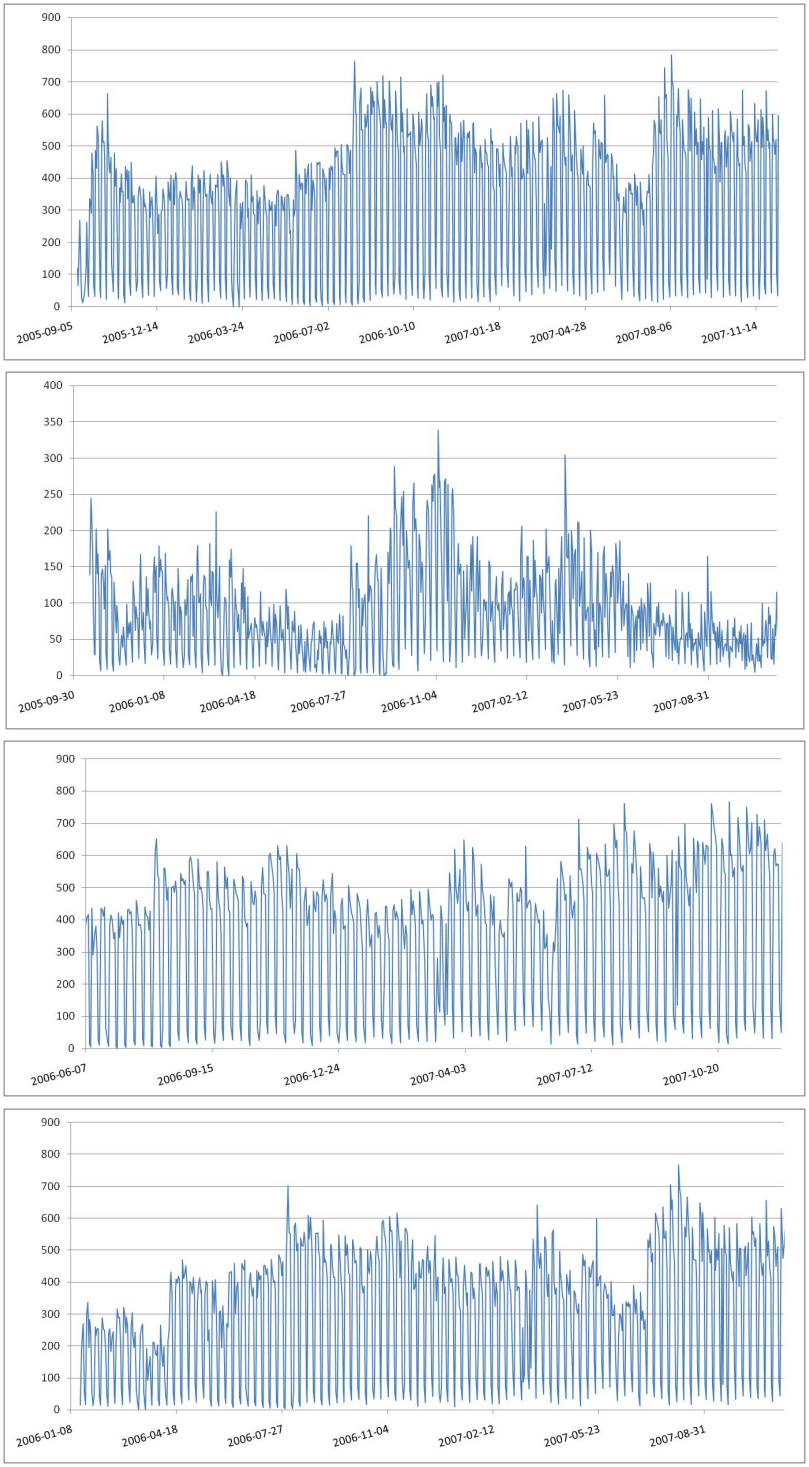

Figure 2.2: Four call center series, out of 21.

In a first series of experiments, we let the methods predict the τ-quantiles at the 1911 dates for $\tau \in \{0.1, 0.5, 0.9\}$. We compared the performances of our expert-based strategy, denoted hereafter by $\texttt{QuantileExpertMixture}_\tau$, with those of $\texttt{QAR(p)}_\tau$, a τ-quantile linear autoregressive model of order p. This quantile prediction model, which is described in [64], also uses the pinball criterion to fit its parameters. The implementation we used solves the minimization problem with an Iterative Re-weighted Least Square algorithm (IRLS), see for instance Street et al. [91]. Following Takeuchi, Le, Sears and Smola [92], we used two criteria to measure the quality of the overall set of quantile forecastings. First, we evaluated the expected risk with respect to the pinball function ρ_τ, referred to as PinBall Loss in the sequel. Secondly we calculated Ramp Loss, the empirical fraction of quantile estimates which exceed the observed values y_{m+1}. Ideally, the value of Ramp Loss should be close to $1 - \tau$.

Tables 2.1-2.3 show the $\texttt{QuantileExpertMixture}_\tau$ and $\texttt{QAR(p)}_\tau$ results at the selected dates $\mathcal{D}$ of the call center series. The latter algorithm was benchmarked for each order p in $\{1, \ldots, 10\}$, but we reported only the most accurate order $p = 7$. The best results with respect to each criterion are shown in bold. We see that both methods perform roughly similarly, with eventually a slight advantage for the autoregressive strategy for $\tau = 0.1$ whereas $\texttt{QuantileExpertMixture}_\tau$ does better for $\tau = 0.9$.

Method	PinBall Loss (0.1)	Ramp Loss
$\texttt{QuantileExpertMixture}_{0.1}$	13.71	0.80
$\texttt{QAR(7)}_{0.1}$	**13.22**	**0.88**

Table 2.1: Quantile forecastings with $\tau = 0.1$.

Method	PinBall Loss (0.5)	Ramp Loss
$\texttt{QuantileExpertMixture}_{0.5}$	**24.05**	0.42
$\texttt{QAR(7)}_{0.5}$	29.157	**0.47**

Table 2.2: Quantile forecastings with $\tau = 0.5$.

Method	PinBall Loss (0.9)	Ramp Loss
$\texttt{QuantileExpertMixture}_{0.9}$	**12.27**	**0.07**
$\texttt{QAR(7)}_{0.9}$	19.31	**0.07**

Table 2.3: Quantile forecastings with $\tau = 0.9$.

Median-based predictors are well known for their robustness while predicting individual values for time series, see for instance Hall, Peng and Yao [61]. Therefore, in a second series of experiments, we fixed $\tau = 0.5$ and focused on the problem of predicting future outcomes of the series. We decided to compare the results of $\texttt{QuantileExpertMixture}_{0.5}$ with those of 6 concurrent predictive procedures:

- `MA` denotes the simple moving average model.
- `AR(p)` is a linear autoregressive model of order p, with parameters computed with respect to the usual least square criterion.
- `QAR(p)` is the τ-quantile linear autoregressive model of order p described earlier.
- `DayOfTheWeekMA` is a naive model, which applies moving averages on the days of the week, that is a moving average on the Sundays, Mondays, and so on.
- `MeanExpertMixture` is an online prediction algorithm described in [13]. It is based on conditional mean estimation and close in spirit to the strategy $\texttt{QuantileExpertMixture}_{0.5}$.
- And finally, we let `HoltWinters` be the well-known procedure which performs exponential smoothing on three components of the series, namely Level, Trend and Seasonality. For a thorough presentation of `HoltWinters` techniques we refer the reader to Madrikakis et al. [76].

Accuracy of all forecasting methods were measured using the Average Absolute Error (Avg Abs Error, which is proportional to the pinball error since $\tau = 0.5$), Average Squared Error (Avg Sqr Error), and the unstable but widely spread criterion Mean Average Percentage Error (MAPE, see [76] for definition and discussion). We also reported the figure Abs Std Dev which corresponds to the empirical standard deviation of the differences $|y_t^F - y_t^R|$, where y_t^F stands for the forecasted value while y_t^R stands for the observed value of the time series at time t. `AR(p)` and `QAR(p)` algorithms were run for each order p in $\{1, \ldots, 10\}$, but we reported only the most accurate orders.

Method	Avg Abs Error	Avg Sqr Error	MAPE (%)	Abs Std Dev
`MA`	179.0	62448	52.0	174.8
`AR(7)`	65.8	9738	31.6	73.5
$\texttt{QAR(8)}_{0.5}$	57.8	9594	24.9	79.2
`DayOfTheWeekMA`	54.1	7183	22.8	64.7
$\texttt{QuantileExpertMixture}_{0.5}$	**48.1**	**5731**	21.6	**58.4**
`MeanExpertMixture`	52.4	6536	22.3	61.6
`HoltWinters`	49.8	6025	**21.5**	59.5

Table 2.4: Future outcomes forecastings.

We see via Table 2.4 that the nearest neighbor strategy presented here outperforms all other methods in terms of Average Absolute Error. Interestingly, this forecasting procedure also provides the best results with respect to the Aver-

age Squared Error criterion. This is remarkable, since $\texttt{QuantileExpertMixture}_{0.5}$ does not rely on a squared error criterion, contrary to `MeanExpertMixture`. The same comment applies to $\texttt{QAR(8)}_{0.5}$ and `AR(7)`. In terms of the Mean Average Percentage Error, the present method and `HoltWinters` procedure provide good and broadly similar results.

2.5 Proofs

2.5.1 Proof of Theorem 2.3.1

The following lemmas will be essential in the proof of Theorem 2.3.1. The first one is known as Breiman's generalized ergodic theorem (Breiman [25]).

Lemma 2.5.1 (Breiman [25]). *Let $Z = \{Z_n\}_{-\infty}^{\infty}$ be a stationary and ergodic process. For each positive integer t, let T^t denote the left shift operator, shifting any sequence of real numbers $\{\ldots, z_{-1}, z_0, z_1, \ldots\}$ by t digits to the left. Let $\{f_t\}_{t=1}^{\infty}$ be a sequence of real-valued functions such that $\lim_{t\to\infty} f_t(Z) = f(Z)$ a.s. for some function f. Suppose that $\mathbb{E}[\sup_t |f_t(Z)|] < \infty$. Then*

$$\lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} f_t(T^t Z) = \mathbb{E}\left[f(Z)\right] \quad a.s.$$

Lemma 2.5.2 below is due to Györfi and Ottucsák [54]. These authors proved the inequality for any cumulative normalized loss of form $L_n(h) = \frac{1}{n}\sum_{t=1}^{n} \ell_t(h)$, where $\ell_t(h) = \ell_t(h_t, Y_t)$ is convex in its first argument, what is the case for the function $\ell_t(h_t, Y_t) = \rho_\tau(Y_t - h_t(Y_1^{t-1}))$.

Lemma 2.5.2 (Györfi and Ottucsák [54]). *Let $g = \{g_n\}_{n=1}^{\infty}$ be the nearest neighbor quantile prediction strategy defined in (2.5). Then, for every $n \geq 1$, a.s.,*

$$L_n(g) \leq \inf_{k,\ell} \left(L_n(h_n^{(k,\ell)}) - \frac{2\ln b_{k,\ell}}{n\eta_{n+1}} \right) + \frac{1}{2n} \sum_{t=1}^{n} \eta_t \sum_{k,\ell=1}^{\infty} p_{k,\ell,n} \Big[\rho_\tau \left(Y_t - h_t^{(k,\ell)}(Y_1^{t-1}) \right) \Big]^2.$$

Lemma 2.5.3. *Let $x, y \in \mathbb{R}$ and $\ell \in \mathbb{N}$. Then*

1. $\rho_\tau(x) \leq |x|$.
2. $\rho_\tau(x + y) \leq \rho_\tau(x) + \rho_\tau(y)$.
3. $\rho_\tau\left(T_\ell(x) - T_\ell(y)\right) \leq \rho_\tau(x - y)$.

Proof of Lemma 2.5.3 Let $x, y \in \mathbb{R}$ and $\ell \in \mathbb{N}$.

1. We have
$$|\rho_\tau(x)| = \left|x(\tau - \mathbb{1}_{\{x\leq 0\}})\right| = |x|\left|\tau - \mathbb{1}_{\{x\leq 0\}}\right| \leq |x|.$$

2. Clearly,
$$\begin{aligned} & \rho_\tau(x+y) \leq \rho_\tau(x) + \rho_\tau(y) \\ \iff \quad & x\mathbb{1}_{\{x\leq 0\}} + y\mathbb{1}_{\{y\leq 0\}} \leq x\mathbb{1}_{\{x+y\leq 0\}} + y\mathbb{1}_{\{x+y\leq 0\}}. \end{aligned}$$
The conclusion follows by examining the different positions of x and y with respect to 0.

3. If $x > \ell$ and $|y| \leq \ell$, then
$$\begin{aligned} \rho_\tau\left(T_\ell(x) - T_\ell(y)\right) &= \rho_\tau(\ell - y) \\ &= (\ell - y)(\tau - \mathbb{1}_{\{\ell - y\leq 0\}}) \\ &= (\ell - y)\tau \\ &\leq (x - y)\tau \\ &= (x - y)(\tau - \mathbb{1}_{\{x - y\leq 0\}}) \\ &= \rho_\tau(x - y). \end{aligned}$$
Similarly, if $x < -\ell$ and $|y| \leq \ell$, then
$$\begin{aligned} \rho_\tau\left(T_\ell(x) - T_\ell(y)\right) &= \rho_\tau(-\ell - y) \\ &= (-\ell - y)(\tau - \mathbb{1}_{\{-\ell - y\leq 0\}}) \\ &= (-\ell - y)(\tau - 1) \\ &\leq (x - y)(\tau - 1) \\ &= (x - y)(\tau - \mathbb{1}_{\{x - y\leq 0\}}) \\ &= \rho_\tau(x - y). \end{aligned}$$
All the other cases are similar and left to the reader.

□

Recall that a sequence $\{\mu_n\}_{n=1}^{\infty}$ of probability measures on $\mathbb{R}$ is defined to converge weakly to the probability measure μ_∞ if for every bounded, continuous real function f,
$$\int f \mathrm{d}\mu_n \longrightarrow \int f \mathrm{d}\mu_\infty \quad \text{as } n \to \infty.$$
Recall also that the sequence $\{\mu_n\}_{n=1}^{\infty}$ is said to be uniformly integrable if
$$\lim_{\alpha\to\infty} \sup_{n\geq 1} \int_{|x|\geq\alpha} |x| \mathrm{d}\mu_n(x) = 0.$$

Moreover, if

$$\sup_{n\geq 1}\int |x|^{1+\varepsilon}\mathrm{d}\mu_n(x) < \infty$$

for some positive ε, then the sequence $\{\mu_n\}_{n=1}^{\infty}$ is uniformly integrable (Billingsley [16]).

The next lemma may be summarized by saying that if a sequence of probability measures converges in terms of weak convergence topology, then the associated quantile sequence will converge too.

Lemma 2.5.4. *Let $\{\mu_n\}_{n=1}^{\infty}$ be a uniformly integrable sequence of real probability measures, and let μ_∞ be a probability measure with (strictly) increasing distribution function. Suppose that $\{\mu_n\}_{n=1}^{\infty}$ converges weakly to μ_∞. Then, for all $\tau \in (0,1)$,*

$$q_{\tau,n} \longrightarrow q_{\tau,\infty} \quad \textit{as } n \to \infty,$$

where $q_{\tau,n} \in \mathcal{Q}_\tau(\mu_n)$ for all $n \geq 1$ and $\{q_{\tau,\infty}\} = \mathcal{Q}_\tau(\mu_\infty)$.

Proof of Lemma 2.5.4 Since $\{\mu_n\}_{n=1}^{\infty}$ converges weakly to μ_∞, it is a tight sequence. Consequently, there is a compact set, say $[-M, M]$, such that $\mu_n(\mathbb{R} \setminus [-M, M]) < \min(\tau, 1-\tau)$. This implies $q_{\tau,n} \in [-M, M]$ for all $n \geq 1$. Consequently, it will be enough to prove that any consistent subsequence of $\{q_{\tau,n}\}_{n=1}^{\infty}$ converges towards $q_{\tau,\infty}$.

Using a slight abuse of notation, we still denote by $\{q_{\tau,n}\}_{n=1}^{\infty}$ a consistent subsequence of the original sequence, and let $q_{\tau,\star}$ be such that $\lim_{n\to\infty} q_{\tau,n} = q_{\tau,\star}$. Using the assumption on the distribution function of μ_∞, we know by Lemma 2.2.1 that $q_{\tau,\infty}$ is the unique minimizer of problem (2.1). Therefore, to show that $q_{\tau,\star} = q_{\tau,\infty}$, it suffices to prove that, for any $q \in \mathbb{R}$,

$$\mathbb{E}_{\mu_\infty}\left[\rho_\tau(Y-q)\right] \geq \mathbb{E}_{\mu_\infty}\left[\rho_\tau(Y-q_{\tau,\star})\right].$$

Fix $q \in \mathbb{R}$. We first prove that

$$\mathbb{E}_{\mu_n}\left[\rho_\tau(Y-q)\right] \longrightarrow \mathbb{E}_{\mu_\infty}\left[\rho_\tau(Y-q)\right] \quad \text{as } n \to \infty. \tag{2.7}$$

To see this, for $M > 0$ and all $y \in \mathbb{R}$, set

$$\rho_\tau^{(+,M)}(y) = \begin{cases} 0 & \text{if } |y| < M; \\ \rho_\tau(y) & \text{if } |y| > M+1; \\ \rho_\tau(M+1)(y-M) & \text{if } y \in [M, M+1]; \\ \rho_\tau(-M-1)(y+M) & \text{if } y \in [-M-1, -M]. \end{cases}$$

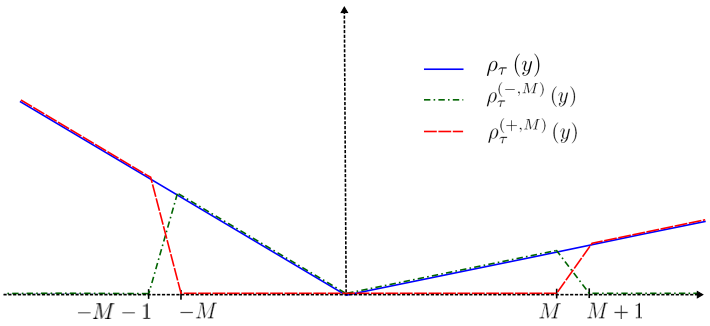

Figure 2.3: Illustration of the decomposition $\rho_\tau = \rho_\tau^{(+,M)} + \rho_\tau^{(-,M)}$.

The function $\rho_\tau^{(+,M)}$ is continuous and, for all $z \in \mathbb{R}$, satisfies the inequality $\rho_\tau^{(+,M)}(z) \leq \rho_\tau(z)\mathbb{1}_{\{|z|>M\}}$. In the sequel, we will denote by $\rho_\tau^{(-,M)}$ the bounded and continuous map $\rho_\tau - \rho_\tau^{(+,M)}$. The decomposition $\rho_\tau = \rho_\tau^{(+,M)} + \rho_\tau^{(-,M)}$ is illustrated in Figure 2.3.

Next, fix $\varepsilon > 0$ and choose M large enough to ensure

$$\sup_{n\geq 1}\left(\mathbb{E}_{\mu_n}\left[|Y-q|\mathbb{1}_{\{|Y-q|>M\}}\right]\right) + \mathbb{E}_{\mu_\infty}\left[|Y-q|\mathbb{1}_{\{|Y-q|>M\}}\right] < \varepsilon/2.$$

Choose also n sufficiently large to have

$$\left|\mathbb{E}_{\mu_n}\left[\rho_\tau^{(-,M)}(Y-q)\right] - \mathbb{E}_{\mu_\infty}\left[\rho_\tau^{(-,M)}(Y-q)\right]\right| < \varepsilon/2.$$

Write

$$\begin{aligned}
&\left|\mathbb{E}_{\mu_n}\left[\rho_\tau(Y-q)\right] - \mathbb{E}_{\mu_\infty}\left[\rho_\tau(Y-q)\right]\right| \\
&\quad = \Big|\mathbb{E}_{\mu_n}\left[\rho_\tau^{(+,M)}(Y-q)\right] + \mathbb{E}_{\mu_n}\left[\rho_\tau^{(-,M)}(Y-q)\right] \\
&\qquad - \mathbb{E}_{\mu_\infty}\left[\rho_\tau^{(+,M)}(Y-q)\right] - \mathbb{E}_{\mu_\infty}\left[\rho_\tau^{(-,M)}(Y-q)\right]\Big|.
\end{aligned}$$

Thus

$$\begin{aligned}
&|\mathbb{E}_{\mu_n}[\rho_\tau(Y-q)] - \mathbb{E}_{\mu_\infty}[\rho_\tau(Y-q)]| \\
&\quad\le \left|\mathbb{E}_{\mu_n}\left[\rho_\tau^{(-,M)}(Y-q)\right] - \mathbb{E}_{\mu_\infty}\left[\rho_\tau^{(-,M)}(Y-q)\right]\right| \\
&\qquad + \left|\mathbb{E}_{\mu_n}\left[\rho_\tau^{(+,M)}(Y-q)\right] - \mathbb{E}_{\mu_\infty}\left[\rho_\tau^{(+,M)}(Y-q)\right]\right| \\
&\quad\le \left|\mathbb{E}_{\mu_n}\left[\rho_\tau^{(-,M)}(Y-q)\right] - \mathbb{E}_{\mu_\infty}\left[\rho_\tau^{(-,M)}(Y-q)\right]\right| \\
&\qquad + \left|\mathbb{E}_{\mu_n}\left[\rho_\tau(Y-q)\mathbb{1}_{\{|Y-q|>M\}}\right]\right| + \left|\mathbb{E}_{\mu_\infty}\left[\rho_\tau(Y-q)\mathbb{1}_{\{|Y-q|>M\}}\right]\right| \\
&\quad\le \left|\mathbb{E}_{\mu_n}\left[\rho_\tau^{(-,M)}(Y-q)\right] - \mathbb{E}_{\mu_\infty}\left[\rho_\tau^{(-,M)}(Y-q)\right]\right| \\
&\qquad + \sup_n \mathbb{E}_{\mu_n}\left[|Y-q|\mathbb{1}_{\{|Y-q|>M\}}\right] + \mathbb{E}_{\mu_\infty}\left[|Y-q|\mathbb{1}_{\{|Y-q|>M\}}\right] \\
&\quad\le \varepsilon.
\end{aligned}$$

for all large enough n. This shows (2.7).

Next, using the fact that the function ρ_τ is uniformly continuous, we may write, for sufficiently large n and all $y \in \mathbb{R}$,

$$\rho_\tau(y - q_{\tau,n}) \ge \rho_\tau(y - q_{\tau,\star}) - \varepsilon. \tag{2.8}$$

Therefore, for all large enough n,

$$\begin{aligned}
\mathbb{E}_{\mu_\infty}[\rho_\tau(Y-q)] &\ge \mathbb{E}_{\mu_n}[\rho_\tau(Y-q)] - \varepsilon \\
&\quad \text{(by identity (2.7))} \\
&\ge \mathbb{E}_{\mu_n}[\rho_\tau(Y - q_{\tau,n})] - \varepsilon \\
&\ge \mathbb{E}_{\mu_n}[\rho_\tau(Y - q_{\tau,\star})] - 2\varepsilon \\
&\quad \text{(by inequality(2.8))} \\
&\ge \mathbb{E}_{\mu_\infty}[\rho_\tau(Y - q_{\tau,\star})] - 3\varepsilon \\
&\quad \text{(by identity (2.7)).}
\end{aligned}$$

Letting $\varepsilon \longrightarrow 0$ leads to the desired result.

□

We are now in a position to prove Theorem 2.3.1.

Because of inequality (2.3) it is enough to show that

$$\limsup_{n\to\infty} L_n(g) \le L^\star \quad \text{a.s.}$$

With this in mind, we first provide an upper bound on the first term of the right hand side of the inequality in Lemma 2.5.2. We have

$$\begin{aligned}
&\limsup_{n\to\infty} \inf_{k,\ell} \left(L_n \left(h_n^{(k,\ell)} - \frac{2 \ln b_{k,\ell}}{n\eta_{n+1}} \right) \right) \\
&\quad \leq \inf_{k,\ell} \left(\limsup_{n\to\infty} L_n \left(h_n^{(k,\ell)} - \frac{2 \ln b_{k,\ell}}{n\eta_{n+1}} \right) \right) \\
&\quad \leq \inf_{k,\ell} \left(\limsup_{n\to\infty} L_n \left(h_n^{(k,\ell)} \right) \right).
\end{aligned}$$

To evaluate $\limsup_{n\to\infty} L_n(h_n^{(k,\ell)})$, we investigate the performance of the expert $h_n^{(k,\ell)}$ on the stationary and ergodic sequence $Y_0, Y_{-1}, Y_{-2}, \ldots$ Fix $p_\ell \in (0,1)$, $\mathbf{s} \in \mathbb{R}^k$, and set $\tilde{\ell} = \lfloor p_\ell j \rfloor$, where j is a positive integer.

For $j > k + \tilde{\ell} + 1$, introduce the set

$$\begin{aligned}
\tilde{J}_{j,\mathbf{s}}^{(k,\tilde{\ell})} = \Big\{ &-j + k + 1 \leq i \leq 0 : Y_{i-k}^{i-1} \text{ is among the } \tilde{\ell}\text{-NN of } \mathbf{s} \\
&\text{in } Y_{-k}^{-1}, \ldots, Y_{-j+1}^{-j+k} \Big\}.
\end{aligned}$$

For any real number a, we denote by δ_a the Dirac (point) measure at a. Let the random measure $\mathbb{P}_{j,\mathbf{s}}^{(k,\ell)}$ be defined by

$$\mathbb{P}_{j,\mathbf{s}}^{(k,\ell)} = \frac{1}{\left| \tilde{J}_{j,\mathbf{s}}^{(k,\tilde{\ell})} \right|} \sum_{i \in \tilde{J}_{j,\mathbf{s}}^{(k,\tilde{\ell})}} \delta_{Y_i}.$$

Take an arbitrary radius $r_{k,\ell}(\mathbf{s})$ such that

$$\mathbb{P}\left[\|Y_{-k}^{-1} - \mathbf{s}\| \leq r_{k,\ell}(\mathbf{s}) \right] = p_\ell.$$

A straightforward adaptation of an argument in Theorem 3.1 of [60] shows that

$$\mathbb{P}_{j,\mathbf{s}}^{(k,\ell)} \xrightarrow[j\to\infty]{} \mathbb{P}_{Y_0 \,|\, \|Y_{-k}^{-1} - \mathbf{s}\| \leq r_{k,\ell}(\mathbf{s})} \triangleq \mathbb{P}_{\infty,\mathbf{s}}^{(k,\ell)}$$

almost surely in terms of weak convergence. Moreover, by a double application of the ergodic theorem (see for instance [13]),

$$\int y^2 \mathrm{d}\mathbb{P}_{j,\mathbf{s}}^{(k,\ell)}(y) \xrightarrow[j\to\infty]{} \int y^2 \mathrm{d}\mathbb{P}_{\infty,\mathbf{s}}^{(k,\ell)}(y) \quad \text{a.s.}$$

Thus

$$\sup_{j\geq 0} \int y^2 \mathrm{d}\mathbb{P}_{j,\mathbf{s}}^{(k,\ell)}(y) < \infty \quad \text{a.s.},$$

and, consequently, the sequence $\{\mathbb{P}^{(k,\ell)}_{j,\mathbf{s}}\}_{j=1}^{\infty}$ is uniformly integrable.

By assumption $(H3)$ the distribution function of the measure $\mathbb{P}_{Y_0|Y_{-\infty}^{-1}}$ is a.s. increasing. We also have $\sigma\left(\|Y_{-k}^{-1}-\mathbf{s}\|\le r_{k,\ell}(\mathbf{s})\right)\subset\sigma\left(Y_{-\infty}^{-1}\right)$ where $\sigma(X)$ denotes the sigma algebra generated by the random variable X. Thus the distribution function of $\mathbb{P}^{(k,\ell)}_{\infty,\mathbf{s}}=\mathbb{P}_{Y_0\,|\,\|Y_{-k}^{-1}-\mathbf{s}\|\le r_{k,\ell}(\mathbf{s})}$ is a.s. increasing, too. Hence, letting

$$q^{(k,\ell)}_{\tau,j}(Y_{-j+1}^{-1},\mathbf{s})\in\mathcal{Q}_\tau(\mathbb{P}^{(k,\ell)}_{j,\mathbf{s}})\quad\text{and}\quad\left\{q^{(k,\ell)}_{\tau,\infty}(\mathbf{s})\right\}=\mathcal{Q}_\tau(\mathbb{P}^{(k,\ell)}_{\infty,\mathbf{s}}),$$

we may apply Lemma 2.5.4, and obtain

$$q^{(k,\ell)}_{\tau,j}(Y_{-j+1}^{-1},\mathbf{s})\xrightarrow[j\to\infty]{}q^{(k,\ell)}_{\tau,\infty}(\mathbf{s})\qquad\text{a.s.}$$

Consequently, for any $y_0\in\mathbb{R}$,

$$\rho_\tau\left(y_0-T_{\min(j^\delta,\ell)}\left(q^{(k,\ell)}_{\tau,j}(Y_{-j+1}^{-1},\mathbf{s})\right)\right)\xrightarrow[j\to\infty]{}\rho_\tau\left(y_0-T_\ell\left(q^{(k,\ell)}_{\tau,\infty}(\mathbf{s})\right)\right)\qquad\text{a.s.}$$

Since y_0 and $\mathbf{s}$ are arbitrary, we are led to

$$\rho_\tau\left(Y_0-T_{\min(j^\delta,\ell)}\left(q^{(k,\ell)}_{\tau,j}(Y_{-j+1}^{-1},Y_{-k}^{-1})\right)\right)\xrightarrow[j\to\infty]{}\rho_\tau\left(Y_0-T_\ell\left(q^{(k,\ell)}_{\tau,\infty}(Y_{-k}^{-1})\right)\right)\qquad\text{a.s.}\tag{2.9}$$

For $y=(\ldots,y_{-1},y_0,y_1,\ldots)$, set

$$\begin{aligned}f_j(y)&=\rho_\tau\left(y_0-h^{(k,\ell)}_j(y_{-j+1}^{-1})\right)\\&=\rho_\tau\left(y_0-T_{\min(j^\delta,\ell)}\left(q^{(k,\ell)}_{\tau,j}(y_{-j+1}^{-1},y_{-k}^{-1})\right)\right).\end{aligned}$$

Clearly,

$$\begin{aligned}|f_j(Y)|&=\left|\rho_\tau\left(Y_0-h^{(k,\ell)}_j(Y_{-j+1}^{-1})\right)\right|\\&\le\left|Y_0-T_{\min(j^\delta,\ell)}(Y_{-j+1}^{-1})\right|\\&\qquad\text{(by statement 1. of Lemma 2.5.3)}\\&\le|Y_0|+\left|T_{\min(j^\delta,\ell)}(Y_{-j+1}^{-1})\right|\\&\le|Y_0|+\ell,\end{aligned}$$

and thus $\mathbb{E}[\sup_j|f_j(Y)|]<\infty$. By identity (2.9),

$$f_j(Y)\xrightarrow[j\to\infty]{}\rho_\tau\left(Y_0-T_\ell(q^{(k,\ell)}_{\tau,\infty}(Y_{-k}^{-1})\right)\quad\text{a.s.}$$

Consequently, Lemma 2.5.1 yields

$$L_n(h^{(k,\ell)}_n)\xrightarrow[n\to\infty]{}\mathbb{E}\left[\rho_\tau\left(Y_0-T_\ell\left(q^{(k,\ell)}_{\tau,\infty}(Y_{-k}^{-1})\right)\right)\right].$$

To lighten notation a bit, we set

$$\varepsilon_{k,\ell} = \mathbb{E}\left[\rho_\tau\left(Y_0 - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\right]$$

and proceed now to prove that $\lim_{k\to\infty}\lim_{\ell\to\infty}\varepsilon_{k,\ell} \leq L^\star$.

We have, a.s., in terms of weak convergence,

$$\mathbb{P}^{(k,\ell)}_{\infty,Y_{-k}^{-1}} \xrightarrow[\ell\to\infty]{} \mathbb{P}_{Y_0\,|\,Y_{-k}^{-1}}$$

(see for instance Theorem 3.1 in [60]). Next, with a slight modification of techniques of Theorem 2.2 in [13],

$$\int y^2 \mathrm{d}\mathbb{P}^{(k,\ell)}_{\infty,Y_{-k}^{-1}}(y) \xrightarrow[\ell\to\infty]{} \int y^2 \mathrm{d}\mathbb{P}_{Y_0\,|\,Y_{-k}^{-1}}(y) \quad \text{a.s.},$$

which leads to

$$\sup_{\ell\geq 0}\int y^2 \mathrm{d}\mathbb{P}^{(k,\ell)}_{\infty,Y_{-k}^{-1}}(y) < \infty \quad \text{a.s.}$$

Moreover, by assumption $(H3)$, the distribution function of $\mathbb{P}_{Y_0\,|\,Y_{-k}^{-1}}$ is a.s. increasing. Thus, setting

$$\left\{q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right\} = \mathcal{Q}_\tau(\mathbb{P}^{(k,\ell)}_{\infty,Y_{-k}^{-1}}) \quad \text{and}\left\{q_\tau^{(k)}(Y_{-k}^{-1})\right\} = \mathcal{Q}_\tau(\mathbb{P}_{Y_0\,|\,Y_{-k}^{-1}})$$

and applying Lemma 2.5.4 yields

$$q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1}) \xrightarrow[\ell\to\infty]{} q_\tau^{(k)}(Y_{-k}^{-1}) \quad \text{a.s.}$$

Consequently,

$$\rho_\tau\left(Y_0 - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right) \xrightarrow[\ell\to\infty]{} \rho_\tau\left(Y_0 - q_\tau^{(k)}(Y_{-k}^{-1})\right) \quad \text{a.s.}$$

It turns out that the above convergence also holds in mean. To see this, note first that

$$\begin{aligned}
&\rho_\tau\left(Y_0 - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\\
&\quad= \rho_\tau\left(Y_0 - T_\ell(Y_0) + T_\ell(Y_0) - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\\
&\quad\leq \rho_\tau\left(Y_0 - T_\ell(Y_0)\right) + \rho_\tau\left(T_\ell(Y_0) - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\\
&\qquad\text{(by statement 2. of Lemma 2.5.3)}\\
&\quad\leq 2|Y_0| + \rho_\tau\left(Y_0 - q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right) \quad \text{a.s.}\\
&\qquad\text{(by statement 3. of Lemma 2.5.3).}
\end{aligned}$$

Thus

$$\begin{aligned}
&\mathbb{E}\left[\left(\rho_\tau\left(Y_0 - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\right)^2\right] \\
&\quad \le \mathbb{E}\left[\left(2|Y_0| + \rho_\tau\left(Y_0 - q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)^2\right] \\
&\quad \le 8\mathbb{E}\left[Y_0^2\right] + 2\mathbb{E}\left[\left(\rho_\tau\left(Y_0 - q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)^2\right].
\end{aligned}$$

In addition,

$$\begin{aligned}
&\sup_{\ell\ge 1}\mathbb{E}\left[\left(\rho_\tau\left(Y_0 - q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)^2\right] \\
&\quad = \sup_{\ell\ge 1}\mathbb{E}\left[\left(\min_{q(.)}\mathbb{E}_{\mathbb{P}_{\infty,Y_{-k}^{-1}}^{(k,\ell)}}\rho_\tau\left(Y_0 - q(Y_{-k}^{-1})\right)\right)^2\right] \\
&\quad \le \mathbb{E}\left[\left(\rho_\tau(Y_0)\right)^2\right] \\
&\qquad \text{(by Jensen's inequality)} \\
&\quad \le \mathbb{E}\left[Y_0^2\right] < \infty.
\end{aligned}$$

This implies

$$\mathbb{E}\left[\left(\rho_\tau\left(Y_0 - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\right)^2\right] < \infty,$$

i.e., the sequence is uniformly integrable. Thus we obtain, as desired,

$$\lim_{\ell\to\infty}\mathbb{E}\left[\rho_\tau\left(Y_0 - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\right] = \mathbb{E}\left[\rho_\tau\left(Y_0 - q_\tau^{(k)}(Y_{-k}^{-1})\right)\right].$$

Putting all pieces together,

$$\begin{aligned}
\lim_{\ell\to\infty}\varepsilon_{k,\ell} &= \lim_{\ell\to\infty}\mathbb{E}\left[\rho_\tau\left(Y_0 - T_\ell\left(q_{\tau,\infty}^{(k,\ell)}(Y_{-k}^{-1})\right)\right)\right] \\
&= \mathbb{E}\left[\rho_\tau\left(Y_0 - q_\tau^{(k)}(Y_{-k}^{-1})\right)\right] \\
&\triangleq \varepsilon_k^\star.
\end{aligned}$$

It remains to prove that $\lim_{k\to\infty}\varepsilon_k^\star = L^\star$. To this aim, for all $k \ge 1$, let Z_k be the $\sigma(Y_{-k}^{-1})$-measurable random variable defined by

$$Z_k = \rho_\tau\left(Y_0 - q_\tau^{(k)}(Y_{-k}^{-1})\right) = \min_{q(.)}\mathbb{E}_{\mathbb{P}_{Y_0\,|\,Y_{-k}^{-1}}}\left[\rho_\tau\left(Y_0 - q(Y_{-k}^{-1})\right)\right].$$

Observe that $\{Z_k\}_{k=0}^\infty$ is a nonnegative supermartingale with respect to the family

of sigma algebras $\{\sigma(Y_{-k}^{-1})\}_{k=1}^{\infty}$. In addition,

$$\begin{aligned}
\sup_{k\geq 1}\mathbb{E}[Z_k^2] &= \sup_{k\geq 1}\mathbb{E}\left[\left(\min_{q(.)}\mathbb{E}_{\mathbb{P}_{Y_0\,|\,Y_{-k}^{-1}}}\rho_\tau\left(Y_0 - q(Y_k^{-1})\right)\right)^2\right] \\
&\leq \sup_{k\geq 1}\mathbb{E}\left[\left(\mathbb{E}_{\mathbb{P}_{Y_0\,|\,Y_{-k}^{-1}}}\rho_\tau\left(Y_0\right)\right)^2\right] \\
&\leq \sup_{k\geq 1}\mathbb{E}\left[\left(\rho_\tau(Y_0)\right)^2\right] \\
&\quad \text{(by Jensen's inequality)} \\
&\leq \sup_{k\geq 1}\mathbb{E}[Y_0^2] < \infty.
\end{aligned}$$

Therefore,

$$\mathbb{E}[Z_k] \xrightarrow[k\to\infty]{} \mathbb{E}[Z_\infty],$$

where

$$Z_\infty = \min_{q(.)}\mathbb{E}_{\mathbb{P}_{Y_0\,|\,Y_{-\infty}^{-1}}}\left[\rho_\tau\left(Y_0 - q(Y_{-\infty}^{-1})\right)\right].$$

Consequently,

$$\lim_{k\to\infty}\varepsilon_k^\star = L^\star.$$

We finish the proof by using Lemma 2.5.2. On the one hand, a.s.,

$$\begin{aligned}
&\limsup_{n\to\infty}\inf_{k,\ell}\left(L_n\left(h_n^{(k,\ell)} - \frac{2\ln b_{k,\ell}}{n\eta_{n+1}}\right)\right) \\
&\leq \inf_{k,\ell}\left(\limsup_{n\to\infty} L_n\left(h_n^{(k,\ell)} - \frac{2\ln b_{k,\ell}}{n\eta_{n+1}}\right)\right) \\
&\leq \inf_{k,\ell}\left(\limsup_{n\to\infty} L_n\left(h_n^{(k,\ell)}\right)\right) \\
&= \inf_{k,\ell}\varepsilon_{k,\ell} \\
&\leq \lim_{k\to\infty}\lim_{\ell\to\infty}\varepsilon_{k,\ell} \\
&\leq L^\star.
\end{aligned}$$

Moreover,

$$\begin{aligned}
&\frac{1}{2n}\sum_{t=1}^{n}\eta_t\sum_{k,\ell=1}^{\infty}p_{k,\ell,n}\left[\rho_\tau\left(Y_t - h_t^{(k,\ell)}(Y_1^{t-1})\right)\right]^2 \\
&\leq \frac{1}{2n}\sum_{t=1}^{n}\eta_t\sum_{k=1}^{\infty}\sum_{\ell=1}^{\infty}p_{k,\ell,n}\left[\rho_\tau\left(Y_t - T_{\min(t^\delta,\ell)}\left(\bar{h}_t^{(k,\ell)}(Y_1^{t-1})\right)\right)\right]^2 \\
&\leq \frac{1}{2n}\sum_{t=1}^{n}\eta_t\sum_{k=1}^{\infty}\sum_{\ell=1}^{\infty}p_{k,\ell,n}\left|Y_t - T_{\min(t^\delta,\ell)}\left(\bar{h}_t^{(k,\ell)}(Y_1^{t-1})\right)\right|^2.
\end{aligned}$$

Thus

$$\begin{aligned}
&\frac{1}{2n}\sum_{t=1}^{n}\eta_t\sum_{k,\ell=1}^{\infty}p_{k,\ell,n}\left[\rho_\tau\left(Y_t-h_t^{(k,\ell)}(Y_1^{t-1})\right)\right]^2\\
&\quad\le\frac{1}{n}\sum_{t=1}^{n}\eta_t\sum_{k=1}^{\infty}\left(\sum_{\ell=1}^{\infty}p_{k,\ell,n}|Y_t|^2+\sum_{\ell=1}^{\infty}p_{k,\ell,n}\left[T_{\min(t^\delta,\ell)}\left(\bar{h}_t^{(k,\ell)}(Y_1^{t-1})\right)\right]^2\right)\\
&\quad\le\frac{1}{n}\sum_{t=1}^{n}\eta_t\sum_{k=1}^{\infty}\left(\sum_{\ell=1}^{\infty}p_{k,\ell,n}|Y_t|^2+\sum_{\ell=1}^{\infty}p_{k,\ell,n}t^{2\delta}\right)\\
&\quad\le\frac{1}{n}\sum_{t=1}^{n}\eta_t\sum_{k,\ell=1}^{\infty}p_{k,\ell,n}(Y_t^2+t^{2\delta})\\
&\quad=\frac{1}{n}\sum_{t=1}^{n}\eta_t(t^{2\delta}+Y_t^2).
\end{aligned}$$

Therefore, since $n^{2\delta}\eta_n\longrightarrow 0$ as $n\to\infty$ and $\mathbb{E}[Y_0^2]<\infty$,

$$\limsup_{n\to\infty}\frac{1}{2n}\sum_{t=1}^{n}\eta_t\sum_{k,\ell=1}^{\infty}p_{k,\ell,n}\left[\rho_\tau\left(Y_t-h_t^{(k,\ell)}(Y_1^{t-1})\right)\right]^2=0\quad\text{a.s.}$$

Putting all pieces together, we obtain, a.s.,

$$\limsup_{n\to\infty}L_n(g)\le L^\star,$$

and this proves the result.

□

2.5.2 Proof of Lemma 2.2.1

To prove the first statement of the lemma, it will be enough to show that, for all $q\in\mathbb{R}$,

$$\mathbb{E}\left[\rho_\tau\left(Y-q\right)\right]-\mathbb{E}\left[\rho_\tau\left(Y-q_\tau\right)\right]\ge 0.$$

We separate the cases $q\ge q_\tau$ and $q<q_\tau$.

(i) If $q\ge q_\tau$, then

$$\begin{aligned}
&\mathbb{E}\left[\rho_\tau(Y-q)\right]-\mathbb{E}\left[\rho_\tau(Y-q_\tau)\right]\\
&\quad=\mathbb{E}\left[(Y-q)(\tau-\mathbb{1}_{\{Y\le q\}})-(Y-q_\tau)(\tau-\mathbb{1}_{\{Y\le q_\tau\}})\right]\\
&\quad=\mathbb{E}\left[(Y-q)\left(\tau-(\mathbb{1}_{\{Y\le q_\tau\}}+\mathbb{1}_{\{q_\tau<Y\le q\}})\right)-(Y-q_\tau)(\tau-\mathbb{1}_{\{Y\le q_\tau\}})\right]\\
&\quad=\mathbb{E}\left[(q_\tau-q)(\tau-\mathbb{1}_{\{Y\le q_\tau\}})\right]-\mathbb{E}\left[(Y-q)\mathbb{1}_{\{q_\tau<Y\le q\}}\right].
\end{aligned}$$

We have

$$\begin{aligned}\mathbb{E}\left[(q_\tau - q)(\tau - \mathbb{1}_{\{Y \le q_\tau\}})\right] &= (q_\tau - q)\left(\tau - \mathbb{P}[Y \le q_\tau]\right) \\ &= (q_\tau - q)\left[\tau - F_Y\left(F_Y^{\leftarrow}(\tau)\right)\right] \\ &\ge 0\end{aligned}$$

and, clearly,

$$-\mathbb{E}\left[(Y - q)\mathbb{1}_{\{q_\tau < Y \le q\}}\right] \ge 0.$$

This proves the desired statement.

(*ii*) If $q < q_\tau$, then

$$\begin{aligned}&\mathbb{E}\left[\rho_\tau\left(Y - q\right)\right] - \mathbb{E}\left[\rho_\tau(Y - q_\tau)\right] \\ &\quad= \mathbb{E}\left[(Y - q)(\tau - \mathbb{1}_{\{Y \le q\}}) - (Y - q_\tau)(\tau - \mathbb{1}_{\{Y \le q_\tau\}})\right] \\ &\quad= \mathbb{E}\left[(Y - q)(\tau - \mathbb{1}_{\{Y \le q\}}) - (Y - q_\tau)\left(\tau - (\mathbb{1}_{\{Y \le q\}} + \mathbb{1}_{\{q < Y \le q_\tau\}})\right)\right] \\ &\quad= \mathbb{E}\left[(q_\tau - q)(\tau - \mathbb{1}_{\{Y \le q\}})\right] - \mathbb{E}\left[(Y - q_\tau)(\tau - \mathbb{1}_{\{q < Y \le q_\tau\}})\right].\end{aligned}$$

For $q < q_\tau$, $\mathbb{P}\left[Y \le q\right] = F_Y(q) < \tau$. Consequently

$$\mathbb{E}\left[(q_\tau - q)(\tau - \mathbb{1}_{\{Y \le q\}})\right] > 0.$$

Since

$$-\mathbb{E}\left[(Y - q_\tau)(\tau - \mathbb{1}_{\{q < Y \le q_\tau\}})\right] \ge 0,$$

we are led to the desired result.

Suppose now that F_Y is increasing. To establish the second statement of the lemma, a quick inspection of the proof reveals that it is enough to prove that, for $q > q_\tau$,

$$\mathbb{E}\left[(Y - q)\mathbb{1}_{\{q_\tau < Y \le q\}}\right] < 0.$$

Take $q' \in (q_\tau, q)$ and set $S = [q_\tau < Y \le q']$. Clearly,

$$\mathbb{P}(S) = F_Y(q') - F_Y(q_\tau) > 0.$$

Therefore

$$\mathbb{E}\left[(Y - q)\mathbb{1}_{\{q_\tau < Y \le q\}}\right] \le \mathbb{E}[(Y - q)\mathbb{1}_S] < 0.$$

□

3 Convergence of Distributed Asynchronous Learning Vector Quantization algorithms

Ce chapitre a fait l'objet d'un article actuellement soumis.

3.1 Introduction

Distributed algorithms arise in a wide range of applications, including telecommunications, distributed information processing, scientific computing, real time process control and many others. Parallelization is one of the most promising ways to harness greater computing resources, whereas building faster serial computers is increasingly expensive and also faces some physical limits such as transmission speeds and miniaturization. One of the challenges proposed for Machine Learning is to build scalable applications that quickly process large amounts of data in sophisticated ways. Building such large scale algorithms attacks several problems in a distributed framework, such as communication delays in the network or numerous problems caused by the lack of shared memory.

Clustering algorithms are one of the primary tools of unsupervised learning. From a practical perspective, clustering plays an outstanding role in data mining applications such as text mining, web analysis, marketing, medical diagnostics, computational biology and many others. Clustering is a separation of data into groups of similar objects. As clustering represents the data with fewer clusters, there is a necessary loss of certain fine details, but simplification is achieved. The popular Competitive Learning Vector Quantization (CLVQ) algorithm (see Gersho and Gray [48]) provides a technique for building reliable clusters characterized by their prototypes. As pointed out by Bottou in [21], the CLVQ algorithm can also be viewed as the on-line version of the widespread Lloyd's method (see Lloyd's [73] for the definition) which is referred to as batch k-means in [21]. The CLVQ also belongs to the class of stochastic gradient descent algorithms (for more information on stochastic gradient descent procedures we refer the reader to Benveniste et al. [9]).

The analysis of parallel stochastic gradient procedures in a Machine Learning context has recently received a great deal of attention (see for instance Zinkevich et al. [97] and Mac Donald et al. [77]). In the present paper, we go further by introducing a model that brings together the original CLVQ algorithm and the comprehensive theory of asynchronous parallel linear algorithms developed by Tsitsiklis [93], Tsitsiklis et al. [94] and Bertsekas and Tsitsiklis [11]. The resulting model will be called Distributed Asynchronous Learning Vector Quantization (DALVQ for short). At a high level, the DALVQ algorithm parallelizes several executions of the CLVQ method concurrently on different processors while the results of these algorithms are broadcast through the distributed framework asynchronously and efficiently. Here, the term processor refers to any computing instance in a distributed architecture (see Bullo et al. [27, Chapter 1] for more details). Let us remark that there is a series of publications similar in spirit to this paper. Indeed in Frasca et al. [45] and in Durham et al. [40], a coverage control problem is formulated as an optimization problem where the functional cost to be minimized is the same of the quantization problem stated in this manuscript.

Let us provide a brief mathematical introduction to the CLVQ technique and DALVQ algorithms. The first technique computes quantization scheme for d dimensional samples $\mathbf{z}_1, \mathbf{z}_2, \ldots$ using the following iterations on a $\left(\mathbb{R}^d\right)^\kappa$ vector,

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

In the equation above, $w(0) \in \left(\mathbb{R}^d\right)^\kappa$ and the ε_t are positive reals. The vector $H(\mathbf{z}, w)$ is the opposite of the difference between the sample $\mathbf{z}$ and its nearest component in w. Assume that there are M computing entities, the data are split among the memory of these machines: $\mathbf{z}_1^i, \mathbf{z}_2^i, \ldots$, where $i \in \{1, \ldots, M\}$. Therefore, the DALVQ algorithms are defined by the M iterations $\{w^i(t)\}_{t=0}^\infty$, called versions, satisfying (with slight simplifications)

$$w^i(t+1) = \sum_{j=1}^M a^{i,j}(t) w^j(\tau^{i,j}(t)) - \varepsilon_{t+1}^i H\left(\mathbf{z}_{t+1}^i, w^i(t)\right),$$

$i \in \{1, \ldots, M\}$ and $t \geq 0$. The time instants $\tau^{i,j}(t) \geq 0$ are deterministic but unknown and the delays satisfy $t - \tau^{i,j}(t) \geq 0$. The families $\{a^{i,j}(t)\}_{j=1}^M$ define the weights of convex combinations.

As a striking result, we prove that multiple versions of the quantizers, distributed among the processors in a parallel architecture, asymptotically reach a consensus almost surely. Using the materials introduced above, it writes

$$w^i(t) - w^j(t) \xrightarrow[t\to\infty]{} 0, \quad (i,j) \in \{1, \ldots, M\}^2, \text{ almost surely (a.s.)}.$$

Furthermore, we also show that these versions converge almost surely towards (the same) nearly optimal value for the quantization criterion. These convergence results are similar in spirit to the most satisfactory almost sure convergence theorem for the CLVQ algorithm obtained by Pagès in [82].

For a given time span, our parallel DALVQ algorithm is able to process much more data than a single processor execution of the CLVQ procedure. Moreover, DALVQ is also asynchronous. This means that local algorithms do not have to wait at preset points for messages to become available. This allows some processors to compute faster and execute more iterations than others, and it also allows communication delays to be substantial and unpredictable. The communication channels are also allowed to deliver messages out of order, that is, in a different order than the one in which they were transmitted. Asynchronism can provide two major advantages. First, a reduction of the synchronization penalty, which could bring a speed advantage over a synchronous execution. Second, for potential industrialization, asynchronism has greater implementation flexibility. Tolerance to system failures and uncertainty can also be increased. As in the case with any on-line algorithm, DALVQ also deals with variable data loads over time. In fact, on-line algorithms avoid tremendous and non scalable batch requests on all data sets. Moreover, with an on-line algorithm, new data may enter the system and be taken into account while the algorithm is already running.

The paper is organized as follows. In Section 3.2 we review some standard facts on the clustering problem. We extract the relevant material from Pagès [82] without proof, thus making our exposition self-contained. In Section 3.3 we give a brief exposition of the mathematical framework for parallel asynchronous gradient methods introduced by Tsitsiklis et al. in [94] and Bertsekas and Tsitsiklis [94, 11]. The results of Blondel et al. [17] on the asymptotic consensus in asynchronous parallel averaging problems are also recalled. In Section 3.4, our main results are stated and proved.

3.2 Quantization and CLVQ algorithm

3.2.1 Overview

Let μ be a probability measure on $\mathbb{R}^d$ with finite second-order moment. The quantization problem consists in finding a "good approximation" of μ by a set of κ vectors of $\mathbb{R}^d$ called quantizer. Throughout the document the κ quantization points (or prototypes) will be seen as the components of a $\left(\mathbb{R}^d\right)^\kappa$-dimensional vector $w = (w_1, \ldots, w_\kappa)$. To measure the correctness of a quantization scheme

given by w, one introduces a cost function called distortion, defined by

$$C_\mu(w) = \frac{1}{2} \int_{\mathbb{R}^d} \min_{1 \leq \ell \leq \kappa} \|\mathbf{z} - w_\ell\|^2 \, d\mu(\mathbf{z}).$$

Under some minimal assumptions, the existence of an optimal quantizer vector $w^\circ \in \operatorname{argmin}_{w \in (\mathbb{R}^d)^\kappa} C_\mu(w)$ has been established by Pollard in [84] (see also Sabin and Gray [90, Appendix 2]).

In a statistical context, the distribution μ is only known through n independent random observations $\mathbf{z}_1, \ldots, \mathbf{z}_n$ drawn according to μ. Denote by μ_n the empirical distribution based on $\mathbf{z}_1, \ldots, \mathbf{z}_n$, that is, for every Borel subset A of $\mathbb{R}^d$

$$\mu_n(A) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{\mathbf{z}_i \in A\}}.$$

Much attention has been devoted to the convergence study of the quantization scheme provided by the empirical minimizers

$$w_n^\circ \in \operatorname*{argmin}_{w \in (\mathbb{R}^d)^\kappa} C_{\mu_n}(w). \tag{3.1}$$

The almost sure convergence of $C_\mu(w_n^\circ)$ towards $\min_{w \in (\mathbb{R}^d)^\kappa} C_\mu(w)$ was proved by Pollard in [86, 84] and Abaya and Wise in [2]. Rates of convergence and nonasymptotic performance bounds have been considered by Pollard [85], Chou [32], Linder et al. [72], Bartlett et al. [8], Linder [71, 70], Antos [1] and Antos et al. [6]. Convergence results have been established by Biau et al. in [14] where μ is a measure on a Hilbert space. It turns out that the minimization of the empirical distortion is a computationally hard problem. As shown by Inaba et al. in [62], the computational complexity of this minimization problem is exponential in the number of quantizers κ and the dimension of the data d. Therefore, exact computations are untractable for most of the practical applications.

Based on this, our goal in this document is to investigate effective methods that produce accurate quantizations with data samples. One of the most popular procedure is Lloyd's algorithm (see Lloyd [73]) sometimes refereed to as batch k-means. A convergence theorem for this algorithm is provided by Sabin and Gray in [90]. Another celebrated quantization algorithm is the Competitive Learning Vector Quantization (CLVQ), also called on-line k-means. The latter acronym outlines the fact that data arrive over time while the execution of the algorithm and their characteristics are unknown until their arrival times. The main difference between the CLVQ and the Lloyd's algorithm is that the latter run in batch training mode.

This means that the whole training set is presented before performing an update, whereas the CLVQ algorithm uses each item of the training sequence at each update.

The CLVQ procedure can be seen as a stochastic gradient descent algorithm. In the more general context of gradient descent methods, one cannot hope for the convergence of the procedure towards global minimizers with a non convex objective function (see for instance Benveniste et al. [9]). In our quantization context, the distortion mapping C_μ is not convex (see for instance Graf and Luschgy [49]). Thus, just as in Lloyd's method, the iterations provided by the CLVQ algorithm converge towards local minima of C_μ.

Assuming that the distribution μ has a compact support and a bounded density with respect to the Lebesgue measure, Pagès states in [82] a result regarding the almost sure consistency of the CLVQ algorithm towards critical points of the distortion C_μ. The author shows that the set of critical points necessarily contains the global and local optimal quantizers. The main difficulties in the proof arise from the fact that the gradient of the distortion is singular on κ-tuples having equal components and the distortion function C_μ is not convex. This explains why standard theories for stochastic gradient algorithm do not apply in this context.

3.2.2 The quantization problem, basic properties

In the sequel, we denote by $\mathcal{G}$ the closed convex hull of $\mathrm{supp}\,(\mu)$, where $\mathrm{supp}\,(\mu)$ stands for the support of the distribution. Observe that, with this notation, the distortion mapping is the function $C : \left(\mathbb{R}^d\right)^\kappa \longrightarrow [0,\infty)$ defined by

$$C(w) \triangleq \frac{1}{2}\int_{\mathcal{G}} \min_{1\leq \ell \leq \kappa} \|\mathbf{z} - w_\ell\|^2 \, d\mu(\mathbf{z}), \quad w = (w_1, \ldots, w_\kappa) \in \left(\mathbb{R}^d\right)^\kappa. \tag{3.2}$$

Throughout the document, with a slight abuse of notation, $\|.\|$ means both the Euclidean norm of $\mathbb{R}^d$ or $\left(\mathbb{R}^d\right)^\kappa$. In addition, the notation $\mathcal{D}_*^\kappa$ stands for the set of all vector of $\left(\mathbb{R}^d\right)^\kappa$ with pairwise distinct components, that is,

$$\mathcal{D}_*^\kappa \triangleq \left\{ w \in \left(\mathbb{R}^d\right)^\kappa \mid w_\ell \neq w_k \text{ if and only if } \ell \neq k \right\}.$$

Under some extra assumptions on μ, the distortion function can be rewritten using space partition set called Voronoï tessellation.

Definition 3.2.1. *Let $w \in \left(\mathbb{R}^d\right)^\kappa$, the Voronoï tessellation of $\mathcal{G}$ related to w is the family of open sets $\{W_\ell(w)\}_{1\leq \ell \leq \kappa}$ defined as follows:*

– *If* $w \in \mathcal{D}_*^\kappa$, *for all* $1 \leq \ell \leq \kappa$,

$$W_\ell(w) = \left\{ v \in \mathcal{G} \;\middle|\; \|w_\ell - v\| < \min_{k \neq \ell} \|w_k - v\| \right\}.$$

– *If* $w \in \left(\mathbb{R}^d\right)^\kappa \setminus \mathcal{D}_*^\kappa$, *for all* $1 \leq \ell \leq \kappa$,

– *if* $\ell = \min\{k \mid w_k = w_\ell\}$,

$$W_\ell(w) = \left\{ v \in \mathcal{G} \;\middle|\; \|w_\ell - v\| < \min_{w_k \neq w_\ell} \|w_k - v\| \right\}$$

– *otherwise,* $W_\ell(w) = \emptyset$.

As an illustration, Figure 3.1 shows Voronoï tessellations associated to a vector $w \in ([0,1] \times [0,1])^{50}$ whose components have been drawn independently and uniformly. This figure also highlights a remarkable property of the cell borders, which are portions of hyperplanes (see Graf and Luschgy [49]).

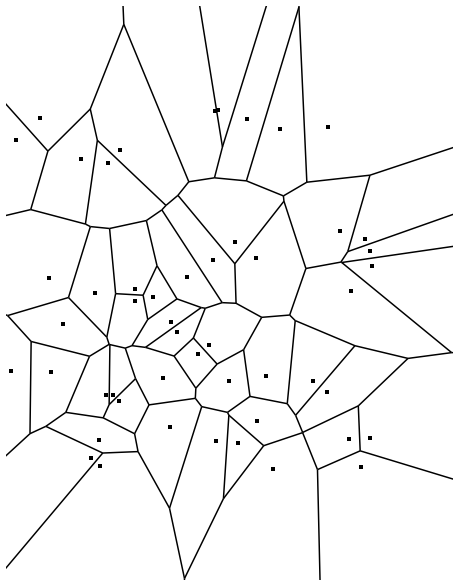

Figure 3.1: Voronoï tessellation of 50 points of $\mathbb{R}^2$ drawn uniformly in a square.

Observe that if $\mu(H)$ is zero for any hyperplane H of $\mathbb{R}^d$ (a property which is sometimes referred to as strong continuity) then using Definition 3.2.1, it is easy to see that the distortion takes the form:

$$C(w) = \frac{1}{2} \sum_{\ell=1}^{\kappa} \int_{W_\ell(w)} \|\mathbf{z} - w_\ell\|^2 \, d\mu(\mathbf{z}), \quad w \in \left(\mathbb{R}^d\right)^\kappa. \tag{3.3}$$

The following assumption will be needed throughout the paper. This assumption is similar to the peak power constraint (see Chou [32] and Linder [70]). Note that most of the results of this subsection are still valid if μ satisfies the weaker strong continuity property.

Assumption 3.2.1 (Compact Supported Density)**.** *The probability measure μ has a bounded density with respect to the Lebesgue measure on $\mathbb{R}^d$. Moreover, the support of μ is equal to its convex hull $\mathcal{G}$, which in turn, is compact.*

The next proposition states the differentiability of the distortion C, and provides an explicit formula for the gradient ∇C whenever the distortion is differentiable.

Proposition 3.2.1 (Pagès [82])**.** *Under Assumption 3.2.1, the distortion C is continuously differentiable at every $w = (w_1, \ldots, w_\kappa) \in \mathcal{D}_*^\kappa$. Furthermore, for all $1 \leq \ell \leq \kappa$,*

$$\nabla_\ell C(w) = \int_{W_\ell(w)} (w_\ell - \mathbf{z})\, d\mu(\mathbf{z}).$$

Some necessary conditions on the location of the minimizers of C can be derived from its differentiability properties. Therefore, Proposition 3.2.2 below states that the minimizers of C have parted components and that they are contained in the support of the density. Thus, the gradient is well defined and these minimizers are necessarily some zeroes of ∇C. For the sequel it is convenient to let $\overset{\circ}{A}$ be the interior of any subset A of $\left(\mathbb{R}^d\right)^\kappa$.

Proposition 3.2.2 (Pagès [82])**.** *Under Assumption 3.2.1, we have*

$$\operatorname*{argmin}_{w \in \left(\mathbb{R}^d\right)^\kappa} C(w) \subset \operatorname*{argminloc}_{w \in \mathcal{G}^\kappa} C(w) \subset \overset{\circ}{\mathcal{G}^\kappa} \cap \{\nabla C = 0\} \cap \mathcal{D}_*^\kappa,$$

where $\operatorname{argminloc}_{w \in \mathcal{G}^\kappa} C(w)$ *stands for the set of local minimizers of C over $\mathcal{G}^\kappa$.*

For any $\mathbf{z} \in \mathbb{R}^d$ and $w \in \left(\mathbb{R}^d\right)^\kappa$, let us define the following vector of $\left(\mathbb{R}^d\right)^\kappa$

$$H(\mathbf{z}, w) \triangleq \left((w_\ell - \mathbf{z})\, \mathbb{1}_{\{\mathbf{z} \in W_\ell(w)\}}\right)_{1 \leq \ell \leq \kappa}. \tag{3.4}$$

On $\mathcal{D}_*^\kappa$, the function H may be interpreted as an observation of the gradient. With this notation, Proposition 3.2.1 states that

$$\nabla C(w) = \int_{\mathcal{G}} H(\mathbf{z}, w) d\mu(\mathbf{z}), \quad w \in \mathcal{D}_*^\kappa. \tag{3.5}$$

Let $\complement A$ stands for the complementary in $\left(\mathbb{R}^d\right)^\kappa$ of a subset $A \subset \left(\mathbb{R}^d\right)^\kappa$. Clearly, for all $w \in \complement\mathcal{D}_*^\kappa$, the mapping $H(.,w)$ is integrable. Therefore, ∇C can be extended on $\left(\mathbb{R}^d\right)^\kappa$ *via* the formula

$$h(w) \triangleq \int_{\mathcal{G}} H(\mathbf{z},w)d\mu(\mathbf{z}), \quad w \in \left(\mathbb{R}^d\right)^\kappa. \tag{3.6}$$

Note however that the function h, which is sometimes called the average function of the algorithm, is not continuous.

Remark 3.2.1. *Under Assumption 3.2.1, a computation for all $w \in \mathcal{D}_*^\kappa$ of the Hessian matrix $\nabla^2 C(w)$ can be deduced from Theorem 4 of Fort and Pagès [43]. In fact, the formula established in this theorem is valid for cost functions which are more complex than C (they are associated to Kohonen Self Organizing Maps, see Kohonen [67] for more details). In Theorem 4, letting $\sigma(k) = \mathbb{1}_{\{k=0\}}$, provides the result for our distortion C. The resulting formula shows that h is singular on $\complement\mathcal{D}_*^\kappa$ and, consequently, that this function cannot be Lipschitz on $\mathcal{G}^\kappa$.*

3.2.3 Convergence of the CLVQ algorithm

The problem of finding a reliable clustering scheme for a dataset is equivalent to find optimal (or at least nearly optimal) minimizers for the mapping C. A minimization procedure by a usual gradient descent method cannot be implemented as long as ∇C is unknown. Thus, the gradient is approximated by a single example extracted from the data. This leads to the following stochastic gradient descent procedure

$$w(t+1) = w(t) - \varepsilon_{t+1} H\left(\mathbf{z}_{t+1}, w(t)\right), \quad t \geq 0, \tag{3.7}$$

where $w(0) \in \overset{\circ}{\mathcal{G}}{}^\kappa \cap \mathcal{D}_*^\kappa$ and $\mathbf{z}_1, \mathbf{z}_2 \ldots$ are independent observations distributed according to the probability measure μ.

The algorithm defined by the iterations (3.7) is known as the CLVQ algorithm in the data analysis community. It is also called the Kohonen Self Organizing Map algorithm with 0 neighbor (see for instance Kohonen [67]) or the on-line k-means procedure (see MacQueen [75] and Bottou [20]) in various fields related to statistics. As outlined by Pagès in [82], this algorithm belongs to the class of stochastic gradient descent methods. However, the almost sure convergence of this type of algorithm cannot be obtained by general tools such as Robbins-Monro method (see Robbins and Monro [87]) or the Kushner-Clark's Theorem (see Kushner and Clark [68]). Indeed, the main difficulty essentially arises from the non convexity of the function C, its non coercivity and the singularity of h at $\complement\mathcal{D}_*^\kappa$ (we refer the

reader to [82, Section 6] for more details).

The following assumption set is standard in a gradient descent context. It basically upraises constraints on the decreasing speed of the sequence of steps $\{\varepsilon_t\}_{t=0}^{\infty}$.

Assumption 3.2.2 (Decreasing steps)**.** *The $(0,1)$-valued sequence $\{\varepsilon_t\}_{t=0}^{\infty}$ satisfies the following two constraints:*

1. $\sum_{t=0}^{\infty} \varepsilon_t = \infty$.
2. $\sum_{t=0}^{\infty} \varepsilon_t^2 < \infty$.

An examination of identities (3.7) and (3.4) reveals that if $\mathbf{z}_{t+1} \in W_{\ell_0}(w(t))$, where $\ell_0 \in \{1, \ldots, M\}$ then

$$w_{\ell_0}(t+1) = (1 - \varepsilon_{t+1})\, w_{\ell_0}(t) + \varepsilon_{t+1}\mathbf{z}_{t+1}. \tag{3.8}$$

The component $w_{\ell_0}(t+1)$ can be viewed as the image of $w_{\ell_0}(t)$ by a $\mathbf{z}_{t+1}$-centered homothety with ratio $1 - \varepsilon_{t+1}$ (Figure 3.2 provides an illustration of this fact). Thus, under Assumptions 3.2.1 and 3.2.2, the trajectories of $\{w(t)\}_{t=0}^{\infty}$ stay in $\overset{\circ}{\mathcal{G}}{}^{\kappa} \cap \mathcal{D}_*^{\kappa}$. More precisely, if

$$w(0) \in \overset{\circ}{\mathcal{G}}{}^{\kappa} \cap \mathcal{D}_*^{\kappa}$$

then

$$w(t) \in \overset{\circ}{\mathcal{G}}{}^{\kappa} \cap \mathcal{D}_*^{\kappa}, \quad t \geq 0, \text{ a.s.}$$

Although ∇C is not continuous some regularity can be obtained. To this end, we need to introduce the following materials. For any $\delta > 0$ and any compact set $L \subset \mathbb{R}^d$, let the compact set $L_\delta^{\kappa} \subset \left(\mathbb{R}^d\right)^{\kappa}$ be defined as

$$L_\delta^{\kappa} \triangleq \left\{ w \in L^{\kappa} \mid \min_{k \neq \ell} \|w_\ell - w_k\| \geq \delta \right\}. \tag{3.9}$$

The next lemma that states on the regularity of ∇C will prove to be extremely useful in the proof of Theorem 3.2.2 and throughout Section 3.4.

Lemma 3.2.1 (Pagès [82])**.** *Assume that μ satisfies Assumption 3.2.1 and let L be a compact set of $\mathbb{R}^d$. Then, there is some constant P_δ such that for all w and v in L_δ^{κ} with $[w, v] \subset \mathcal{D}_*^{\kappa}$,*

$$\|\nabla C(w) - \nabla C(v)\| \leq P_\delta \|w - v\|.$$

The following lemma, called G-Lemma in [82] is an easy-to-apply convergence results on stochastic algorithms. It is particularly adapted to the present situation of the CLVQ algorithm where the average function of the algorithm h is singular.

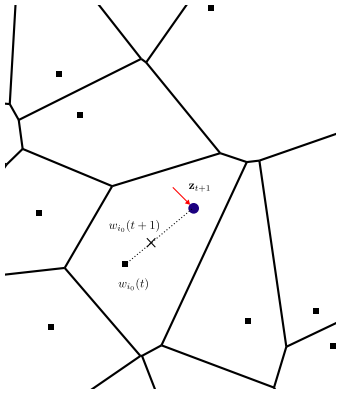

Figure 3.2: Drawing of a portion of a 2-dimensional Voronoï tessellation. For $t \geq 0$, if $\mathbf{z}_{t+1} \in W_{\ell_0}(w(t))$ then $w_\ell(t+1) = w_\ell(t)$ for all $\ell \neq \ell_0$ and $w_{\ell_0}(t+1)$ lies in the segment $[w_{\ell_0}(t), \mathbf{z}_{t+1}]$. The update of the vector $w_{\ell_0}(t)$ can also be viewed as a $\mathbf{z}_{t+1}$-centered homothety with ratio $1 - \varepsilon_{t+1}$.

Theorem 3.2.1 (G-Lemma, Fort and Pagès [44])**.** *Assume that the iterations (3.7) of the CLVQ algorithm satisfy the following conditions:*

1. $\sum_{t=1}^{\infty} \varepsilon_t = \infty$ *and* $\varepsilon_t \xrightarrow[t\to\infty]{} 0$.
2. *The sequences* $\{w(t)\}_{t=0}^{\infty}$ *and* $\{h(w(t))\}_{t=0}^{\infty}$ *are bounded a.s.*
3. *The series* $\sum_{t=0}^{\infty} \varepsilon_{t+1} \left(H(\mathbf{z}_{t+1}, w(t)) - h(w(t))\right)$ *converge a.s. in* $\left(\mathbb{R}^d\right)^\kappa$.
4. *There exists a lower semi-continuous function* $G : \left(\mathbb{R}^d\right)^\kappa \longrightarrow [0, \infty)$ *such that*
$$\sum_{t=0}^{\infty} \varepsilon_{t+1} G(w(t)) < \infty, \quad a.s.$$

Then, there exists a random connected component Ξ *of* $\{G = 0\}$ *such that*

$$\mathrm{dist}\left(w(t), \Xi\right) \xrightarrow[t\to\infty]{} 0, \quad a.s.,$$

where the symbol dist *denotes the usual distance function between a vector and a subset of* $\left(\mathbb{R}^d\right)^\kappa$. *Note also that if the connected components of* $\{G = 0\}$ *are singletons then there exists* $\xi \in \{G = 0\}$ *such that* $w(t) \xrightarrow[t\to\infty]{} \xi$ *a.s.*

For a definition of the topological concept of connected component, we refer the reader to Choquet [31]. The interest of the G-Lemma depends upon the choice of G. In our context, a suitable lower semi-continuous function is $\widehat{G}$ defined by

$$\widehat{G}(w) \triangleq \liminf_{v \in \mathcal{G}^\kappa \cap \mathcal{D}_*^\kappa,\, v \to w} \left\|\nabla C(v)\right\|^2, \qquad w \in \mathcal{G}^\kappa. \tag{3.10}$$

The next theorem is, as far as we know, the first almost sure convergence theorem for the stochastic algorithm CLVQ.

Theorem 3.2.2 (Pagès [82])**.** *Under Assumptions 3.2.1 and 3.2.2, conditioned on the event*

$$\left\{\liminf_{t\to\infty} \operatorname{dist}\left(w(t), \complement\mathcal{D}_*^\kappa\right) > 0\right\}, \quad \textit{one has}$$

$$\operatorname{dist}(w(t), \Xi_\infty) \xrightarrow[t\to\infty]{} 0, \quad a.s.,$$

where Ξ_∞ *is some random connected component of* $\{\nabla C = 0\}$.

The proof is an application of the above G-Lemma with the mapping $\widehat{G}$ defined by equation (3.10). Theorem 3.2.2 states that the iterations of the CLVQ necessarily converge towards some critical points (zeroes of ∇C). From Proposition 3.2.2 we deduce that the set of critical points necessarily contains optimal quantizers. Recall that without more assumption than $w(0) \in \overset{\circ}{\mathcal{G}}{}^\kappa \cap \mathcal{D}_*^\kappa$, we have already discussed the fact that the components of $w(t)$ are almost surely parted for all $t \geq 0$. Thus, it is easily seen that the two following events only differ on a set of zero probability

$$\left\{\liminf_{t\to\infty} \operatorname{dist}\left(w(t), \complement\mathcal{D}_*^\kappa\right) > 0\right\}$$

and

$$\left\{\inf_{t\geq 0} \operatorname{dist}\left(w(t), \complement\mathcal{D}_*^\kappa\right) > 0\right\}.$$

Some results are provided by Pagès in [82] for asymptotically stuck components but, as pointed out by the author, they are less satisfactory.

3.3 General distributed asynchronous algorithm

3.3.1 Model description

Let $s(t)$ be any $\left(\mathbb{R}^d\right)^\kappa$-valued vector and consider the following iterations on a vector $w \in \left(\mathbb{R}^d\right)^\kappa$

$$w(t+1) = w(t) + s(t), \quad t \geq 0. \tag{3.11}$$

Here, the model of discrete time described by iterations (3.11) can only be performed by a single computing entity. Therefore, if the computations of the vectors $s(t)$ are relatively time consuming then not many iterations can be achieved for a given time span. Consequently, a parallelization of this computing scheme should be investigated. The aim of this section is to discuss a precise mathematical description of a distributed asynchronous model for the iterations (3.11). This model

for distributed computing was originally proposed by Tsitsiklis et al. in [94] and was revisited in Bertsekas and Tsitsiklis [11, Section 7.7].

Assume that we dispose of a distributed architecture with M computing entities called processors (or agents, see for instance Bullo et al. [27]). Each processor is labeled, for simplicity of notation, by a natural number $i \in \{1, \ldots, M\}$. Throughout the paper, we will add the superscript i on the variables possessed by the processor i. In the model we have in mind, each processor has a buffer where its current version of the iterated vector is kept, i.e a local memory. Thus, for agent i such iterations are represented by the $\left(\mathbb{R}^d\right)^\kappa$-valued sequence $\{w^i(t)\}_{t=0}^{\infty}$.

Let $t \geq 0$ denote the current time. For any pair of processors $(i, j) \in \{1, \ldots, M\}^2$, the value kept by agent j and available for agent i at time t is not necessarily the most recent one, $w^j(t)$, but more probably and outdated one, $w^j(\tau^{i,j}(t))$, where the deterministic time instant $\tau^{i,j}(t)$ satisfy $0 \leq \tau^{i,j}(t) \leq t$. Thus, the difference $t - \tau^{i,j}(t)$ can be seen as a communication delay. This is a modeling of some aspects of the network: latency and bandwidth finiteness.

We insist on the fact that there is a distinction between "global" and "local" time. The time variable we refer above to as t corresponds to a global clock. Such a global clock is needed only for analysis purposes. The processors work without knowledge of this global clock. They have access to a local clock or to no clock at all.

The algorithm is initialized at $t = 0$, where each processor $i \in \{1, \ldots, M\}$ has an initial version $w^i(0) \in \left(\mathbb{R}^d\right)^\kappa$ in its buffer. We define the general distributed asynchronous algorithm by the following iterations

$$w^i(t+1) = \sum_{j=1}^{M} a^{i,j}(t) w^j(\tau^{i,j}(t)) + s^i(t), \quad i \in \{1, \ldots, M\} \text{ and } t \geq 0. \tag{3.12}$$

The model can be interpreted as follows: at time $t \geq 0$, processor i receives messages from other processors containing $w^j(\tau^{i,j}(t))$. Processor i incorporates these new vectors by forming a convex combination and incorporates the vector $s^i(t)$ resulting from its own "local" computations. The coefficients $a^{i,j}(t)$ are nonnegative numbers which satisfy the constraint

$$\sum_{j=1}^{M} a^{i,j}(t) = 1, \quad i \in \{1, \ldots, M\} \text{ and } t \geq 0. \tag{3.13}$$

As the combining coefficients $a^{i,j}(t)$ depend on t, the network communication topology is sometimes referred to as time-varying. The sequences $\{\tau^{i,j}(t)\}_{t=0}^{\infty}$ need

not to be known in advance by any processor. In fact, their knowledge is not required to execute iterations defined by equation (3.12). Thus, we do not necessary dispose of a shared global clock or synchronized local clocks at the processors.

As for now the descent terms $\{s^i(t)\}_{t=0}^{\infty}$ will be arbitrary $\left(\mathbb{R}^d\right)^{\kappa}$-valued sequences. In Section 3.4, when we define the Distributed Asynchronous Learning Vector Quantization (DALVQ), the definition of the descent terms will be made more explicit.

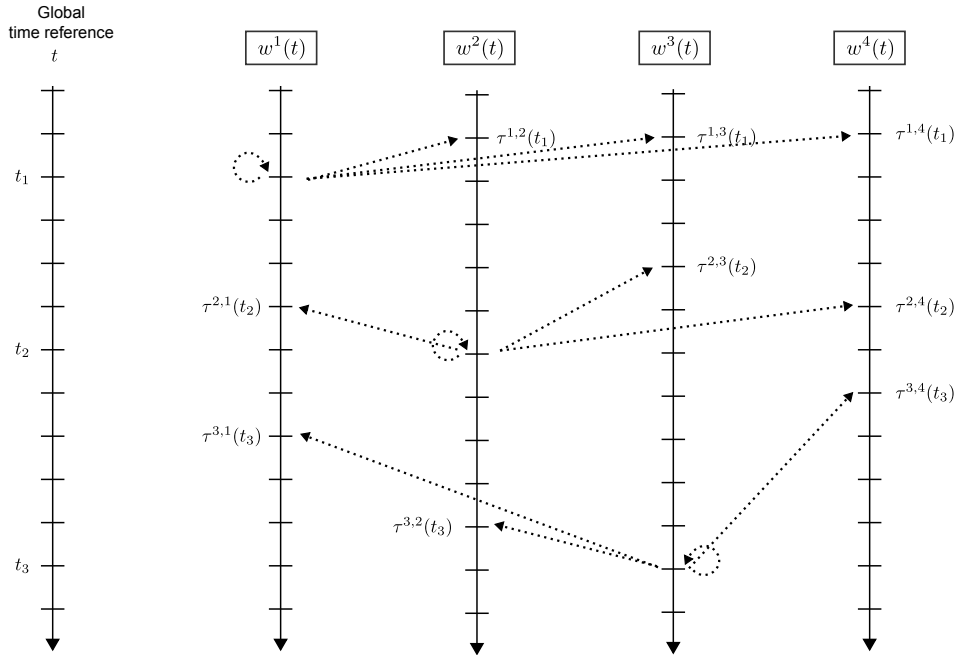

Figure 3.3: Illustration of the time delays introduced in the general distributed asynchronous algorithm. Here, there are $M = 4$ different processors with their own computations of the vectors $w^{(i)}$, $i \in \{1, 2, 3, 4\}$. Three arbitrary values of the global time t are represented (t_1, t_2 and t_3), with $\tau^{i,i}(t_k) = t_k$ for all $i \in \{1, 2, 3, 4\}$ and $1 \leq k \leq 3$. The dashed arrows head towards the versions available at time t_k for an agent $i \in \{1, 2, 3, 4\}$ represented by the tail of the arrow.

3.3.2 The agreement algorithm

This subsection is devoted to a short survey of the results, found by Blondel et al. in [17], for a natural simplification of the general distributed asynchronous

algorithm (3.12). This simplification is called agreement algorithm by Blondel et al. and is defined by

$$x^i(t+1) = \sum_{j=1}^{M} a^{i,j}(t) x^j(\tau^{i,j}(t)), \quad i \in \{1, \dots, M\} \text{ and } t \geq 0. \tag{3.14}$$

where $x^i(0) \in \left(\mathbb{R}^d\right)^\kappa$. An observation of these equations reveals that they are similar to iterations (3.12), the only difference being that all descent terms equal 0.

In order to analyse the convergence of the agreement algorithm (3.14), Blondel et al in [17] define two sets of assumptions that enforce some weak properties on the communication delays and the network topology. As shown in [17], if the assumptions contained in one of these two set hold, then the distributed versions of the agreement algorithm, namely the x^i, reach an asymptotical consensus. This latter statement means that there exists a vector $x^\star$ (independent of i) such that

$$x^i(t) \xrightarrow[t\to\infty]{} x^\star, \quad i \in \{1, \dots, M\}.$$

The agreement algorithm (3.14) is essentially driven by the communication times $\tau^{i,j}(t)$ assumed to be deterministic but do not need to be known *a priori* by the processors. The following Assumption 3.3.1 essentially ensures, in its third statement, that the communication delays $t - \tau^{i,j}(t)$ are bounded. This assumption prevents some processor from taking into account some arbitrarily old values computed by others processors. Assumption 3.3.1 1. is just a convention: when $a^{i,j}(t) = 0$ the value $\tau^{i,j}(t)$ has no effect on the update. Assumption 3.3.1 2. is rather natural because processors have access to their own most recent value.

Assumption 3.3.1 (Bounded communication delays)**.**

1. *If* $a^{i,j}(t) = 0$ *then* $\tau^{i,j}(t) = t, \quad (i,j) \in \{1, \dots, M\}^2$ *and* $t \geq 0$,
2. $\tau^{i,i}(t) = t, \quad i \in \{1, \dots, M\}$ *and* $t \geq 0$.
3. *There exists a positive integer* B_1 *such that*

$$t - B_1 < \tau^{i,j}(t) \leq t, \quad (i,j) \in \{1, \dots, M\}^2 \text{ and } t \geq 0.$$

The next Assumption 3.3.2 states that the value possessed by agent i at time $t+1$, namely $x^i(t+1)$, is a weighted average of its own value and the values that it has just received from other agents.

Assumption 3.3.2 (Convex combination and threshold)**.** *There exists a positive constant* $\alpha > 0$ *such that the following three properties hold:*

1. $a^{i,i}(t) \geq \alpha, \quad i \in \{1, \ldots, M\}$ *and* $t \geq 0$.
2. $a^{i,j}(t) \in \{0\} \cup [\alpha, 1], \quad (i,j) \in \{1, \ldots, M\}^2$ *and* $t \geq 0$.
3. $\sum_{j=1}^{M} a^{i,j}(t) = 1, \quad i \in \{1, \ldots, M\}$ *and* $t \geq 0$.

Let us mention one particular relevant case for the choice of the combining coefficients $a^{i,j}(t)$. Let $i \in \{1, \ldots, M\}$ and $t \geq 0$, the set

$$N^i(t) \triangleq \left\{ j \in \{1, \ldots, M\} \in \{1, \ldots, M\} \mid a^{i,j}(t) \neq 0 \right\}$$

corresponds to the set of agents whose version is taken into account by processor i at time t. For all $(i,j) \in \{1, \ldots, M\}^2$ and $t \geq 0$, the weights $a^{i,j}(t)$ are defined by

$$a^{i,j}(t) = \begin{cases} 1/\#N^i(t) & \text{if } j \in N^i(t); \\ 0 & \text{otherwise;} \end{cases}$$

where $\#A$ denotes the cardinal of any finite set A. The above definition on the combining coefficients appears to be relevant for practical implementations of the model DALVQ introduced in Section 3.4. For a discussion on others special interest cases regarding the choices of the coefficients $a^{i,j}(t)$ we refer the reader to [17].

The communication patterns, sometimes refereed to as the network communication topology, can be expressed in terms of directed graph. For a thorough introduction to graph theory, see Jungnickel [63].

Definition 3.3.1 (Communication graph)**.** *Let us fix $t \geq 0$, the communication graph at time t, $(\mathcal{V}, E(t))$, is defined by*

- *the set of vertices $\mathcal{V}$ is formed by the set of processors $\mathcal{V} = \{1, \ldots, M\}$,*
- *the set of edges $E(t)$ is defined via the relationship*

$$(j,i) \in E(t) \text{ if and only if } a^{i,j}(t) > 0.$$

Assumption 3.3.3 is a minimal condition required for a consensus among the processors. More precisely, it states that for any pair of agents $(i,j) \in \{1, \ldots, M\}^2$ there is a sequence of communications where the values computed by agent i will influence (directly or indirectly) the future values kept by agent j.

Assumption 3.3.3 (Graph connectivity)**.** *The graph $(\mathcal{V}, \cup_{s \geq t} E(s))$ is strongly connected for all $t \geq 0$.*

Finally, we define two supplementary assumptions. The combination of one of the two following assumptions with the three previous ones will ensure the convergence of the agreement algorithm. As mentioned above, if Assumption 3.3.3 holds then there is a communication path between any pair of agents. Assumption 3.3.4 below expresses the fact that there is a finite upper bound for the length of such paths.

Assumption 3.3.4 (Bounded communication intervals)**.** *If i communicates with j an infinite number of times then there is a positive integer B_2 such that*

$$(i,j) \in E(t) \cup E(t+1) \cup \ldots \cup E(t+B_2-1), \quad t \geq 0.$$

Assumption 3.3.5 is a symmetry condition: if agent $i \in \{1,\ldots,M\}$ communicates with agent $j \in \{1,\ldots,M\}$ then j has communicated or will communicate with i during the time interval $(t-B_3, t+B_3)$ where $B_3 > 0$.

Assumption 3.3.5 (Symmetry)**.** *There exists some $B_3 > 0$ such that whenever $(i,j) \in E(t)$, there exists some τ that satisfies $|t-\tau| < B_3$ and $(j,i) \in E(\tau)$.*

To shorten a little bit the notation, we set

$$(\mathbf{AsY})_{\mathbf{1}} \equiv \begin{cases} \text{Assumption 3.3.1;} \\ \text{Assumption 3.3.2;} \\ \text{Assumption 3.3.3;} \\ \text{Assumption 3.3.4.} \end{cases} \qquad (\mathbf{AsY})_{\mathbf{2}} \equiv \begin{cases} \text{Assumption 3.3.1;} \\ \text{Assumption 3.3.2;} \\ \text{Assumption 3.3.3;} \\ \text{Assumption 3.3.5;} \end{cases}$$

We are now in a position to state the main result of this section. The Theorem 3.3.1 expresses the fact that, for the agreement algorithm, a consensus is asymptotically reached by the agents.

Theorem 3.3.1 (Blondel et al. [17])**.** *Under the set of Assumptions $(\mathbf{AsY})_{\mathbf{1}}$ or $(\mathbf{AsY})_{\mathbf{2}}$, there is a consensus vector $x^\star \in \left(\mathbb{R}^d\right)^\kappa$ (independent of i) such that*

$$\lim_{t\to\infty} \left\| x^i(t) - x^\star \right\| = 0, \quad i \in \{1,\ldots,M\}.$$

Besides, there exist $\rho \in [0,1)$ and $L > 0$ such that

$$\left\| x^i(t) - x^i(\tau) \right\| \leq L\rho^{t-\tau}, \quad i \in \{1,\ldots,M\} \text{ and } t \geq \tau \geq 0.$$

3.3.3 Asymptotic consensus

This subsection is devoted to the analysis of the general distributed asynchronous algorithm (3.12). For this purpose, the study of the agreement algorithm defined by equations (3.14) will be extremely fruitful. The following lemma states that the version possessed by agent $i \in \{1,\ldots,M\}$ at time $t \geq 0$, namely $w^i(t)$, depends linearly on the others initialization vectors $w^j(0)$ and the descent subsequences $\{s^j(\tau)\}_{\tau=-1}^{t-1}$, where $j \in \{1,\ldots,M\}$.

Lemma 3.3.1 (Tsitsiklis [93])**.** *For all $(i,j) \in \{1,\ldots,M\}^2$ and $t \geq 0$, there exists a real-valued sequence $\{\phi^{i,j}(t,\tau)\}_{\tau=-1}^{t-1}$ such that*

$$w^i(t) = \sum_{j=1}^{M} \phi^{i,j}(t,-1)\, w^j(0) + \sum_{\tau=0}^{t-1} \sum_{j=1}^{M} \phi^{i,j}(t,\tau)\, s^j(\tau).$$

For all $(i,j) \in \{1,\dots,M\}^2$ and $t \geq 0$, the real-valued sequences $\{\phi^{i,j}(t,\tau)\}_{\tau=-1}^{t-1}$ do not depend on the value taken by the descent terms $s^i(t)$. The real numbers $\phi^{i,j}(t,\tau)$ are determined by the sequences $\{\tau^{i,j}(\tau)\}_{\tau=0}^{t}$ and $\{a^{i,j}(\tau)\}_{\tau=0}^{t}$ which do not depend on w. These last sequences are unknown in general, but some useful qualitative properties can be derived, as expressed in Lemma 3.3.2 below.

Lemma 3.3.2 (Tsitsiklis [93]). *For all $(i,j) \in \{1,\dots,M\}^2$, let $\{\phi^{i,j}(t,\tau)\}_{\tau=-1}^{t-1}$ be the sequences defined in Lemma 3.3.1.*

1. *Under Assumption 3.3.2,*
$$0 \leq \phi^{i,j}(t,\tau) \leq 1, \quad (i,j) \in \{1,\dots,M\}^2 \text{ and } t > \tau \geq -1.$$
2. *Under Assumptions $(\mathbf{AsY})_\mathbf{1}$ or $(\mathbf{AsY})_\mathbf{2}$, we have:*
   a) *For all $(i,j) \in \{1,\dots,M\}^2$ and $\tau \geq -1$, the limit of $\phi^{i,j}(t,\tau)$ as t tends to infinity exists and is independent of j. It will be denoted $\phi^i(\tau)$.*
   b) *There exists some $\eta > 0$ such that*
$$\phi^i(\tau) > \eta, \quad i \in \{1,\dots,M\} \text{ and } \tau \geq -1.$$
   c) *There exist a constant $A > 0$ and $\rho \in (0,1)$ such that*
$$\left|\phi^{i,j}(t,\tau) - \phi^i(\tau)\right| \leq A\rho^{t-\tau}, \quad (i,j) \in \{1,\dots,M\}^2 \text{ and } t > \tau \geq -1.$$

Take $t' \geq 0$ and assume that the agents stop performing update after time t', but keep communicating and merging the results. This means that $s^j(t) = 0$ for all $t \geq t'$. Then, equations (3.12) write

$$w^i(t+1) = \sum_{j=1}^{M} a^{i,j}(t) w^j\left(\tau^{i,j}(t)\right), \quad i \in \{1,\dots,M\} \text{ and } t \geq t'.$$

If Assumptions $(\mathbf{AsY})_\mathbf{1}$ or $(\mathbf{AsY})_\mathbf{2}$ are satisfied then Theorem 3.3.1 shows that there is a consensus vector, depending on the time instant t'. This vector will be equal to $w^\star(t')$ defined below (see Figure 3.4). Lemma 3.3.2 provides a good way to define the sequence $\{w^\star(t)\}_{t=0}^{\infty}$ as shown in Definition 3.14. Note that this definition does not involve any assumption on the descent terms.

Definition 3.3.2 (Agreement vector). *Assume that Assumptions $(\mathbf{AsY})_\mathbf{1}$ or $(\mathbf{AsY})_\mathbf{2}$ are satisfied. The agreement vector sequence $\{w^\star(t)\}_{t=0}^{\infty}$ is defined by*

$$w^\star(t) \triangleq \sum_{j=1}^{M} \phi^j(-1)\, w^j(0) + \sum_{\tau=0}^{t-1} \sum_{j=1}^{M} \phi^j(\tau)\, s^j(\tau), \quad t \geq 0.$$

It is noteworthy that the agreement vector sequence $w^\star$ satisfies the following recursion formula

$$w^\star(t+1) = w^\star(t) + \sum_{j=1}^{M} \phi^j(t) s^j(t), \quad t \geq 0. \tag{3.15}$$

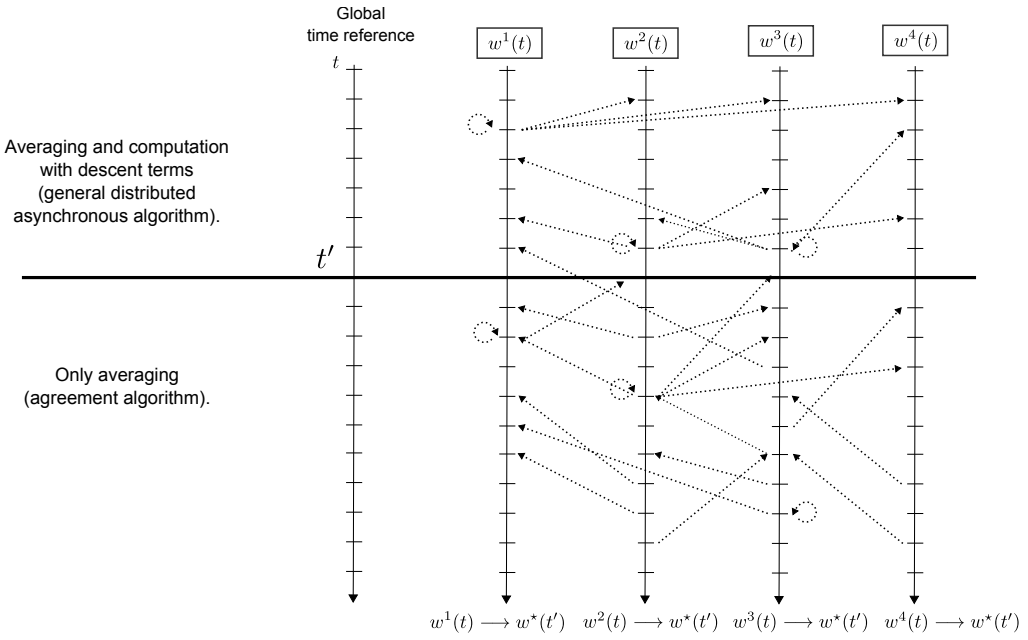

Figure 3.4: The agreement vector at time t', $w^\star(t')$ corresponds to the common value asymptotically achieved by all processors if computations integrating descent terms have stopped after t', i.e, $s^j(t) = 0$ for all $t \geq t'$.

3.4 Distributed asynchronous learning vector quantization

3.4.1 Introduction, model presentation

From now on, and until the end of the paper, we assume that one of the two set of assumptions $(\mathbf{AsY})_\mathbf{1}$ or $(\mathbf{AsY})_\mathbf{2}$ holds, as well as the compact-supported density Assumption 3.2.1. In addition, we will also assume that $0 \in \mathcal{G}$. For the sake of clarity, all the proofs of the main theorems as well as the lemmas needed for these proofs have been postponed at the end of the paper, in Annex.

Tsitsiklis in [93], Tsitsiklis et al in [94] and Bertsekas and Tsitsiklis in [11] studied distributed asynchronous stochastic gradient optimization algorithms. In this series of publications, for the distributed minimization of a cost function $F : \left(\mathbb{R}^d\right)^\kappa \longrightarrow \mathbb{R}$, the authors considered the general distributed asynchronous algorithm defined by equation (3.12) with specific choices for stochastic descent

terms s^i. Using the notation of Section 3.3, the algorithm writes

$$w^i(t+1) = \sum_{j=1}^{M} a^{i,j}(t) w^j(\tau^{i,j}(t)) + s^i(t), \quad i \in \{1, \ldots, M\} \text{ and } t \geq 0,$$

with stochastic descent terms $s^i(t)$ satisfying

$$\mathbb{E}\left\{ s^i(t) \,\middle|\, s^j(\tau),\ j \in \{1, \ldots, M\} \text{ and } t > \tau \geq 0 \right\} = -\varepsilon_{t+1}^i \nabla F\left(w^i(t)\right), \quad i \in \{1, \ldots, M\} \text{ and } t \geq 0. \tag{3.16}$$

where $\{\varepsilon_t^i\}_{t=0}^{\infty}$ are decreasing steps sequences. The definition of the descent terms in [11, 94] is more general than the one appearing in equation (3.16). We refer the reader to Assumption 3.2 and 3.3 in [94] and Assumption 8.2 in [11] for the precise definition of the descent terms in [11, 94]. As discussed in Section 3.2, the CLVQ algorithm is also a stochastic gradient descent procedure. Unfortunately, the results from Tsitisklis et al. in [94, 93, 11] do not apply with our distortion function, C, since the authors assume that F is continuously differentiable and ∇F is Lipschitz. Therefore, the aim of this section is to extend the results of Tsitsiklis et al. to the context of vector quantization and on-line clustering.

We first introduce the Distributed Asynchronous Learning Vector Quantization (DALVQ) algorithm. To prove its almost sure consistency, we will need an Asynchronous G-Lemma, which is inspired from the G-Lemma, Theorem 3.2.1, presented in Section 3.2. This theorem may be seen as an easy-to-apply tool for the almost sure consistency of a distributed asynchronous system where the average function is not necessary regular. Our approach sheds also some new light on the convergence of distributed asynchronous stochastic gradient descent algorithms. Precisely, Proposition 8.1 in [94] claims that $\liminf_{t\to\infty} \|\nabla F(w^i(t))\| = 0$ while our main Theorem 3.4.2 below states that $\lim_{t\to\infty} \|\nabla C(w^i(t))\| = 0$. However, there is a price to pay for this more precise result with the non Lipschitz gradient ∇C. Similarly to Pagès [82], who assumes that the trajectory of the CLVQ algorithm has almost surely asymptotically parted components (see Theorem 3.2.2 in Section 3.2), we will suppose that the agreement vector sequence has, almost surely, asymptotically parted component trajectories.

Recall that the goal of the DALVQ is to provide a well designed distributed algorithm that processes quickly (in term of wall clock time) very large data sets to produce accurate quantization. The data sets (or streams of data) are distributed among several queues sending data to the different processors of our distributed

framework. Thus, in this context the sequence $\mathbf{z}_1^i, \mathbf{z}_2^i, \ldots$ stands for the data available for processor, where $i \in \{1, \ldots, M\}$. The random variables

$$\mathbf{z}_1^1, \mathbf{z}_2^1, \ldots, \mathbf{z}_1^2, \mathbf{z}_2^2, \ldots$$

are assumed to be independent and identically distributed according to μ.

In the definition of the CLVQ procedure (3.7), the term $H\left(\mathbf{z}_{t+1}, w(t)\right)$ can be seen as an observation of the gradient $\nabla C\left(w(t)\right)$. Therefore, in our DALVQ algorithm, each processor $i \in \{1, \ldots, M\}$ is able to compute such observations using its own data $\mathbf{z}_1^i, \mathbf{z}_2^i, \ldots$. Thus, the DALVQ procedure is defined by equation (3.12) with the following choice for the descent term s^i:

$$s^i(t) = \begin{cases} -\varepsilon_{t+1}^i H\left(\mathbf{z}_{t+1}^i, w^i(t)\right) & \text{if } t \in T^i; \\ 0 & \text{otherwise;} \end{cases} \tag{3.17}$$

where $\{\varepsilon_t^i\}_{t=0}^{\infty}$ are $(0,1)$-valued sequences. The sets T^i contain the time instants where the version w^i, kept by processor i, is updated with the descent terms. This fine grain description of the algorithm allows some processors to be idle for computing descent terms (when $t \notin T^i$). This reflects the fact that the computing operations might not take the same time for all processors, which is precisely the core of asynchronous algorithms analysis. Similarly to time delays and combining coefficients, the sets T^i are supposed to be deterministic but do not need to be known *a priori* for the execution of the algorithm.

In the DALVQ model, randomness arises from the data $\mathbf{z}$. Therefore, it is natural to let $\{\mathcal{F}_t\}_{t=0}^{\infty}$ be the filtration built on the σ-algebras

$$\mathcal{F}_t \triangleq \sigma\left(\mathbf{z}_s^i,\ i \in \{1, \ldots, M\} \text{ and } t \geq s \geq 0\right), \quad t \geq 0.$$

An easy verification shows that, for all $j \in \{1, \ldots, M\}$ and $t \geq 0$, $w^\star(t)$ and $w^j(t)$ are $\mathcal{F}_t$-measurable random variables.

For simplicity, the assumption on the decreasing speed of the sequences $\{\varepsilon_t^i\}_{t=0}^{\infty}$ is strengthened as follows. The notation $a \vee b$ stands for the maximum of two reals a and b.

Assumption 3.4.1. *There exist two real numbers $K_1 > 0$ and $K_2 \geq 1$ such that*

$$\frac{K_1}{t \vee 1} \leq \varepsilon_{t+1}^i \leq \frac{K_2}{t \vee 1}, \quad i \in \{1, \ldots, M\} \text{ and } t \geq 0.$$

If Assumption 3.4.1 holds then the sequences $\{\varepsilon_t^i\}_{t=0}^{\infty}$ satisfy the standard Assumption 3.2.2 for stochastic optimization algorithms. Note that the choice of steps proportional to $1/t$ has been proved to be a satisfactory learning rate, theoretically speaking and also for practical implementations (see for instance Murata [80] and Bottou and LeCun [22]).

For practical implementation, the sequences $\left\{\varepsilon_{t+1}^i\right\}_{t=0}^{\infty}$ satisfying Assumption 3.4.1 can be implemented without a global clock, that is, without assuming that the current value of t is known by the agents. This assumption is satisfied, for example, by taking the current value of ε_t^i proportional to $1/n_t^i$, where n_t^i is the number of times that processor i as performed an update, i.e., the cardinal of the set $T^i \cap \{0, \ldots, t\}$. For a given processor, if the time span between consecutive updates is bounded from above and from below, a straightforward examination shows that the sequence of steps satisfy Assumption 3.4.1.

Finally, the next assumption is essentially technical in nature. It enables to avoid time instants where all processors are idle. It basically requires that, at any time $t \geq 0$, there is at least one processor $i \in \{1, \ldots, M\}$ satisfying $s^i(t) \neq 0$.

Assumption 3.4.2. *One has* $\sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \geq 1$ *for all* $t \geq 0$.

3.4.2 The asynchronous G-Lemma

The aim of this subsection is to state a useful theorem similar to Theorem 3.2.1, but adapted to our asynchronous distributed context. The precise Definition 3.3.2 of the agreement vector sequence should not cast aside the intuitive definition. The reader should keep in mind that the vector $w^\star(t)$ is also the asymptotical consensus if descent terms are zero after time t. Consequently, even if the agreement vector $\{w^\star(t)\}_{t=0}^{\infty}$ is adapted to the filtration $\{\mathcal{F}_t\}_{t=0}^{\infty}$, the vector $w^\star(t)$ cannot be accessible for a user at time t. Nevertheless, the agreement vector $w^\star(t)$ can be interpreted as a "probabilistic state" of the whole distributed quantization scheme at time t. This explains why the agreement vector is a such convenient tool for the analysis of the DALVQ convergence and will be central in our adaptation of G-Lemma, Theorem 3.4.1.

Let us remark that equation (3.15), writes for all $t \geq 0$,

$$\begin{aligned} w^\star(t+1) &= w^\star(t) + \sum_{j=1}^{M} \phi^j(t) s^j(t) \\ &= w^\star(t) - \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j H\left(\mathbf{z}_{t+1}^j, w^j(t)\right). \end{aligned}$$

We recall the reader that the $[0,1]$-valued functions ϕ^j's are defined in Lemma 3.3.1.

Using the function h defined by identity (3.6) and the fact that the random variables $w^\star(t)$ and $w^j(t)$ are $\mathcal{F}_t$-measurable then it holds

$$h(w^\star(t)) = \mathbb{E}\left\{H\left(\mathbf{z}, w^\star(t)\right) \mid \mathcal{F}_t\right\}, \quad t \geq 0.$$

and

$$h(w^j(t)) = \mathbb{E}\left\{H\left(\mathbf{z}, w^j(t)\right) \mid \mathcal{F}_t\right\}, \quad j \in \{1, \ldots, M\} \text{ and } t \geq 0.$$

where $\mathbf{z}$ is a random variable of law μ independent of $\mathcal{F}_t$.

For all $t \geq 0$, set

$$\varepsilon_{t+1}^\star \triangleq \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j. \tag{3.18}$$

Clearly, the real numbers $\varepsilon_t^\star$ are nonnegative. Their strictly positiveness will be discussed in Proposition 3.4.1.

Set

$$\Delta M_t^{(1)} \triangleq \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j \left(h(w^\star(t)) - h(w^j(t))\right), \quad t \geq 0, \tag{3.19}$$

and

$$\Delta M_t^{(2)} \triangleq \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j \left(h(w^j(t)) - H\left(\mathbf{z}_{t+1}^j, w^j(t)\right)\right), \quad t \geq 0. \tag{3.20}$$

Note that $\mathbb{E}\left\{\Delta M_t^{(2)}\right\} = 0$ and, consequently, that the random variables $\Delta M_t^{(2)}$ can be seen as the increments of a martingale with respect to the filtration $\{\mathcal{F}_t\}_{t=0}^\infty$.

Finally, with this notation, equation (3.15) takes the form

$$w^\star(t+1) = w^\star(t) - \varepsilon_{t+1}^\star h(w^\star(t)) + \Delta M_t^{(1)} + \Delta M_t^{(2)}, \quad t \geq 0. \tag{3.21}$$

We are now in a position to state our most useful tool, which is similar in spirit to the G-Lemma, but adapted to the context of distributed asynchronous stochastic gradient descent algorithm.

Theorem 3.4.1 (Asynchronous G-Lemma)**.** *Assume that* $(\mathbf{AsY})_\mathbf{1}$ *or* $(\mathbf{AsY})_\mathbf{2}$ *and Assumption 3.2.1 hold and that the following conditions are satisfied:*

1. $\sum_{t=0}^\infty \varepsilon_t^\star = \infty$ *and* $\varepsilon_t^\star \xrightarrow[t\to\infty]{} 0$.
2. *The sequences* $\{w^\star(t)\}_{t=0}^\infty$ *and* $\{h(w^\star(t))\}_{t=0}^\infty$ *are bounded a.s.*

3. *The series* $\sum_{t=0}^{\infty} \Delta M_t^{(1)}$ *and* $\sum_{t=0}^{\infty} \Delta M_t^{(2)}$ *converge a.s. in* $\left(\mathbb{R}^d\right)^\kappa$.

4. *There exists a lower semi-continuous function* $G : \left(\mathbb{R}^d\right)^\kappa \longrightarrow [0, \infty)$ *such that*

$$\sum_{t=0}^{\infty} \varepsilon_{t+1}^\star G\left(w^\star(t)\right) < \infty, \quad a.s.$$

Then, there exists a random connected component Ξ *of* $\{G = 0\}$ *such that*

$$\operatorname{dist}\left(w^\star(t), \Xi\right) \xrightarrow[t\to\infty]{} 0, \quad a.s.$$

3.4.3 Trajectory analysis

The Pagès's proof in [82] on the almost sure convergence of the CLVQ procedure required a careful examination of the trajectories of the process $\{w(t)\}_{t=0}^{\infty}$. Thus, in this subsection we investigate similar properties and introduce the assumptions that will be needed to prove our main convergence result, Theorem 3.4.2.

The next Assumption 3.4.3 ensures that, for each processor, the quantizers stay in the support of the density.

Assumption 3.4.3. *One has*

$$\mathbb{P}\left\{w^j(t) \in \mathcal{G}^\kappa\right\} = 1, \quad j \in \{1, \ldots, M\} \text{ and } t \geq 0.$$

Firstly, let us mention that since the set $\mathcal{G}^\kappa$ is convex, if Assumption 3.4.3 holds then

$$\mathbb{P}\left\{w^\star(t) \in \mathcal{G}^\kappa\right\} = 1, \quad t \geq 0.$$

Secondly, note that the Assumption 3.4.3 is not particularly restrictive. This assumption is satisfied under the condition: for each processor, no descent term is added while a combining computation is performed. This writes

$$a_{i,j}(t) = \delta_{i,j} \text{ and } \tau^{i,i}(t) = t, \quad (i,j) \in \{1, \ldots, M\}^2 \text{ and } t \in T^i.$$

This requirement makes sense for practical implementations.

Recall that if $t \notin T^i$, then $s^i(t) = 0$. Thus, equation (3.12) takes the form

$$w^i(t+1) = \begin{cases} w^i(t+1) &= w^i(t) - \varepsilon_{t+1}^i \left(w^i(t) - \mathbf{z}_{t+1}^i\right) \\ &= \left(1 - \varepsilon_{t+1}^i\right) w^i(t) + \varepsilon_{t+1}^i \mathbf{z}_{t+1}^i & \text{if } t \in T^i; \\ w^i(t+1) &= \sum_{j=1}^M a^{i,j}(t) w^j(\tau^{i,j}(t)) & \text{otherwise.} \end{cases} \tag{3.22}$$

Since $\mathcal{G}^\kappa$ is a convex set, it follows easily that if $w^j(0) \in \mathcal{G}^\kappa$, then $w^j(t) \in \mathcal{G}^\kappa$ for all $j \in \{1, \ldots, M\}$ and $t \geq 0$ and, consequently, that Assumption 3.4.3 holds.

The next Lemma 3.4.1 provides a deterministic upper bound on the differences between the distributed versions w^i and the agreement vector. For any subset A of $\left(\mathbb{R}^d\right)^\kappa$, the notation $\mathrm{diam}(A)$ stands for the usual diameter defined by

$$\mathrm{diam}(A) = \sup_{x,y \in A} \{\|x - y\|\}.$$

Lemma 3.4.1. *Assume* $(\mathbf{AsY})_\mathbf{1}$ *or* $(\mathbf{AsY})_\mathbf{2}$ *holds and that Assumptions 3.2.1, 3.4.1 and 3.4.3 are satisfied then*

$$\|w^\star(t) - w^i(t)\| \leq \sqrt{\kappa} M \,\mathrm{diam}(\mathcal{G}) A K_2 \theta_t, \quad i \in \{1, \ldots, M\} \text{ and } t \geq 0, \text{ a.s.},$$

where $\theta_t \triangleq \sum_{\tau=-1}^{t-1} \frac{1}{\tau \vee 1} \rho^{t-\tau}$, *$A$ and ρ are the constants introduced in Lemma 3.3.2, K_2 is defined in Assumption 3.4.1.*

The sequence $\{\theta_t\}_{t=0}^\infty$ defined in Lemma 3.4.1 satisfies

$$\theta_t \xrightarrow[t\to\infty]{} 0 \text{ and } \sum_{t=0}^{\infty} \frac{\theta_t}{t} < \infty. \tag{3.23}$$

We give some calculations justifying the statements at the end of the Annex.

Thus, under Assumptions 3.4.1 and 3.4.3, it follows easily that

$$w^\star(t) - w^i(t) \xrightarrow[t\to\infty]{} 0, \quad i \in \{1, \ldots, M\}, \text{ a.s.},$$

and

$$w^i(t) - w^j(t) \xrightarrow[t\to\infty]{} 0, \quad (i,j) \in \{1, \ldots, M\}^2, \text{ a.s.} \tag{3.24}$$

This shows that the trajectories of the distributed versions of the quantizers reach asymptotically a consensus with probability 1. In other words, if one of the sequences $\{w^i(t)\}_{t=0}^\infty$ converges then they all converge towards the same value. The rest of the paper is devoted to prove that this common value is in fact a zero of ∇C, i.e. a critical point.

To prove the result mentioned above, we will need the following assumption, which basically states that the components of $w^\star$ are parted, for every time t but also asymptotically. This assumption is similar in spirit to the main requirement of Theorem 3.2.2.

Assumption 3.4.4. *One has*

1. $\mathbb{P}\left\{w^\star(t) \in \mathcal{D}_*^\kappa\right\} = 1, \quad t \geq 0.$
2. $\mathbb{P}\left\{\liminf_{t\to\infty} \mathrm{dist}\left(w^\star(t), \complement\mathcal{D}_*^\kappa\right) > 0\right\} = 1, \quad t \geq 0.$

3.4.4 Consistency of the DALVQ

In this subsection we state our main theorem on the consistency of the DALVQ. Its proof is based on the Asynchronous G-Lemma, Theorem 3.4.1. The goal of the next proposition is to ensure that the first assumption of Theorem 3.4.1 holds.

Proposition 3.4.1. *Assume* $(\mathbf{AsY})_\mathbf{1}$ *or* $(\mathbf{AsY})_\mathbf{2}$ *holds and that Assumptions 3.2.1, 3.4.1 and 3.4.2 are satisfied then* $\varepsilon_t^\star > 0$, $t \geq 0$, $\varepsilon_t^\star \xrightarrow[t\to\infty]{} 0$ *and* $\sum_{t=0}^{\infty} \varepsilon_t^\star = \infty$.

The second condition required in Theorem 3.4.1 deals with the convergence of the two series defined by equations (3.19) and (3.20). The next Proposition 3.4.2 provides sufficient condition for the almost sure convergence of these series.

Proposition 3.4.2. *Assume* $(\mathbf{AsY})_\mathbf{1}$ *or* $(\mathbf{AsY})_\mathbf{2}$ *holds and that Assumptions 3.2.1, 3.4.1, 3.4.3 and 3.4.4 are satisfied then the series* $\sum_{t=0}^{\infty} \Delta M_t^{(1)}$ *and* $\sum_{t=0}^{\infty} \Delta M_t^{(2)}$ *converge almost surely in* $\left(\mathbb{R}^d\right)^\kappa$.

This next proposition may be considered has the most important step in the proof of the convergence of the DALVQ. It establishes the convergence of a series of the form $\sum_{t=0}^{\infty} \varepsilon_{t+1} \left\|\nabla C\left(w(t)\right)\right\|^2$. The analysis of the convergence of this type of series is standard for the analysis of stochastic gradient method (see for instance Benveniste et al. [9] and Bottou [19]). In our context, we pursue the fruitful use of the agreement vector sequence, $\{w^\star(t)\}_{t=0}^{\infty}$, and its related "steps", $\{\varepsilon_t^\star\}_{t=0}^{\infty}$.

Note that under Assumption 3.4.4, we have $h\left(w^\star(t)\right) = \nabla C\left(w^\star(t)\right)$ for all $t \geq 0$, almost surely, therefore the sequence $\{\nabla C\left(w^\star(t)\right)\}_{t=0}^{\infty}$ below is well defined.

Proposition 3.4.3. *Assume* $(\mathbf{AsY})_\mathbf{1}$ *or* $(\mathbf{AsY})_\mathbf{2}$ *holds and that Assumptions 3.2.1, 3.4.1, 3.4.3 and 3.4.4 are satisfied then*

1. $C\left(w^\star(t)\right) \xrightarrow[t\to\infty]{} C_\infty, \quad a.s.,$
   where C_∞ *is a* $[0,\infty)$*-valued random variable,*
2. $$\sum_{t=0}^{\infty} \varepsilon_{t+1}^\star \left\|\nabla C\left(w^\star(t)\right)\right\|^2 < \infty, \quad a.s. \tag{3.25}$$

Remark that from the convergence of the series given by equation (3.25) one can only deduce that $\liminf_{t\to\infty} \left\|\nabla C\left(w^\star(t)\right)\right\| = 0$.

We are now in a position to state the main theorem of this paper, which expresses the convergence of the distributed version towards some zero of the gradient of the distortion. In addition, the convergence results (3.24) imply that if a version converges then all the versions converge towards this value.

Theorem 3.4.2 (Asynchronous Theorem)**.** *Assume* $(\mathbf{AsY})_\mathbf{1}$ *or* $(\mathbf{AsY})_\mathbf{2}$ *holds and that Assumptions 3.2.1, 3.4.1, 3.4.2, 3.4.3 and 3.4.4 are satisfied then*

1. $w^*(t) - w^i(t) \xrightarrow[t\to\infty]{} 0, \quad i \in \{1, \ldots, M\}$, *a.s.,*
2. $w^i(t) - w^j(t) \xrightarrow[t\to\infty]{} 0, \quad (i,j) \in \{1, \ldots, M\}^2$, *a.s.,*
3. $\mathrm{dist}\left(w^\star(t), \Xi_\infty\right) \xrightarrow[t\to\infty]{} 0, \quad$ *a.s.,*
4. $\mathrm{dist}\left(w^i, \Xi_\infty\right) \xrightarrow[t\to\infty]{} 0, \quad i \in \{1, \ldots, M\}$, *a.s.,*

where Ξ_∞ *is some random connected component of the set* $\{\nabla C = 0\} \cap \mathcal{G}^\kappa$.

3.4.5 Annex

Sketch of the proof of Asynchronous G-Lemma 3.4.1. The proof is an adaptation of the one found by Fort and Pagès, Theorem 4 in [44]. The recursive equation (3.21) satisfied by the sequence $\{w^\star(t)\}_{t=0}^\infty$ is similar to the iterations (2) in [44] (with the notation of this paper):

$$X^{t+1} = X^t - \varepsilon_{t+1} h\left(X^t\right) + \varepsilon_{t+1}\left(\Delta M^{t+1} + \eta^{t+1}\right), \quad t \geq 0.$$

Thus, similarly, we define the following family of continuous time stepwise function $\{u \mapsto \check{w}(t,u)\}_{t=1}^\infty$.

$$\check{w}^\star(0,u) \triangleq w^\star(s), \text{ if } u \in [\varepsilon_1^\star + \ldots + \varepsilon_s^\star, \varepsilon_1^\star + \ldots + \varepsilon_{s+1}^\star), \quad u \in [0,\infty).$$

and if $u < \varepsilon_1^\star$, $\check{w}^\star(0,u) = w^\star(0)$.

$$\check{w}^\star(t,u) \triangleq \check{w}^\star\left(0, \varepsilon_1^\star + \ldots + \varepsilon_t^\star + u\right), \quad t \geq 1 \text{ and } u \in [0,\infty).$$

Hence, for every $t \in \mathbb{N}$,

$$\check{w}^\star(t,u) = \check{w}^\star(0,t) - \int_0^u h\left(\check{w}^\star(t,v)\right) dv + R_u(t), \quad u \in [0,\infty),$$

where, for every $t \geq 1$ and $u \in [\varepsilon_1^\star + \ldots + \varepsilon_{t+t'}^\star, \varepsilon_1^\star + \ldots + \varepsilon_{t+t'+1}^\star)$,

$$R_u(t) \triangleq \int_{\varepsilon_t^\star + \ldots + \varepsilon_{t+t'}^\star}^{\varepsilon_1^\star + \ldots + \varepsilon_t^\star + u} \check{w}^\star(0,v) dv + \sum_{s=t+1}^{t+t'} \left(\Delta M_s^{(1)} + \Delta M_s^{(2)}\right).$$

The only difference between the families of continuous time functions $\{\check{w}(t,u)\}_{t=1}^\infty$ and $\left\{X^{(t)}\right\}_{t=1}^\infty$ defined in [44] is the remainder term $R_u(t)$. The convergence

$$\sup_{u\in[0,T]} \|R_u(t)\| \xrightarrow[t\to\infty]{} 0, \quad T > 0.$$

follows easily from the third assumption of Theorem 3.4.1. The rest of the proof follows similarly as in Theorem 4 [44].

□

Proof of Lemma 3.4.1. For all $i \in \{1, \dots, M\}$, and all $t \geq 0$, and all integer $1 \leq \ell \leq \kappa$, we may write

$$\begin{aligned}
&\left\|w_\ell^i(t) - w_\ell^\star(t)\right\| \\
&= \left\|\sum_{j=1}^{M}\left(\left(\phi^{i,j}(t,-1) - \phi^j(-1)\right) w_\ell^j(0) + \sum_{\tau=0}^{t-1}\left(\phi^{i,j}(t,\tau) - \phi^j(t)\right) s_\ell^j(\tau)\right)\right\| \\
&\qquad \text{(by Definition 3.3.2 and Lemma 3.3.1)} \\
&\leq \sum_{j=1}^{M}\left|\phi^{i,j}(t,-1) - \phi^j(-1)\right| \left\|w_\ell^j(0)\right\| + \sum_{\tau=0}^{t-1}\sum_{j=1}^{M}\left|\phi^{i,j}(t,\tau) - \phi^j(t)\right| \left\|s_\ell^j(\tau)\right\| \\
&\leq A\rho^{t+1}\sum_{j=1}^{M}\left\|w_\ell^j(0)\right\| + A\sum_{\tau=0}^{t-1}\sum_{j=1}^{M}\rho^{t-\tau}\left\|s_\ell^j(\tau)\right\| \\
&\qquad \text{(by Lemma 3.3.2).}
\end{aligned}$$

Thus,

$$\begin{aligned}
&\left\|w_\ell^i(t) - w_\ell^\star(t)\right\| \\
&\leq A\rho^{t+1}\sum_{j=1}^{M}\left\|w_\ell^j(0)\right\| + A\sum_{\tau=0}^{t-1}\sum_{j=1}^{M}\rho^{t-\tau}\varepsilon_{\tau+1}^j \mathbb{1}_{\{\tau \in T^j\}}\left\|H(\mathbf{z}_{\tau+1}^j, w^j(\tau))_\ell\right\| \\
&\qquad \text{(by equation (3.17))} \\
&\leq A\rho^{t+1}\sum_{j=1}^{M}\left\|w_\ell^j(0)\right\| \\
&\quad + A\sum_{\tau=0}^{t-1}\sum_{j=1}^{M}\rho^{t-\tau}\varepsilon_{\tau+1}^j \mathbb{1}_{\tau \in T^j}\mathbb{1}_{\left\{\mathbf{z}_{\tau+1}^j \in W_\ell(w^j(\tau))\right\}}\left\|w_\ell^j(\tau) - \mathbf{z}_{\tau+1}^j\right\|.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\left\|w_\ell^i(t) - w_\ell^\star(t)\right\| \\
&\leq AM\,\mathrm{diam}(\mathcal{G})\rho^{t+1} + A\,\mathrm{diam}(\mathcal{G})K_2 M\sum_{\tau=0}^{t-1}\frac{1}{\tau \vee 1}\rho^{t-\tau} \\
&\qquad \text{(because } 0 \in \mathcal{G} \text{ and by Assumptions 3.4.1 and 3.4.3)} \\
&\leq A\,\mathrm{diam}(\mathcal{G})K_2 M\sum_{\tau=-1}^{t-1}\frac{1}{\tau \vee 1}\rho^{t-\tau}.
\end{aligned}$$

Consequently,

$$\begin{aligned}\left\|w^\star(t) - w^i(t)\right\| &= \sqrt{\sum_{\ell=1}^{\kappa} \left\|w_\ell^i(t) - w_\ell^\star(t)\right\|^2} \\ &\leq \sqrt{\kappa} M \operatorname{diam}(\mathcal{G}) A K_2 \sum_{\tau=-1}^{t-1} \frac{1}{\tau \vee 1} \rho^{t-\tau}.\end{aligned}$$

This proves the desired result.

□

Let us now introduce the following events: for any $\delta > 0$ and $t \geq 0$,

$$A_\delta^t \triangleq \left\{w^\star(\tau) \in \mathcal{G}_\delta^\kappa \ , \ \ t \geq \tau \geq 0\right\}.$$

Recall that the $\mathcal{G}_\delta^\kappa$ is a compact subset of $\mathcal{G}^\kappa$ defined by equality (3.9). The next lemma establishes a detailed analysis of security regions for the parted components of the sequences $\{w^\star(t)\}_{t=0}^\infty$ and $\{w^j(t)\}_{t=0}^\infty$.

Lemma 3.4.2. *Let Assumptions 3.4.1 and 3.4.3 hold. Then,*

1. *there exists an integer $t_\delta^1 \geq 1$ such that*

$$A_\delta^t \subset A_{\delta/2}^{t+1}, \quad t \geq t_\delta^1.$$

Moreover,

$$w^\star(t) \in \mathcal{G}_\delta^\kappa \Rightarrow [w^\star(t), w^\star(t+1)] \subset \mathcal{G}_{\delta/2}^\kappa, \quad t \geq t_\delta^1.$$

2. *There exists an integer $t_\delta^2 \geq 1$ such that*

$$w^\star(t) \in \mathcal{G}_\delta^\kappa \Rightarrow [w^\star(t), w^i(t)] \subset \mathcal{G}_{\delta/2}^\kappa, \quad i \in \{1, \ldots, M\} \text{ and } t \geq t_\delta^2.$$

Proof of Lemma 3.4.2. Proof of statement 1. The proof starts with the observation that under Assumption 3.4.3 we have $w^j(t) \in \mathcal{G}^\kappa$, for all $i \in \{1, \ldots, M\}$ and $t \geq 0$. It follows that, for any $1 \leq \ell \leq \kappa$,

$$\begin{aligned}\left\|H\left(\mathbf{z}_{t+1}^j, w^j(t)\right)_\ell\right\| &\leq \left\|\mathbf{z}_{t+1}^j - w_\ell^j(t)\right\| \\ &\leq \operatorname{diam}(\mathcal{G}).\end{aligned}$$

Let us now provide an upper bound on the norm of the differences between two consecutive values of the agreement vector sequence. We may write, for all $t \geq 0$ and all $1 \leq \ell \leq M$,

$$\begin{aligned}
&\|w_\ell^\star(t+1) - w_\ell^\star(t)\| \\
&\quad = \left\| \sum_{j=1}^{M} \phi^j(t) s_\ell^j(t) \right\| \\
&\quad \leq \sum_{j=1}^{M} \phi^j(t) \left\| s_\ell^j(t) \right\| \\
&\quad \leq \sum_{j=1}^{M} \varepsilon_{t+1}^j \mathbb{1}_{\{t \in T^j\}} \left\| H\left(\mathbf{z}_{t+1}^j, w^j(t)\right)_\ell \right\| \\
&\qquad \text{(by equation (3.17) and statement 1. of Lemma 3.3.2)} \\
&\quad \leq \frac{M \operatorname{diam}(\mathcal{G}) K_2}{t \vee 1} \\
&\qquad \text{(by Assumption 3.4.1).}
\end{aligned} \tag{3.26}$$

Take $t \geq \frac{4}{\delta} M \operatorname{diam}(\mathcal{G}) K_2$ and $1 \leq k \neq \ell \leq M$. Let α be a real number in the interval $[0, 1]$.

If $w^\star(t) \in \mathcal{G}_\delta^\kappa$ then

$$\begin{aligned}
&\|(1-\alpha) w_\ell^\star(t) + \alpha w_\ell^\star(t+1) - (1-\alpha) w_k^\star(t) - \alpha w_k^\star(t+1)\| \\
&\quad = \|w_\ell^\star(t) - w_k^\star(t) + \alpha\left(w_\ell^\star(t+1) - w_\ell^\star(t)\right) + \alpha\left(w_k^\star(t) - w_k^\star(t+1)\right)\| \\
&\quad \geq \|w_\ell^\star(t) - w_k^\star(t)\| - \|\alpha\left(w_\ell^\star(t+1) - w_\ell^\star(t)\right) + \alpha\left(w_k^\star(t) - w_k^\star(t+1)\right)\| \\
&\quad \geq \|w_\ell^\star(t) - w_k^\star(t)\| - \alpha\|w_\ell^\star(t+1) - w_\ell^\star(t)\| - \alpha\|w_k^\star(t) - w_k^\star(t+1)\| \\
&\quad \geq \delta - 2\alpha\frac{\delta}{4} \\
&\quad \geq \delta/2.
\end{aligned}$$

This proves that the whole segment $[w^\star(t), w^\star(t+1)]$ is contained in $\mathcal{G}_{\delta/2}^\kappa$.

Proof of statement 2. Take $t \geq 1$ and $1 \leq \ell \leq M$. If $w^\star(t) \in \mathcal{G}_\delta^\kappa$ then by Lemma 3.4.1, there exists t_δ^2 such that

$$\left\| w_\ell^\star(t) - w_\ell^i(t) \right\| \leq \frac{\delta}{4}, \quad i \in \{1, \ldots, M\} \text{ and } t \geq t_\delta^2.$$

Let k and ℓ two distinct integers between 1 and M. For any $t \geq t_\delta^2$,

$$\begin{aligned}
&\left\|\alpha w_k^i(t) + (1-\alpha)w_k^\star(t) - \alpha w_\ell^i(t) - (1-\alpha)w_\ell^\star(t)\right\| \\
&\quad = \left\|w_k^\star(t) - w_\ell^\star(t) + \alpha(w_k^i(t) - w_k^\star(t)) + \alpha(w_\ell^\star(t) - w_\ell^i(t))\right\| \\
&\quad \geq \|w_k^\star(t) - w_\ell^\star(t)\| - \alpha\left\|w_k^i(t) - w_k^\star(t)\right\| - \alpha\left\|w_\ell^\star(t) - w_\ell^i(t)\right\| \\
&\quad \geq \delta - 2\alpha\frac{\delta}{4} \\
&\quad \geq \delta/2.
\end{aligned}$$

This implies $[w^\star(t), w^i(t)] \subset \mathcal{G}_{\delta/2}^\kappa$, as desired.

□

Proof of Proposition 3.4.1. By definition $\varepsilon_{t+1}^\star$ equals $\sum_{j=1}^M \mathbb{1}_{\{t \in T^j\}}\phi^j(t)\varepsilon_{t+1}^j$, for all $t \geq 0$.

On the one hand, since the real number $\phi^j(t)$ belongs to the interval $[\eta, 1]$ (by Lemma 3.3.2) $\varepsilon_{t+1}^\star$ is bounded from above by $\frac{MK_2}{t \vee 1}$ using the right-hand side inequality of Assumption 3.4.1.

On the other hand, $\varepsilon_{t+1}^\star$ is bounded from below by the nonnegative real number $\eta\frac{K_1}{t\vee 1}$ using the left-hand side inequality of Assumption 3.4.1. Note also that as Assumption 3.4.2 holds, this real number is a positive one. Therefore, it follows that

$$\varepsilon_t^\star \xrightarrow[t\to\infty]{} 0$$

and

$$\sum_{t=0}^{\infty} \varepsilon_t^\star = \infty.$$

□

Proof of Proposition 3.4.2. Consistency of $\sum_{t=0}^\infty \Delta M_t^{(1)}$. Let δ be a positive

real number and let $t \geq t_\delta^2$, where t_δ^2 is given by Lemma 3.25. We may write

$$\begin{aligned}
&\mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j \left\| h\left(w^\star(t)\right) - h\left(w^j(t)\right) \right\| \\
&\quad \leq \mathbb{1}_{\left\{[w^\star(t), w^j(t)] \subset \mathcal{G}_{\delta/2}^\kappa\right\}} \sum_{j=1}^{M} \phi^j(t) \varepsilon_{t+1}^j \left\| \nabla C\left(w^\star(t)\right) - \nabla C\left(w^j(t)\right) \right\| \\
&\qquad \text{(using statement 2. of Lemma 3.4.2 and the fact that } \nabla C = h \text{ on } \mathcal{D}_*^\kappa) \\
&\quad \leq \mathbb{1}_{\left\{[w^\star(t), w^j(t)] \subset \mathcal{G}_{\delta/2}^\kappa\right\}} P_{\delta/2} \sum_{j=1}^{M} \varepsilon_{t+1}^j \left\| w^\star(t) - w^j(t) \right\| \\
&\qquad \text{(by Lemma 3.2.1)} \\
&\quad \leq \sqrt{\kappa}\, \mathrm{diam}(\mathcal{G}) A K_2^2 P_{\delta/2} M^2 \frac{\theta_t}{t} \\
&\qquad \text{(by Lemma 3.4.1).}
\end{aligned}$$

Thus, since $\sum_{t=0}^{\infty} \frac{\theta_t}{t} < \infty$, the series

$$\sum_{t=0}^{\infty} \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j \left\| h\left(w^\star(t)\right) - h\left(w^j(t)\right) \right\|$$

is almost surely convergent. Under Assumption 3.4.4, we have

$$\mathbb{P}\left\{ \bigcup_{\delta>0} \bigcap_{t \geq 0} A_\delta^t \right\} = 1.$$

It follows that the series $\sum_{t=0}^{\infty} \Delta M_t^{(1)}$ converges almost surely in $\left(\mathbb{R}^d\right)^\kappa$.

Consistency of $\sum_{t=0}^{\infty} \Delta M_t^{(2)}$. The sequence of random variables $M_t^{(2)}$ defined, for all $t \geq 0$, by

$$\begin{aligned}
M_t^{(2)} &\triangleq \sum_{\tau=0}^{t} \Delta M_\tau^{(2)} \\
&= \sum_{\tau=0}^{t} \sum_{j=1}^{M} \mathbb{1}_{\{\tau \in T^j\}} \varepsilon_{\tau+1}^j \phi^j(\tau) \left(h\left(w^j(\tau)\right) - H\left(\mathbf{z}_{\tau+1}^j, w^j(\tau)\right) \right).
\end{aligned}$$

is a vector valued martingale with respect to the filtration $\{\mathcal{F}_t\}_{t=0}^{\infty}$. It turns out that this martingale has square integrable increments. Precisely,

$$\sum_{t=0}^{\infty} \mathbb{E}\left\{ \left\| M_{t+1}^{(2)} - M_t^{(2)} \right\|^2 \;\middle|\; \mathcal{F}_t \right\} = \sum_{t=1}^{\infty} \mathbb{E}\left\{ \left\| \Delta M_t^{(2)} \right\|^2 \;\middle|\; \mathcal{F}_t \right\} < \infty.$$

Indeed, for all $j \in \{1, \ldots, M\}$ and $t \geq 1$,

$$\begin{aligned}
&\sum_{\tau=1}^{t} \mathbb{E}\left\{ \left\| \mathbb{1}_{\{\tau \in T^j\}} \varepsilon_{\tau+1}^j \left(h\left(w^j(\tau)\right) - H\left(\mathbf{z}_{\tau+1}^j(\tau), w^j(\tau)\right)\right)\right\|^2 \;\middle|\; \mathcal{F}_\tau \right\} \\
&\quad \leq \sum_{\tau=1}^{t} \left(\varepsilon_{\tau+1}^j\right)^2 \mathbb{E}\left\{ \left\| h\left(w^j(\tau)\right) - H\left(\mathbf{z}_{\tau+1}^j(\tau), w^j(\tau)\right)\right\|^2 \;\middle|\; \mathcal{F}_\tau \right\} \\
&\quad \leq 2 \sum_{\tau=1}^{t} \left(\varepsilon_{\tau+1}^j\right)^2 \mathbb{E}\left\{ \left\| h\left(w^j(\tau)\right)\right\|^2 + \left\| H\left(\mathbf{z}_{\tau+1}^j(\tau), w^j(\tau)\right)\right\|^2 \;\middle|\; \mathcal{F}_\tau \right\} \\
&\quad \leq 4\kappa \operatorname{diam}(\mathcal{G})^2 \sum_{\tau=1}^{t} \left(\varepsilon_{\tau+1}^j\right)^2 \\
&\qquad \text{(using Assumption 3.4.3)} \\
&\quad \leq 4\kappa \operatorname{diam}(\mathcal{G})^2 K_2^2 \sum_{\tau=1}^{t} \frac{1}{\tau^2}.
\end{aligned}$$

We conclude that the series $\sum_{t \geq 1} \Delta M_t^{(2)}$ is almost surely convergent.

□

Proof of proposition 3.4.3. Denote by $\langle x, y \rangle$ the canonical inner product of two vectors $x, y \in \mathbb{R}^d$ and also, with a slight abuse of notation, the canonical inner product of two vectors $x, y \in \left(\mathbb{R}^d\right)^\kappa$. Let δ be a positive real number.
Take any $t \geq \max\left\{t_\delta^1, t_\delta^2\right\}$, where t_δ^1 and t_δ^2 are defined as in Lemma 3.4.2. One has,

$$\mathbb{1}_{A_\delta^{t+1}} C\left(w^\star(t+1)\right) \leq \mathbb{1}_{A_\delta^t} C\left(w^\star(t+1)\right).$$
$$\text{(by definition } A_\delta^{t+1} \subset A_\delta^t\text{)}$$

Consequently,

$$\begin{aligned}
&\mathbb{1}_{A_\delta^{t+1}} C\left(w^\star(t+1)\right) \\
&\quad \leq \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) + \mathbb{1}_{A_\delta^t} \langle \nabla C(w^\star(t)), w^\star(t+1) - w^\star(t) \rangle \\
&\qquad + \mathbb{1}_{\left\{[w^\star(t), w^\star(t+1)] \subset \mathcal{G}_{\delta/2}^\kappa\right\}} \\
&\qquad\quad \times \left[\sup_{z \in [w^\star(t), w^\star(t+1)]} \left\{ \left\| \nabla C(z) - \nabla C(w^\star(t)) \right\| \right\} \left\| w^\star(t+1) - w^\star(t) \right\| \right] \\
&\quad \leq \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) + \mathbb{1}_{A_\delta^t} \langle \nabla C(w^\star(t)), w^\star(t+1) - w^\star(t) \rangle \\
&\qquad + P_{\delta/2} \left\| w^\star(t+1) - w^\star(t) \right\|^2 \\
&\qquad\quad \text{(using Lemma 3.2.1.)}
\end{aligned}$$

The first inequality above holds since the bounded increment formula above is valid by statement 1 of Lemma 3.4.2. Let us now bound separately the right hand side members of the second inequality.

Firstly, the next inequality holds by inequality (3.26) provided in the proof of Lemma 3.4.2,

$$P_{\delta/2} \left\| w^\star(t+1) - w^\star(t) \right\|^2 \leq \kappa P_{\delta/2} \left(\frac{K_2 M \operatorname{diam}(\mathcal{G})}{t} \right)^2 .$$

Secondly,

$$\begin{aligned}
&\mathbb{1}_{A_\delta^t} \langle \nabla C(w^\star(t)), w^\star(t+1) - w^\star(t) \rangle \\
&\quad = \mathbb{1}_{A_\delta^t} \langle \nabla C(w^\star(t)), \sum_{j=1}^{M} \phi^j(t) s^j(t) \rangle \\
&\qquad \text{(by equation (3.15))} \\
&\quad = \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^j(t)), \phi^j(t) s^j(t) \rangle \\
&\qquad + \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^\star(t)) - \nabla C(w^j(t)), \phi^j(t) s^j(t) \rangle .
\end{aligned}$$

Thus,

$$\begin{aligned}
&\mathbb{1}_{A_\delta^t} \langle \nabla C(w^\star(t)), w^\star(t+1) - w^\star(t) \rangle \\
&\quad \leq \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^j(t)), \phi^j(t) s^j(t) \rangle \\
&\qquad + \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \left| \langle \nabla C(w^\star(t)) - \nabla C(w^j(t)), \phi^j(t) s^j(t) \rangle \right| \\
&\quad \leq \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^j(t)), \phi^j(t) s^j(t) \rangle \\
&\qquad + \sum_{j=1}^{M} \mathbb{1}_{A_\delta^t} \left\| \nabla C(w^\star(t)) - \nabla C(w^j(t)) \right\| \left\| \phi^j(t) s^j(t) \right\| \\
&\qquad \text{(using Cauchy-Schwarz inequality).}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\mathbb{1}_{A_\delta^t}\langle \nabla C(w^\star(t)), w^\star(t+1) - w^\star(t)\rangle \\
&\quad \leq \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^j(t)), \phi^j(t) s^j(t)\rangle \\
&\qquad + \sum_{j=1}^{M} \mathbb{1}_{\left\{[w^\star(t), w^j(t)] \subset \mathcal{G}_{\delta/2}^\kappa\right\}} \left\|\nabla C(w^\star(t)) - \nabla C(w^j(t))\right\| \left\|\phi^j(t) s^j(t)\right\| \\
&\qquad\quad \text{(by statement 2 of Lemma 3.4.2)} \\
&\quad \leq \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^j(t)), \phi^j(t) s^j(t)\rangle \\
&\qquad + P_{\delta/2} \sum_{j=1}^{M} \left\|w^\star(t) - w^j(t)\right\| \left\|\phi^j(t) s^j(t)\right\| \\
&\qquad\quad \text{(using Lemma 3.2.1)}
\end{aligned}$$

$$\begin{aligned}
&\mathbb{1}_{A_\delta^t}\langle \nabla C(w^\star(t)), w^\star(t+1) - w^\star(t)\rangle \\
&\quad \leq \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^j(t)), \phi^j(t) s^j(t)\rangle \\
&\qquad + P_{\delta/2} A K_2^2 \kappa M^2 \operatorname{diam}(\mathcal{G})^2 \frac{\theta_t}{t} \\
&\qquad\quad \text{(using Lemma 3.4.1 and the upper bound (3.26)).}
\end{aligned}$$

Finally,

$$\begin{aligned}
&\mathbb{1}_{A_\delta^{t+1}} C\left(w^\star(t+1)\right) \\
&\quad \leq \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) + \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \langle \nabla C(w^j(t)), \phi^j(t) s^j(t)\rangle \\
&\qquad + P_{\delta/2} A K_2^2 \kappa M^2 \operatorname{diam}(\mathcal{G})^2 \frac{\theta_t}{t} \\
&\qquad + \kappa P_{\delta/2} \left(\frac{K_2 M \operatorname{diam}(\mathcal{G})}{t}\right)^2 .
\end{aligned} \tag{3.27}$$

Set

$$\Omega_\delta^1 \triangleq P_{\delta/2} A K_2^2 \kappa M^2 \operatorname{diam}(\mathcal{G})^2$$

and

$$\Omega_\delta^2 \triangleq \kappa P_{\delta/2} \left(K_2 M \operatorname{diam}(\mathcal{G})\right)^2 .$$

In the sequel, we shall need the following lemma.

Lemma 3.4.3. *For all $t \geq \max\{t_\delta^1, t_\delta^2\}$, the quantity W_t below is a nonnegative supermartingale with respect to the filtration $\{\mathcal{F}_t\}_{t=0}^{\infty}$:*

$$W_t \triangleq \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) + \eta K_1 \sum_{\tau=0}^{t-1} \mathbb{1}_{A_\delta^\tau} \frac{1}{\tau} \sum_{j=1}^{M} \mathbb{1}_{\{\tau \in T^j\}} \left\| \nabla C\left(w^j(\tau)\right) \right\|^2 + \Omega_\delta^1 \sum_{\tau=t}^{\infty} \frac{\theta(\tau)}{\tau} + \Omega_\delta^2 \sum_{\tau=t}^{\infty} \frac{1}{\tau^2}, \quad t \geq 1.$$

Proof of Lemma 3.4.3. Indeed, using the upper bound provided by equation (3.27),

$$\begin{aligned}
&\mathbb{E}\left\{ \mathbb{1}_{A_\delta^{t+1}} C\left(w^\star(t+1)\right) \;\middle|\; \mathcal{F}_t \right\} \\
&\leq \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) + \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \mathbb{E}\left\{ \langle \nabla C(w^j(t)), \phi^j(t) s^j(t) \rangle \;\middle|\; \mathcal{F}_t \right\} \\
&\quad + \Omega_\delta^1 \frac{1}{t} \theta_t + \Omega_\delta^2 \frac{1}{t^2} \\
&= \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) \\
&\quad + \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \left\langle \nabla C(w^j(t)), \mathbb{E}\left\{ -\mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j H(\mathbf{z}_{t+1}^j, w^j(t)) \rangle \;\middle|\; \mathcal{F}_t \right\} \right\rangle \\
&\quad + \Omega_\delta^1 \frac{\theta_t}{t} + \Omega_\delta^2 \frac{1}{t^2} \\
&= \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) \\
&\quad - \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \phi^j(t) \varepsilon_{t+1}^j \left\| \nabla C(w^j(t)) \right\|^2 + \Omega_\delta^1 \frac{\theta_t}{t} + \Omega_\delta^2 \frac{1}{t^2} \\
&\leq \mathbb{1}_{A_\delta^t} C\left(w^\star(t)\right) \\
&\quad - \frac{\eta K_1}{t} \mathbb{1}_{A_\delta^t} \sum_{j=1}^{M} \mathbb{1}_{\{t \in T^j\}} \left\| \nabla C(w^j(t)) \right\|^2 + \Omega_\delta^1 \frac{\theta_t}{t} + \Omega_\delta^2 \frac{1}{t^2}.
\end{aligned}$$

In the last inequality we used the fact that $\phi^j(t) \geq \eta$ (Lemma 3.3.2) and $\varepsilon_{t+1}^j \geq \frac{K_1}{t}$ (Assumption 3.4.1).

It is straightforward to verify that, we have $W_t - \mathbb{E}\{W_{t+1} | \mathcal{F}_t\} \geq 0$ which prove the desired result.

□

Proof of Proposition 3.4.3 (continued). Since $\{W_t\}_{t=1}^{\infty}$ is a nonnegative supermartingale (by Lemma 3.4.3), W_t converges almost surely as $t \to \infty$ (see for

instance Durrett [41]). Then, as $\sum_{\tau=t}^{\infty} \frac{\theta(\tau)}{\tau} \xrightarrow[t\to\infty]{} 0$ and $\sum_{\tau=t}^{\infty} \frac{1}{\tau^2} \xrightarrow[t\to\infty]{} 0$, we have

$$\mathbb{1}_{A_\delta^t} C(w^\star(t)) \xrightarrow[t\to\infty]{} C_\infty, \quad \text{a.s.}, \tag{3.28}$$

where $C_\infty \in [0, \infty)$ and, because the origin of the expression is increasing in t, the following series converges

$$\sum_{\tau=0}^{\infty} \mathbb{1}_{A_\delta^\tau} \frac{1}{\tau \vee 1} \sum_{j=1}^{M} \mathbb{1}_{\{\tau \in T^j\}} \left\| \nabla C \left(w^j(\tau) \right) \right\|^2 < \infty, \quad \text{a.s.} \tag{3.29}$$

Proof of statement 1. Assumption 3.4.4 means that

$$\mathbb{P}\left\{ \bigcup_{\delta>0} \bigcap_{t\geq 0} A_\delta^t \right\} = 1.$$

Statement 1 follows easily from the convergence (3.28).

Proof of statement 2. The required convergence (3.25) is proven as follows. We have

$$\begin{aligned}
&\sum_{\tau=0}^{t} \varepsilon_{\tau+1}^\star \mathbb{1}_{A_\delta^\tau} \left\| \nabla C \left(w^\star(\tau) \right) \right\|^2 \\
&\quad \leq \sum_{\tau=0}^{t} \sum_{j=1}^{M} \phi^j(\tau) \mathbb{1}_{\{\tau \in T^j\}} \mathbb{1}_{A_\delta^\tau} \varepsilon_{\tau+1}^j \left\| \nabla C \left(w^\star(\tau) \right) \right\|^2 \\
&\qquad \text{(using equality (3.18))} \\
&\quad \leq 2K_2 \sum_{\tau=0}^{t} \mathbb{1}_{A_\delta^\tau} \frac{1}{\tau \vee 1} \sum_{j=1}^{M} \mathbb{1}_{\{\tau \in T^j\}} \left\| \nabla C \left(w^j(\tau) \right) \right\|^2 \\
&\qquad \text{(using Assumption 3.4.2)} \\
&\qquad + 2K_2 \sum_{\tau=0}^{t} \mathbb{1}_{\left\{ [w^\star(\tau), w^j(\tau)] \subset \mathcal{G}_{\delta/2}^\kappa \right\}} \frac{1}{\tau \vee 1} \sum_{j=1}^{M} \left\| \nabla C \left(w^j(\tau) \right) - \nabla C \left(w^\star(\tau) \right) \right\|^2 \\
&\qquad \text{(using Assumption 3.4.2 and statement 2 of Lemma 3.4.2.)}
\end{aligned}$$

Thus,

$$\begin{aligned}
&\sum_{\tau=0}^{t} \varepsilon_{\tau+1}^\star \mathbb{1}_{A_\delta^\tau} \left\| \nabla C \left(w^\star(\tau) \right) \right\|^2 \\
&\quad \leq 2K_2 \sum_{\tau=0}^{t} \mathbb{1}_{A_\delta^\tau} \frac{1}{\tau \vee 1} \sum_{j=1}^{M} \mathbb{1}_{\{\tau \in T^j\}} \left\| \nabla C \left(w^j(\tau) \right) \right\|^2 \\
&\qquad + 2K_2 P_{\delta/2}^2 \sum_{\tau=0}^{t} \mathbb{1}_{\left\{ [w^\star(\tau), w^j(\tau)] \subset \mathcal{G}_{\delta/2}^\kappa \right\}} \frac{1}{\tau \vee 1} \sum_{j=1}^{M} \left\| w^j(\tau) - w^\star(\tau) \right\|^2 \\
&\qquad \text{(by Lemma 3.2.1).}
\end{aligned}$$

Thus,

$$\begin{aligned}
&\sum_{\tau=0}^{t} \varepsilon_{\tau+1}^{\star} \mathbb{1}_{A_\delta^\tau} \left\| \nabla C\left(w^\star(\tau)\right)\right\|^2 \\
&\quad \leq 2K_2 \sum_{\tau=0}^{t} \mathbb{1}_{A_\delta^\tau} \frac{1}{\tau \vee 1} \sum_{j=1}^{M} \mathbb{1}_{\{\tau \in T^j\}} \left\| \nabla C\left(w^j(\tau)\right)\right\|^2 \\
&\qquad + 2P_{\delta/2}^2 K_2^3 \kappa M^3 A^2 \operatorname{diam}(\mathcal{G})^2 \sum_{\tau=1}^{t} \frac{1}{\tau \vee 1} \theta_\tau^2 \\
&\qquad \text{(by Lemma 3.4.1).}
\end{aligned}$$

Finally, using the convergence (3.29), one has

$$\sum_{\tau=0}^{\infty} \varepsilon_{\tau+1}^{\star} \mathbb{1}_{A_\delta^\tau} \left\| \nabla C\left(w^\star(\tau)\right)\right\|^2 < \infty, \quad \text{a.s.},$$

and the conclusion follows from the fact that Assumption 3.4.4 implies

$$\mathbb{P}\left\{ \bigcup_{\delta>0} \bigcap_{t\geq 0} A_\delta^t \right\} = 1.$$

□

Proof of Theorem 3.4.2. The proof consists in verifying the assumptions of Theorem 3.4.1 with the function $\widehat{G}$ defined by equation (3.10).

It has been outlined that Assumption 3.4.3 implies that $w^\star(t)$ lie in the compact set $\mathcal{G}^\kappa$, almost surely, for all $t \geq 0$. Consequently, in the definition of $\widehat{G}(w^\star)$ the lim inf symbol can be omitted. For all $\mathbf{z} \in \mathcal{G}$ and all $t \geq 0$, we have $\|H(\mathbf{z}, w^\star(t))\| \leq \sqrt{\kappa} \operatorname{diam}(\mathcal{G})$, almost surely, whereas $\{h(w^\star(t))\}_{t=0}^{\infty}$ satisfies

$$h(w^\star(t)) = \mathbb{E}\left\{ H\left(\mathbf{z}, w^\star(t)\right) \mid \mathcal{F}_t \right\}, \quad t \geq 0, \text{ a.s.}$$

Thus, the sequences $\{w^\star(t)\}_{t=0}^{\infty}$ and $\{h(w^\star(t))\}_{t=0}^{\infty}$ are bounded almost surely.

Proposition 3.4.1, respectively Proposition 3.4.2, respectively Proposition 3.4.3 show that the first assumption, respectively the third assumption, respectively the fourth assumption of Theorem 3.4.1 hold. This concludes the proof of the theorem.

□

Justification of the statements (3.23). Recall that the definition of θ is provided in Lemma 3.4.1. Let us remark that it is sufficient to analyse the behavior in t of the quantity $\sum_{\tau=1}^{t-1} \rho^{t-\tau}/\tau$.
Let $\varepsilon > 0$ then for all $t \geq \lfloor 1/\varepsilon \rfloor + 1$, we have

$$\begin{aligned}
&\sum_{\tau=1}^{t-1} \frac{\rho^{t-\tau}}{\tau} \\
&\quad = \sum_{\tau=1}^{\lfloor 1/\varepsilon \rfloor} \frac{\rho^{t-\tau}}{\tau} + \sum_{\tau=\lfloor 1/\varepsilon \rfloor+1}^{t-1} \frac{\rho^{t-\tau}}{\tau} \\
&\quad \leq \sum_{\tau=1}^{\lfloor 1/\varepsilon \rfloor} \rho^{t-\tau} + \varepsilon \sum_{\tau=\lfloor 1/\varepsilon \rfloor+1}^{t-1} \rho^{t-\tau} \\
&\quad \leq \frac{\rho^{t-\lfloor 1/\varepsilon \rfloor}}{1-\rho} + \frac{\varepsilon}{1-\rho} \\
&\qquad \text{(using the fact that } \rho \in (0,1)\text{)}.
\end{aligned}$$

Consequently, for t sufficiently large we have

$$\sum_{\tau=1}^{t-1} \frac{\rho^{t-\tau}}{\tau} \leq \frac{2\varepsilon}{1-\rho}$$

which proves the first claim.
The second claim follows the same technique by letting "$\varepsilon = 1/\sqrt{t}$".
Thus, for $t \geq 1$ we have

$$\theta_t \leq \frac{\rho^{t-\lfloor \sqrt{t} \rfloor - 1}}{1-\rho} + \frac{1/\sqrt{t}}{1-\rho}.$$

Finally, for $T \geq 1$, it holds

$$\sum_{t=1}^{T} \sum_{\tau=1}^{t-1} \frac{\rho^{t-\tau}}{\tau} \leq \frac{1}{1-\rho} \left(\sum_{t=1}^{T} \rho^{n-\lfloor \sqrt{n} \rfloor - 1} + \sum_{t=1}^{T} \frac{1}{n^{3/2}} \right).$$

The two partial sums in the above parenthesis have finite limits which prove the second statement.

4 Computational considerations of DALVQ implementations

4.1 Introduction

In this chapter we discuss some aspects of practical implementations of Distributed Asynchronous Learning Vector Quantization procedures (DALVQ) introduced in Chapter 3. To this aim, various parallelization schemes are tested using simple programs simulating a distributed architecture, and synthetic vectorial data [1]. The first paragraphs describe some related contributions and the point of view adopted throughout the chapter.

The DALVQ algorithms are based upon the Competitive Learning Vector Quantization technique (CLVQ, see Section 3.2). Indeed, the former methods execute several CLVQ procedures on different (and possibly distant) computing resources while the results are broadcasted efficiently and asynchronously through the network. In Chapter 3 we explained that the CLVQ algorithm belongs to the class of stochastic gradient descent algorithms. Distributed and asynchronous stochastic gradient descent procedures for supervised machine learning problems have been studied theoretically and experimentally in Langford et al. [96] and in Louppe and Geurts [74]. In these papers the computer programs are based on shared memory architectures. However, as mentioned in the introduction of Chapter 3, our work focuses on the unsupervised learning problem of clustering without efficient distributed shared memory. The lack of such a shared memory introduces significant time penalties when accessing data, therefore slowing down the exchange of information between the computing entities. In order to avoid this problem, Zinkevich et al. propose in [98] a parallelized stochastic gradient descent scheme with no communication between processors until the end of the local executions. As mentioned by the authors, this perfectly suits to the popular distributed computing framework MapReduce (see, for instance, Dean and Ghemawat [37]). However, in our quantization (or clustering) context this approach would not work because the cost function, namely the distortion, is not convex (as discussed in Section 3.2). Therefore a final average of completely independent CLVQ executions would not

1. Source code is available at the address `http://code.google.com/p/clouddalvq/`

lead to satisfactory prototypes, as shown by the results of Section 4.3 below when the delays get large.

In the present chapter, we consider the quantization methods with a practical approach, in comparaison with Chapter 3 where we adopted a theoretical point of view. Thus, we suppose having access to a finite dataset containing n $\mathbb{R}^d$-valued samples $\{\mathbf{z}_t\}_{t=1}^n$ ($d \geq 1$, $n \geq 1$). The goal consists in finding a "good" quantization scheme for this dataset by building relevant prototypes $w \in \left(\mathbb{R}^d\right)^\kappa$. They will act as a "summary" of the dataset. The CLVQ technique defined by equation (3.7) builds such prototypes by computing iterations of $\left(\mathbb{R}^d\right)^\kappa$-valued vectors $\{w(t)\}_{t=0}^\infty$. Using the material and notation of Section 3.2, in this practical context, the CLVQ writes

$$w(t+1) = w(t) - \varepsilon_{t+1} H\left(\mathbf{z}_{\{t+1 \bmod n\}}, w(t)\right), \quad t \geq 0, \tag{4.1}$$

where $w(0) \in \left(\mathbb{R}^d\right)^\kappa$. The difference with equation (3.7) is the *mod* operator, which denotes the remainder of an integer division operation. These iterations make passes (i.e., cycles) over the dataset until a stopping criterion is met. The comparison with the theoretical description, where dataset is supposed to be infinite and consequently where there is no cycle, have been studied by Bermejo and Cabestany in [10].

Practical settings also impose a new point of view regarding the performance criterion. In what follows, the performance of the clustering is measured using the empirical distortion. This quantity is denoted by C_{μ_n} in equation (3.1), and it will be, as for now, denoted by C_n for simplification. Thus, the empirical quantization performance of the prototypes $w \in \left(\mathbb{R}^d\right)^\kappa$ takes the form

$$C_n(w) = \sum_{t=1}^{n} \min_{\ell=1,\ldots,\kappa} \|\mathbf{z}_t - w_\ell\|^2. \tag{4.2}$$

We will not be concerned with practical optimizations of the several parameters of CLVQ implementations such as the initialization procedures or the choice of the sequence of steps $\{\varepsilon_t\}_{t=1}^\infty$. In the present chapter, we assume that a satisfactory CLVQ implementation has been found. Therefore, our goal is to make this algorithm "scalable" (i.e., which is able to cope with larger datasets) using multiple computing resources. Consequently, in all our experiments, the initialization procedure is always the same: each initial prototype is an arbitrary average of 20 data vectors. For a review of standard initialization procedures, the reader is referred to Peterson et al. [83] where a satisfactory method is also provided for the usual batch k-means algorithm. The techniques proposed by Milligan and Isaac [78], Bradley and Fayyad in [24], or Mirkin [79] also seem to perform well. As for the

learning rate, similarly to Langford et al. in [96], we set $\varepsilon_t = 1/\sqrt{t}$ for all $t \geq 1$. A thorough analysis of the theoretical properties of the learning rates can be found in Bottou et al. [22, 23].

The chapter is organized as follows. Section 4.2 describes the synthetic functional data used in the experiments of this and the next chapter. Section 4.3 is devoted to analysis and discussions about simple distributed schemes implementing theoretical DALVQ procedures. In particular, we show in this section that the first natural implementation of DALVQ cannot really bring satisfactory results. However, we describe alternative parallelization scheme that, as we hoped for, give substantial speed up, i.e., gain of time execution brought by more computing resources compared to a sequential execution.

4.2 Synthetic functional data

Many real-world applications produce high dimensional data. In numerous situations, such data can be considered as sampled functions. This is the case in the popular context of time series, meteorological data, spectrometric data etc. In order to strike practical large scale problems we analyse the clustering algorithms on such data. In the present section we explain the construction of the B-splines mixtures random generators used in our experiments. Note that recent research by Abraham et al. [3] and Rossi et al. [88] focuses on clustering algorithms for B-splines functional data.

4.2.1 B-splines functions

The B-splines functions have been intensively studied for two centuries. They have been analyzed both for their astonishing mathematical properties and their ability to create nice shapes. Consequently they are frequently used in industries related to design such as automotive industry, architecture, graphics editor softwares etc. In this subsection we provide only a computational definition. For more information on B-splines history and mathematical properties, we refer the reader to de Boor [36].

A B-spline function (we will write spline for short) is a linear combination of basic B-splines. The family of basic B-spline functions of degree n with χ knots $x_0 \leq x_1 \leq \ldots \leq x_{\chi-1}$ is composed of $\chi - n - 1$ piecewise polynomial functions. The symbols n and χ denote both integers satisfying $\chi \geq n + 2$. They can be defined recursively by the Cox-de Boor formula below (see for instance de Boor

[35]). For all $x \in [x_0, x_{\chi-1}]$,

$$b_{i,0}(x) = \begin{cases} 1 & \text{if } x_i \leq x < x_{i+1} \\ 0 & \text{otherwise} \end{cases}, \quad i = 0, \ldots, \chi - 2,$$

and

$$b_{i,n}(x) = \frac{x - x_i}{x_{i+n} - x_i} b_{i,n-1}(x) + \frac{x_{i+n+1} - x}{x_{i+n+1} - x_{i+1}} b_{i+1,n-1}(x), \quad i = 0, \ldots, \chi - n - 2.$$

The Figure 4.1 is a plot of the six cubic basic B-splines where $\chi = 10$ and $n = 3$.

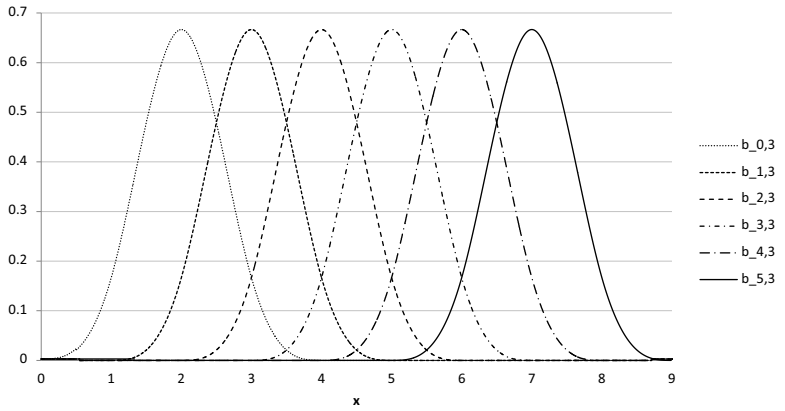

Figure 4.1: Plots of the six basic cubic B-spline functions with ten uniform knots: $x_0 = 0, \ldots, x_9 = 9$ ($n = 3$, $\chi = 10$).

As mentioned earlier, a spline is a vector of the linear span of the basic B-splines. Thus, a spline takes the form, for any $x \in [x_0, x_{\chi-1}]$,

$$b(x) = \sum_{i=0}^{\chi-n-2} p_i b_i(x), \tag{4.3}$$

where $\{p_i\}_{i=0}^{\chi-n-2} \in \mathbb{R}^{\chi-n-1}$. Figure 4.2 shows a cubic (i.e., $n = 3$) B-spline. The drawing illustrates the natural smoothness of such functions. In the sequel, we consider only cubic B-splines and uniform knots: $x_0 = 0, \ldots, x_{\chi-1} = \chi - 1$ with $\chi \geq 6$.

4.2.2 B-splines mixtures random generators

Let us now describe the random generators used throughout our experiments (of Chapter 4 and Chapter 5). Each of these vector-valued generators is based

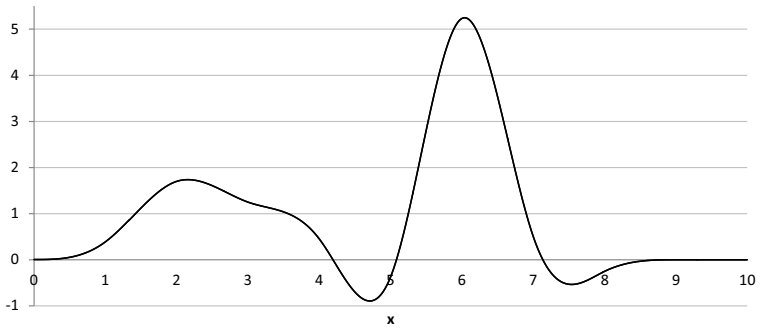

Figure 4.2: Plot of a cubic B-spline, a linear combination of the basic B-splines plotted in Figure 4.1

on G ($G \geq 1$) splines functions which, in the sequel, will be referred to as the centers of the mixture in the following. We also want these centers "not to be too closed to each others". To this aim, we chose our centers with (nearly) orthogonal coefficients p_i's appearing in equation (4.3). In the next paragraph, we explain the construction of such coefficients.

For $g = 0, \ldots, G-1$ and $i = 0, \ldots, \chi - 5$, let the $u_{g,i}$'s be reals, drawn independently from the standard uniform distribution on the open interval $(0, 1)$. The vectors $\{p_{g,i}\}_{i=0}^{\chi-5}$ are defined as block-wise orthogonal vectors computed by applying a revised Gram-Schmidt procedure on the G vectors $\{u_{g,i}\}_{i=0}^{\chi-5}$ (see for instance Greub [51]). Remind that the Gram-Schmidt algorithm is well defined if the number of vectors, here G, is lower than or equal to the dimension of the vector space which is equal to $\chi - 5$. Thus, in our case, we might have $G > \chi - 5$. Therefore, our revised block-wise Gram-Schmidt procedure simply consist in taking the first G vectors and applying the usual algorithm, and so on... The norm of all vectors of coefficients is set to a common value $sc > 0$.

For any $g \in \{1, \ldots, G-1\}$, let B_g be the spline defined by equation (4.3) with coefficients $\{p_{g,i}\}_{i=0}^{\chi-5}$ chosen as explained in the paragraph above. Remember that in our applied context, the data cannot be functions but only sampled function. Consequently, the centers of the distribution are the functions B_g's, only observed through the d-dimensional vector ($d \geq 1$) $\overline{B}_g$, where

$$\overline{B}_g = \{B_g\left(i(\chi - 1)/d\right)\}_{i=0}^{d-1}.$$

The vectors $\overline{B}_1, \ldots, \overline{B}_{G-1}$ define the centers of the $\mathbb{R}^d$-valued law of probability defined below. As shown in Figure 4.3, the orthogonal property of the coefficients

make them "not too close to each other", as requested.

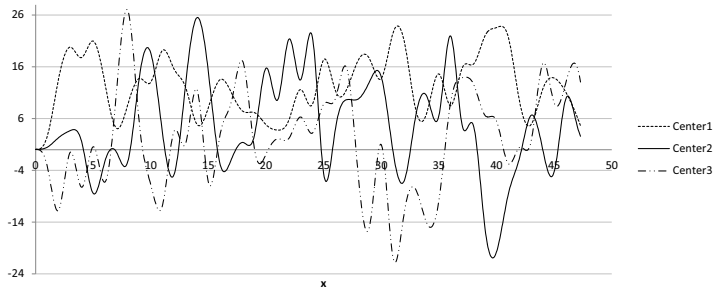

Figure 4.3: Plot of four splines centers with orthogonal coefficients: $\overline{B}_1, \ldots, \overline{B}_4$ where $G = 1500$, $d = 1500$, $\chi = 50$, $sc = 10$.

We are in position to define the distribution simulated by our $\mathbb{R}^d$-valued random generator. Let $\mathbf{N}$ be a uniform random variable over the set of integers $\{0, \ldots, G-1\}$ and ε a $\mathbb{R}^d$-valued Gaussian random variable with 0 mean and $\sigma^2 I_d$ for covariance matrix, where $\sigma > 0$ and I_d stands for the $d \times d$ identity matrix. Our random generators simulate the law of the d-dimensional random variable $\mathbf{Z}$, defined by

$$\mathbf{Z} = \overline{B}_{\mathbf{N}} + \varepsilon. \tag{4.4}$$

Figure 4.4 shows two independent realizations of the random variable $\mathbf{Z}$ defined by equation (4.4), the sampled functional data used in our experiments.

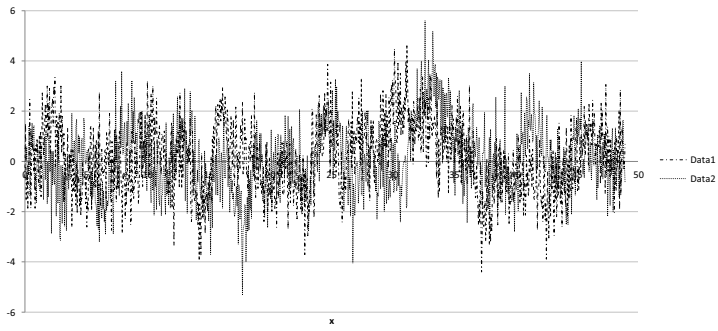

Figure 4.4: Plot of two independent realizations of the random variable $\mathbf{Z}$ defined by equation (4.4): $\overline{B_1}, \ldots, \overline{B_4}$ where $G = 1500$, $d = 1500$, $\chi = 50$, $sc = 10$.

4.3 Discussing parallelization schemes of the CLVQ algorithm

This section is devoted to the analysis of some practical parallelization schemes for the DALVQ algorithms. These methods were presented in Chapter 3 through theoretical equations, which can be models for various achievements. In this quantization context, the practical parallelization is not straightforward. For example, let us remark that, if two different prototypes versions w^1 and w^2 have been computed by two concurrent CLVQ executions, there is no guarantee that the quantization scheme provided by the convex combination $\alpha w_1 + (1-\alpha)w_2$ $(\alpha \in (0,1))$ has good performance regarding to the non convex distortion criterion. Therefore, a careful examination should be paid to the parallelization scheme of any attempt of DALVQ implementation.

In order to bring substantial speed up, three schemes for CLVQ parallelization are proposed below. These schemes have been tested using simple computer programs that simulate distributed situations. We start by the investigation of the most natural parallelization scheme, where the versions resulting of the distributed CLVQ executions are averaged with a temporal regularity. We track the evolution of the empirical distortion with the iterations. This evolution shows that no speed up can be brought using this scheme. Thus, a new model is proposed to overcome this disastrous situation. The main idea beneath this new scheme is the following one: the agreement between the computing entities is not performed using an average anymore, but uses instead the cumulated sum of the so-called "displacement terms". Finally, the last considered model is an improvement of the second model, where delays are introduced to allow analysis of asynchronism in the scheme.

4.3.1 A first parallel implementation

As explained in Section 3.4, the (potentially) large dataset is split among the local memory of M computing entities and is represented by the sequences $\{\mathbf{z}_t^i\}_{t=1}^n$, $i \in \{1,\ldots,M\}$ then, the global dataset is composed of nM vectors. Our investigation starts with the following simple parallelization scheme of the CLVQ technique. All 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:

$$\begin{cases} w_{temp}^i = w^i(t) - \varepsilon_{t+1} H\left(\mathbf{z}^i_{\{t+1 \bmod n\}}, w^i(t)\right) & \\ w^i(t+1) = w_{temp}^i & \text{if } t \bmod \tau \neq 0 \text{ or } t = 0, \\ \begin{cases} w^{srd} = \frac{1}{M}\sum_{j=1}^M w_{temp}^j \\ w^i(t+1) = w^{srd} \end{cases} & \text{if } t \bmod \tau = 0 \text{ and } t \geq \tau. \end{cases} \tag{4.5}$$

This algorithm is composed of two phases. Firstly, there is the processing phase which is given by the first equation of (4.5). It corresponds to a local execution of the CLVQ procedure. The result of these computations is referenced by the variables w^i_{temp}. Secondly, there is the averaging phase, described by the braced inner equations. Note that the averaging phase is executed only whenever τ points have been processed by each concurrent processors. The larger the integer τ is, the more the concurrent processors are left independent for their execution. During the averaging phase a common shared version w^{srd} is computed and all local versions, namely the w^i's, are set equal to w^{srd}. This shared version is the mean of all the w^i_{temp}'s. If the averaging phase does not occur, then the local versions are naturally set equal to the result of the local computations. See Figure 4.5 for a graphical illustration of iterations (4.5).

This algorithm fits the DALVQ model with simple settings: there is no asynchronism and the averaging phases are performed with a temporal regularity. The iterations (4.5) define a DALVQ procedure whose general form is given by equations (3.12) and (3.17). Precisely, using the notation introduced in Section 3.3, we have: if $t \bmod \tau \neq 0$, $a^{i,j}(t) = \delta_{i,j}$ [2] and if $t \bmod \tau = 0$, $a^{i,j}(t) = 1/M$. In addition, we also have all delays $\tau^{i,j}$'s equal zero and $T^i = \mathbb{N}$, for all $(i,j) \in \{1,\ldots,M\}^2$. Remind that if the delays are zero the algorithm has synchronization points. The equality $T^i = \mathbb{N}$ means that, for all time slots and all processors, local iterations of the CLVQ are processed. A straightforward examination shows that both sets of assumptions $(\mathbf{AsY})_\mathbf{1}$ and $(\mathbf{AsY})_\mathbf{2}$ are satisfied. Consequently, the Theorem 3.4.2 guarantees asymptotic consensus and convergence towards critical points.

We conducted the experiments of this parallelization scheme with a simulated distributed architecture. All experiments in this section are carried out with the following parameters: $n = 1000$, $d = 100$, $\kappa = 100$ and with synthetic data sampled by the random generators introduced in Section 4.2. The generators are tuned with the following settings: $G = 150$, $\chi = 100$, $sc = 100$ and $\sigma = 5.0$. The results are gathered with different values of τ ($\tau = 1, 10, 100$) in the charts of Figure 4.6. For each chart, we plot the performance curves when the distributed architecture has different number of computing instances: $M \in \{1, 2, 10\}$. To compare performance with different values of M and, consequently, with different datasets, we used a normalized empirical distortion of the vector w^{srd}. This empirical distortion is defined for all $w \in \left(\mathbb{R}^d\right)^\kappa$ by

$$C_{n,M}(w) = \frac{1}{nM} \sum_{i=1}^{M} \sum_{t=1}^{n} \min_{\ell=1,\ldots,\kappa} \left\| \mathbf{z}^i_t - w_\ell \right\|^2. \tag{4.6}$$

2. $\delta_{i,j}$ stands for the Kronecker symbol: $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ otherwise

The curve of reference is the sequential execution of the CLVQ, that is $M = 1$. We remark that for every value of $\tau = 1, 10, 100$, multiple resources do not bring speed up for convergence. Even if more data are processed, there is no acceleration in the convergence speed. To conclude, no gain in term of wall clock time can be brought using this parallel scheme. In the next subsection we investigate the cause of these non-satisfactory results and propose a solution to overcome this problem.

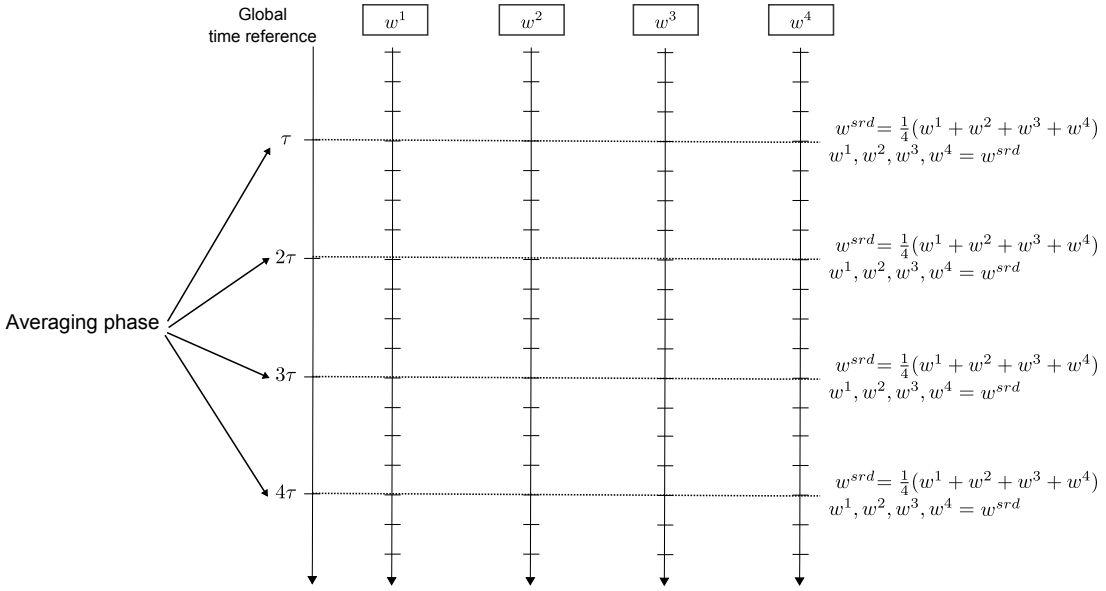

Figure 4.5: Illustration of the parallelization scheme of CLVQ procedures described by equations (4.5).

4.3.2 Towards a better parallelization scheme

Let us start the investigation of the parallel scheme given by iterations (4.5) by rewriting the sequential CLVQ iterations (4.1) using the material introduced in Section 4.1. For $t \geq \tau$ it holds

$$w(t+1) = w(t-\tau+1) - \sum_{t'=t-\tau+1}^{t} \varepsilon_{t'+1} H\left(\mathbf{z}_{\{t'+1 \bmod n\}}, w(t')\right). \tag{4.7}$$

For iterations (4.5), consider a time slots $t \geq 0$ where an averaging phase occurs, that is, $t \bmod \tau = 0$ and $t > 0$. Then, for all $i \in \{1, \ldots, M\}$,

$$w^i(t+1) = w^i(t-\tau+1) - \sum_{t'=t-\tau+1}^{t} \varepsilon_{t'+1} \left(\frac{1}{M} \sum_{j=1}^{M} H\left(\mathbf{z}^j_{t'+1}, w^j(t')\right) \right). \tag{4.8}$$

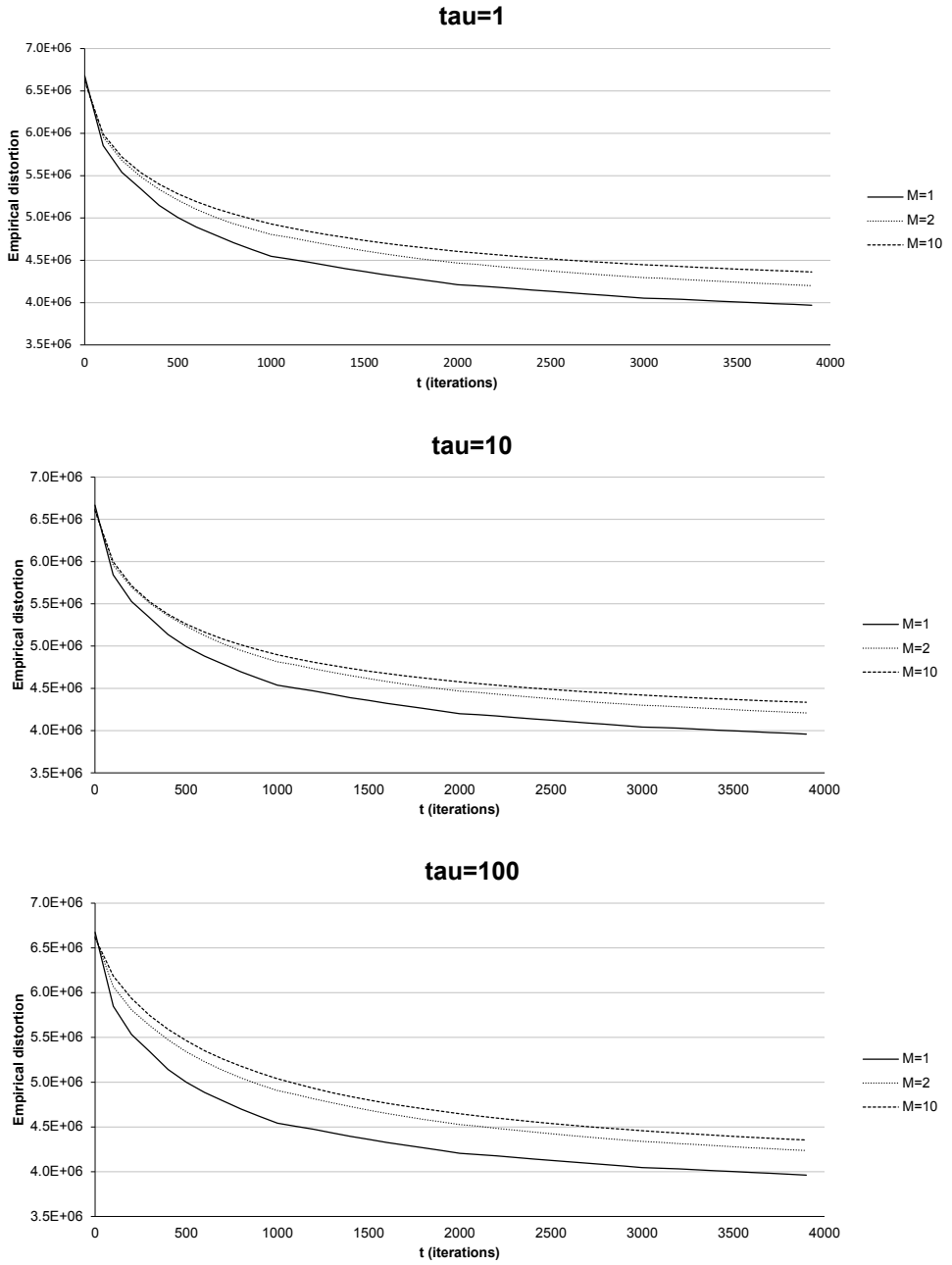

Figure 4.6: Charts of performance curves for iterations (4.5) with different number of computing entities: $M = 1, 2, 10$. The three charts correspond to different values of τ which is the integer that characterizes the frequency of the averaging phase ($\tau = 1, 10, 100$).

In the first situation of iterations (4.7), for a sample $\mathbf{z}$ and $t' > 0$, $H(\mathbf{z}, w(t'))$ is an observation and an estimator of the gradient $\nabla C(w(t'))$ (see Section 3.2). In the second situation of iterations (4.8), let us assume that the multiple versions are close to each other. This means that $w^j(t') \approx w^i(t')$, for all $(i,j) \in \{1,\dots,M\}^2$. Thus, the average

$$\frac{1}{M}\sum_{j=1}^{M} H\left(\mathbf{z}^j_{\{t'+1 \bmod n\}}, w^j(t')\right)$$

can also be viewed as an estimation of the gradient $\nabla C(w^i(t'))$, where $i \in \{1,\dots,M\}$.

In both situations, the descent terms following the step $\varepsilon_{t'+1}$ in equations (4.7) and (4.8) are estimators of gradients. Consequently, the two algorithms can be thought of as stochastic gradient descent procedures with different estimators. However, they are driven by the same learning rate which is given by the sequence of steps $\{\varepsilon_t\}_{t=1}^{\infty}$. The convergence speed of a non-fixed step gradient descent procedure is essentially driven by the decreasing speed of the sequence of steps (see for instance Kushner et al. [68] or Benveniste et al. [9]). The choice of this sequence is subject to an exploration/convergence trade-off. In the case of CLVQ iterations (4.1) the time slot t and the current count of samples processed are equal. Therefore, for the same time slot t, the distributed scheme of iterations (4.5) has visited much more data than the sequential CLVQ algorithm to build its descent terms. Yet, since the two algorithms share the same learning rate, they are expected to get comparable convergence speeds, which is confirmed by the charts in Figure 4.6.

We now introduce for all $j \in \{1,\dots,M\}$ and $t_2 \geq t_1 \geq 0$ the term $\Delta^j_{t_1 \to t_2}$ which corresponds to the displacement of the prototypes computed by processor j during the time interval (t_1, t_2),

$$\Delta^j_{t_1 \to t_2} = \sum_{t'=t_1+1}^{t_2} \varepsilon_{t'+1} H\left(\mathbf{z}^j_{\{t'+1 \bmod n\}}, w^j(t')\right). \tag{4.9}$$

Using the notation above, the parallel scheme given by iterations (4.5) writes, for all $t > 0$ where $t \bmod \tau = 0$ and all $i \in \{1,\dots,M\}$,

$$w^i(t+1) = w^i(t-\tau+1) - \frac{1}{M}\sum_{j=1}^{M} \Delta^j_{t-\tau \to t}. \tag{4.10}$$

In the previous paragraphs we explained that the sequential CLVQ and the distributed scheme above share the same convergence speed with respect to the iterations $t \geq 0$. Therefore, if one considers the evolution of the learning rate with respect to the number of samples processed, then the distributed scheme has a much lower learning rate, thereby favoring exploration to the detriment of the

convergence. As explained in the introduction of the chapter, we assume that the learning rate provided by the sequence $\{\varepsilon_t\}_{t=1}^{\infty}$ is supposed to be satisfactory for the sequential CLVQ. In this work, we seek for a parallel scheme that has, in comparison of a sequential CLVQ, the same learning rate evolution in terms of processed samples and whose convergence speed with respect to iterations is accelerated. Remark that these iterations correspond to the true "wall time" measured by an exterior observer. To this aim we propose a new system of equations (4.11).

At time $t = 0$ all versions are equal, $w^1(0) = \ldots = w^M(0) = w^{srd}$. For all $i \in \{1, \ldots, M\}$ and all $t \geq 0$, the new parallel scheme is given by

$$\begin{cases} w^i_{temp} = w^i(t) - \varepsilon_{t+1} H\left(\mathbf{z}^i_{\{t+1 \bmod n\}}, w^i(t)\right) & \\ w^i(t+1) = w^i_{temp} & \text{if } t \bmod \tau \neq 0 \text{ or } t = 0, \\ \begin{cases} w^{srd} = w^{srd} - \sum_{j=1}^{M} \Delta^j_{t-\tau \to t} \\ w^i(t+1) = w^{srd} \end{cases} & \text{if } t \bmod \tau = 0 \text{ and } t \geq \tau. \end{cases} \tag{4.11}$$

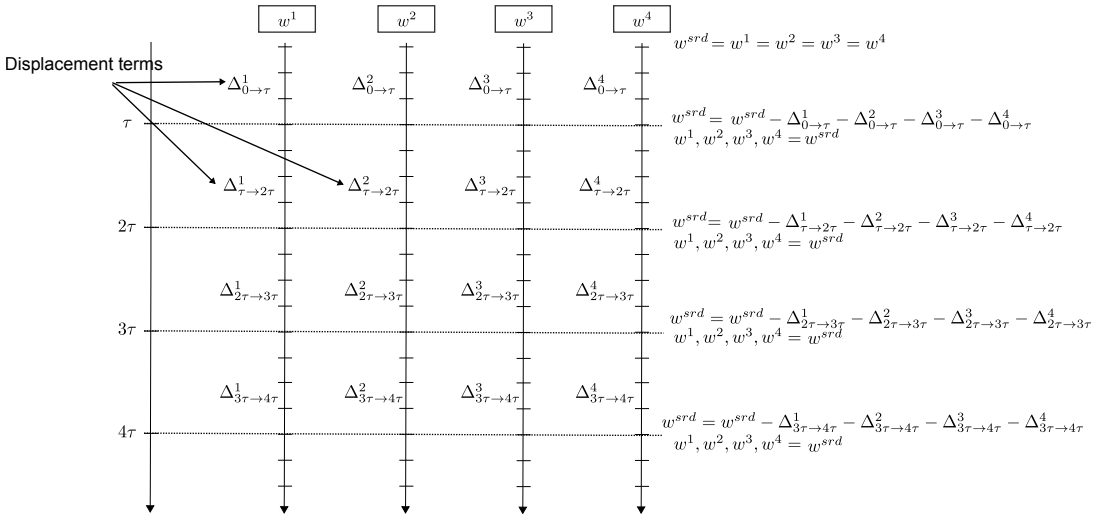

Figure 4.7: Illustration of the parallelization scheme of CLVQ procedures described by equations (4.11).

As a big picture, these iterations sum the displacement terms $\Delta^j_{\cdot\to\cdot}$ rather than using the average as shown in equation (4.10). The computation of w^{srd} described in equations (4.11) is now called "reducing phase" instead of "averaging phase". It can be implemented easily, but at the price of keeping supplementary information in the local memory of the computing entities. Precisely, iterations (4.11) require to have, for processor j at any time $t \geq 0$, the displacement term from $\lfloor t/\tau \rfloor \tau + 1$ (last reducing phase) to the current time t. With the notation above this displacement term writes $\Delta^j_{\lfloor t/\tau \rfloor \tau \to t}$. Figure 4.7 provides a synthetic representation of these iterations. The results of the simulations are displayed in the charts of Figure 4.8. The charts show that substantial speed ups are obtained with more 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.

4.3.3 A parallelization scheme with communication delays.

In the previous subsections we focused on the speed up that can be brought by parallel computing resources. However, the previous parallelization schemes did not deal with the computation of shared versions. A natural way to perform this task is to set a dedicated processor responsible for it. However, in our context no efficient shared memory is available. The shared version will be visible for other processors only through network connections. The resulting communication costs need to be taken into account in our model. To this aim we introduce delays. The effect of such delays for parallel stochastic gradient descent has been studied by Langford et al. in [96]. However, these authors consider only the case where the computing architecture is endowed with an efficient shared memory. Then, all troubles mentioned in the previous subsection (averaging different versions) do not happen. In this subsection, we improve the model of iterations (4.11) with non negligible communication costs, resulting in the following more realistic iterations (4.12).

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 $t \geq 0$,

$$\begin{cases} w^i_{temp} = w^i(t) - \varepsilon_{t+1} H\left(\mathbf{z}^i_{\{t+1 \bmod n\}}, w^i(t)\right) & \\ w^i(t+1) = w^i_{temp} & \text{if } t \bmod \tau \neq 0 \text{ or } t = 0, \\ \begin{cases} w^{srd} = w^{srd} - \sum_{j=1}^M \Delta^j_{t-2\tau \to t-\tau} & \text{if } t \bmod \tau = 0 \text{ and } t \geq 2\tau, \\ w^i(t+1) = w^{srd} - \Delta^i_{t-\tau \to t} & \text{if } t \bmod \tau = 0 \text{ and } t \geq \tau. \end{cases} & \end{cases} \tag{4.12}$$

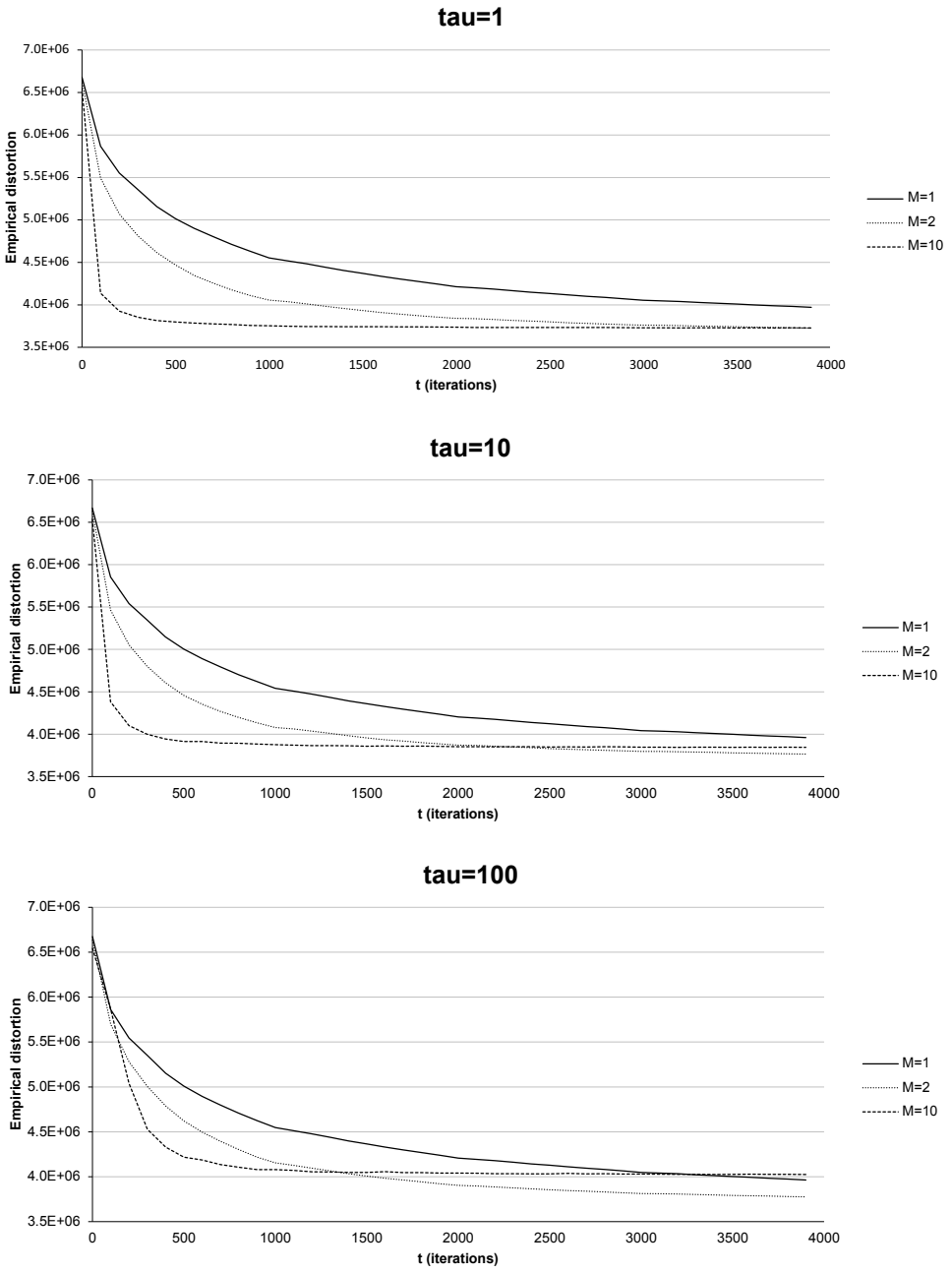

Figure 4.8: Charts of performance curves for iterations (4.11) with different number of computing entities, $M = 1, 2, 10$. The three charts correspond to different values of τ ($\tau = 1, 10, 100$).

The main difference with iterations (4.11) is that the shared version is computed asynchronously during two reducing phases. Consequently, the shared version read by the other processors is delayed. This discussion on delays is also part of the theoretical DALVQ in Chapter 3 and BlobStorage I/O in Chapter 5. In the present model, all delays are set equal to a common value τ which corresponds also to the duration between two successive reducing phases. This model is sufficient for understanding the impact of delays as shown in the charts of Figure 4.10. The Figure 4.9 gives a synthetic view of the iterations (4.12).

The performance curves resulting from the simulations of this distributed scheme are displayed in Figure 4.10. Remark that in our experiments small delays ($\tau = 1$ or 10) do not have severe impact, since the results are similar to the non-delayed model reported in Figure 4.8. However, large delays $\tau = 100$ can have a significant impact and prevent the distributed scheme to bring speed up in convergence with more computing resources.

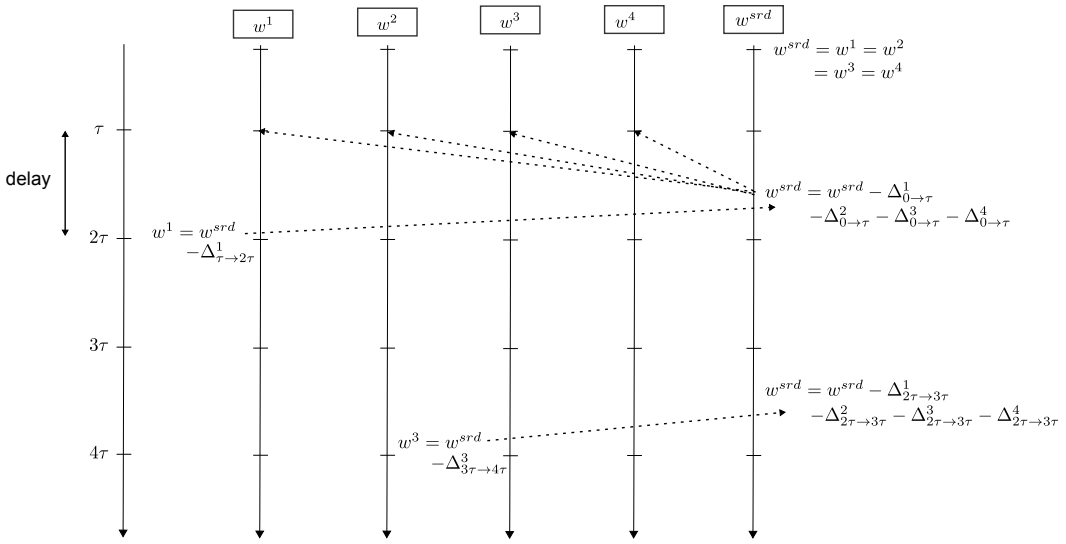

Figure 4.9: Illustration of the parallelization scheme described by equations (4.12). The reducing phase is only drawn for processor 1 where $t = 2\tau$ and processor 4 where $t = 4\tau$.

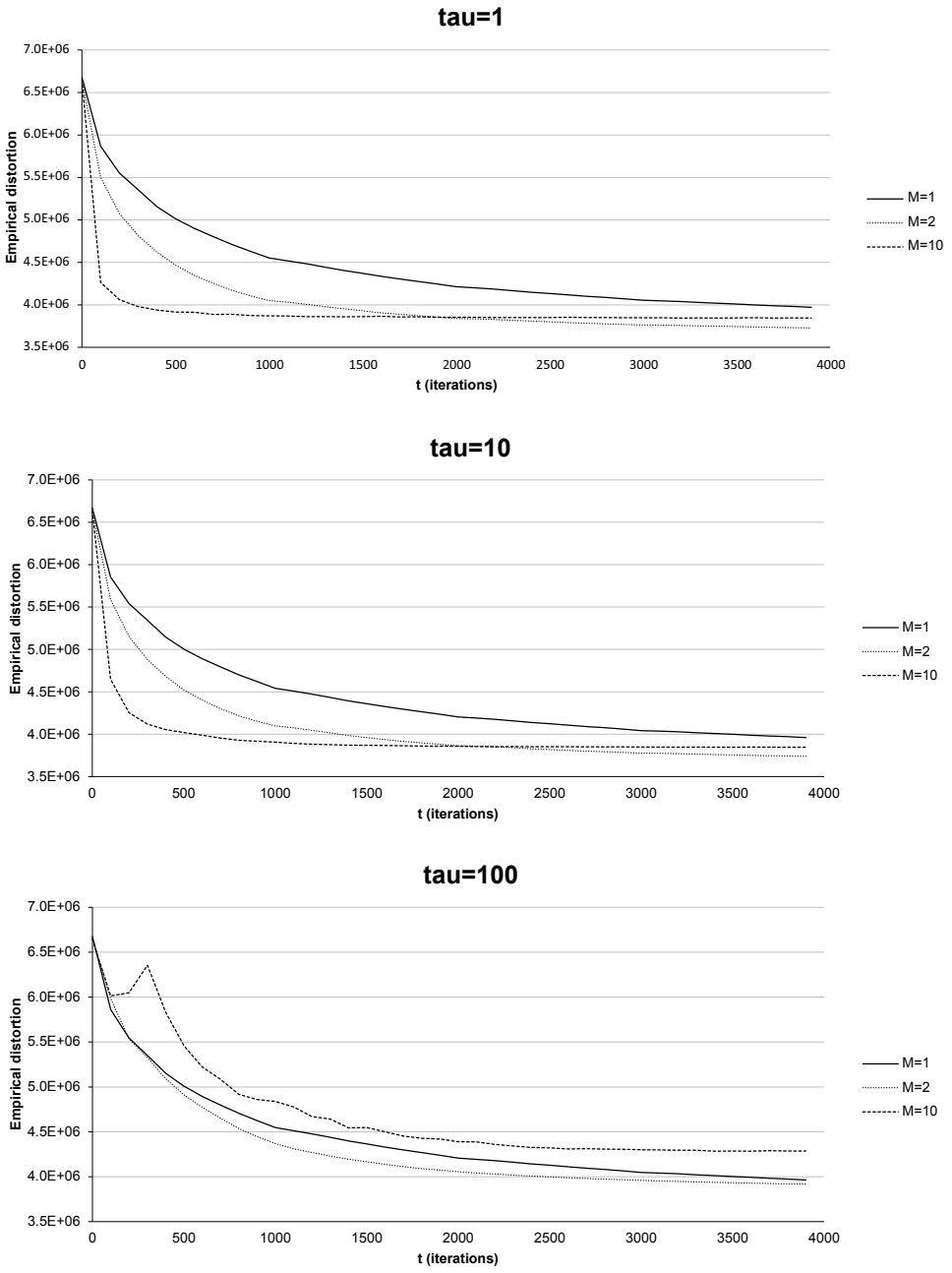

Figure 4.10: Charts of performance curves for iterations (4.12) with different number of computing entities, $M = 1, 2, 10$. The three charts correspond to different values of τ ($\tau = 1, 10, 100$).

5 The software project CloudDALVQ

5.1 Introduction

In this chapter we introduce and deliver some results obtained by the software project CloudDALVQ [1]. This project is built on top of the Cloud Computing platform Microsoft Windows Azure and implements the DALVQ quantization algorithms introduced in Chapter 3. In the latter chapter the descriptions of the procedures were theoretical and they were presented only through mathematical equations. However, they have been designed from the beginning for distributed architectures such as Cloud Computing platforms. Indeed, the DALVQ models in Chapter 3 introduced non negligible and unpredictable delays (see equations 3.12, Section 3.3) that will picture essential aspects of communication in Cloud Computing platforms. The Chapter 4 has been written to attempt to build a connection between the theoretical Chapter 3 and the present chapter which includes more practical software matters. CloudDALVQ implementation makes a great use of the results obtain in Chapter 4 where parallel implementations of DALVQ have been tested and discussed in general (i.e., independently of the distributed architecture). CloudDALVQ is an opensource project released under the new BSD license and written in C#/.NET. The project is also built using the opensource project Lokad.Cloud [2] which provides an O/C (Object to Cloud) Mapper and an execution framework that leverages low level technicalities of Windows Azure.

This chapter is organized as follows. The first following section is an overview of the emerging concept of Cloud Computing. We give there a definition and explain various aspects of these modern computing technologies. Next, Section 5.3 is a short presentation of the platform Microsoft Windows Azure. We describe in this section the main abstractions brought by Windows Azure and Lokad.Cloud. Section 5.4 focuses on the software architecture of CloudDALVQ and many connections are drawn with Chapter 4. The study of the scalability of CloudDALVQ is presented in Section 5.5. The experiments are similar in spirit to those made on Section 4.3. Precisely, we investigate the scalable gain of time execution brought by DALVQ algorithms compared to a single execution of the CLVQ method. Finally, the last

1. http://code.google.com/p/clouddalvq/
2. http://lokad.github.com/lokad-cloud/

Section 5.6 is a benchmark between `CloudDALVQ` and the distributed implementation of the popular Lloyd's method (or batch k-means) designed by Durut and Rossi [42].

5.2 Overview of Cloud Computing

Cloud Computing is the emerging paradigm of computing as a utility. This term is fuzzy and might refer to very different things. From our point of view the best definition has been provided by Armbrust et al. in [7]: Cloud Computing *refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services.* In this chapter we will only consider the pay-as-you-go services with user shared resources sometimes gathered in the terminology Public Cloud as opposed to Private Cloud which stands for an infrastructure dedicated to the IT of a single organization. In the following, a cloud platform (or cloud for short) will denote the sum of the hardware and software of a Public Cloud solution. The main idea under the concept of cloud is the elasticity of (appearing) infinite computing and storage resources quickly available on demand.

The phrase "software as a service" (SaaS), as explained in Vecchiola et al. [95] refers to *solutions that are at the top end of the Cloud Computing stack and they provide end users with an integrated service comprising hardware, development platforms, and applications.* For example the popular Google's webmail `Gmail` is based on a SaaS model. The application is accessible for the user through the internet as opposed to traditional desktop applications (e.g. `MS Excel 2010`). At a much low-level computer vision we have the "infrastructure as a service" (IaaS) which provides low-level hardware and the users may control a large part of the software environment. However, the automation of resource provisioning is not easy due to few infrastructure abstractions. `Amazon EC2` is one of the most popular IaaS offer on the market. The third area of "as a service" paradigm is platform as a service (PaaS) that can be thought as an intermediate abstraction level between IaaS and SaaS. As defined by Vecchiola et al. in [95] a PaaS solution provides *an application or development platform in which users can create their own application that will run on the cloud.* A PaaS solution comes with a framework and a set of APIs to build the application for the cloud. It also comes with a portal that will help the developer to run the application on top of a fully abstracted infrastructure. In both PaaS and IaaS approaches the elasticity of resources is guaranteed by a fine use of virtualization tools which help to perform quick replications of virtual machines (abbreviated in VMs, see for instance Craig [33]). This aspect is therefore much more abstracted in a PaaS offer. The main actors

proposing PaaS solutions are Google AppEngine and Microsoft Windows Azure.

Cloud Computing solutions have a deep impact on economic models. The pay-as-you-go concept brings to a company the opportunity to pay only for what it uses. As an example, it can allow 100 servers for few hours to respond to a peak traffic on their web application and one small server after that. Indeed, the cost of 100 hours of CPU with one machine is the same than using 100 machines for one hour. The Cloud Computing short-term elastic ability of downsizing and upsizing resources available will make the risk of underprovisioning disappear and will reduce the costs due to overprovisioning. Amazon CEO Jeff Bezos in [12] tells the story of Animoto, a company that used intensively the elasticity of Amazon EC2. Their resource needs increased in three days from 50 Amazon EC2 instances to 3500. This type of problem is also encountered in the case of the software company Lokad which delivers various forecasting SaaS applications (Salescast, Callcast, etc.) for business intelligence. Lokad's forecasting engine is deployed on top of Microsoft Azure (PaaS) and makes complex Machine Learning computations which require a lot of computing resources. Furthermore Lokad is bound by the fact to deliver its forecasts within an hour. Therefore some peak of resources consumption naturally appear in rush hours. The automated provisioning service adapts the resources by deploying hundreds of VMs instances in few minutes.

Scientific computing may also make a great use of Cloud Computing paradigm. Indeed, scientists need to perform large scale experiments requiring a large number of computing resources which can be easily provided by Cloud Computing platforms. Usually, they use super computers or clusters to run such research works. However, these solutions are inevitably linked to massive costs of hardware and maintenance. Such approaches are also difficult to setup and operate for a physicist or a mathematician who is not a informatics engineer. The Cloud Computing paradigm offers a new model for the scientist community. Indeed, the storage and computing resources are dynamically provisioned as needed and the experimentation costs could be decreased significantly. CloudDALVQ is an example of a scientific project for large scale experiments based on a PaaS cloud platform. Its goal consists in testing some new quantization algorithms highly scalable.

5.3 Windows Azure and Lokad.Cloud

5.3.1 Services, queues and workers

Microsoft Windows Azure (or simply Azure) is a cloud offer classified as Platform As A Service (PaaS, see Section 5.2 above). It provides elastic computing

resources to deploy internet-scale applications. The basic computing units provided by `Azure` are Virtual Machines hosting instances of roles. There are two types of role. Firstly, the web role acting as a frontend for web applications. Secondly, the worker role used for intensive background computing tasks whose instances can be deployed on many workers. The `CloudDALVQ` project uses only one web role monitoring the application and one worker role with tens of workers. In order to simplify the discussion, VMs and workers can be assimilated. `Azure` provides several types of worker instances as shown in Table 5.1. In all our experiments, we used `Small` computing instances.

Instance	CPU	Memory	Disk	I/O Perf.	Cost Per Hour
Extra Small	1.0 GHz	768 MB	20 GB	Low	$0.05
Small	1.6 GHz	1.75 GB	225 GB	Moderate	$0.12
Medium	2 x 1.6 GHz	3.5 GB	490 GB	High	$0.24
Large	4 x 1.6 GHz	7 GB	1,000 GB	High	$0.48
Extra Large	8 x 1.6 GHz	14 GB	2,040 GB	High	$0.96

Table 5.1: Compute instance details provided by Microsoft on July 2011. `http://www.microsoft.com/windowsazure/compute/`.

The logic of the worker role is all gathered in the implementation of several `QueueServices`. These `QueueServices` are abstractions brought by `Lokad.Cloud` and a part of a message delivery mechanism using distributed queues. Each `QueueService` applies its subroutine on multiple contexts, potentially in parallel using multiple VMs. This design can be thought of as a Single Program Multiple Data (see for instance [34]). Each context is described in a job message, and all job messages of a specific `QueueService` are kept in a dedicated queue. This queue can store a potentially large amount of small messages. A VM runs a given `QueueService` as follows: it starts with unqueuing one message; it applies the `QueueService` subroutine using the context given by the message, then it may push job messages for one or multiple other `QueueServices`; and finally, it deletes the message from the initial queue. Each `QueueService` can be executed by an arbitrary number of VMs, while each VM can alternatively endorse all the `QueueServices` logic according to an internal scheduler. Figure 5.1 provides an illustration of the usage of `QueueServices` for projects built on top of `Lokad.Cloud`.

Using Queues to communicate helps building loosely coupled components and mitigates the impact of individual component failures. Messages stored in a queue are guaranteed to be returned at least once, but possibly several times: this requires one to design idempotent jobs. When a message is unqueued by the so-called `QueueService` and processed asynchronously by the worker running this `QueueService`, the message is not deleted but it becomes invisible for other work-

ers. If a worker fails to complete the corresponding job (because it throws some exception or because the worker dies), the message becomes available after a certain period of time. Through this process, one can make sure that no job is lost because of e.g., a hardware failure. This distributed queue architecture enable large scale parallelization scheme (see for instance Li [69]).

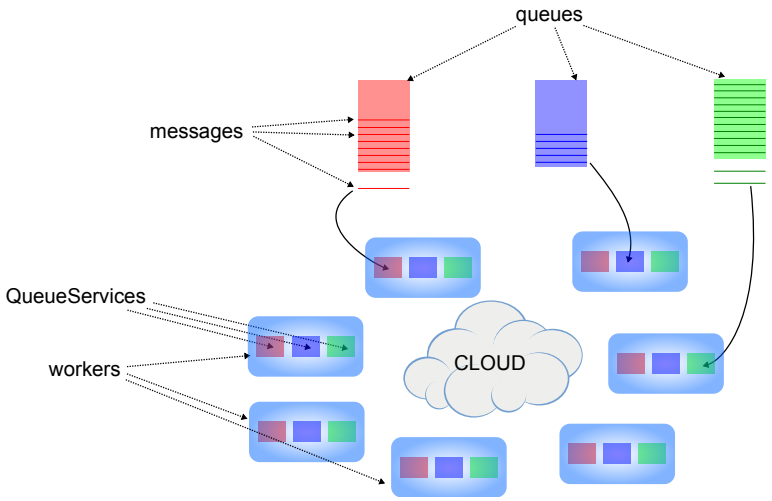

Figure 5.1: Illustration of the abstractions of distributed queues (`QueueServices`) provided by the opensource project `Lokad.Cloud`.

5.3.2 `BlobStorage`

`Azure`'s Blob Storage (`BlobStorage`) allows applications to store large objects, up to 50 GB each. It has been designed to cope with massively parallel read access. The computing resources are provided by workers while `BlobStorage` can be used to implement communication between workers. This storage is accessible thought `http` based-REST API (for more information on REST API we refer the reader to Fielding [89]). The acronym BLOB stands for Binary Large OBjects and sometimes can be viewed as an alternative to relational `SQL` databases. `BlobStorage` is based on a key/value system allowing applications to store values by key. Precisely, the key is contained in a string referencing the binary object value. The string contains also a timestamp (etag) that indicates the last write action on this blob. In addition of Get and Put methods, blobs whose key shares a given prefix can be listed. Optimistic non locking atomic read-modify-write operations can be implemented using a timestamp matching condition: a write succeed if and only if the timestamp of the storage matches the one provided by the write operation.

In `CloudDALVQ` project we used the object-to-cloud mapper (O/C) provided by `Lokad.Cloud` which is a very convenient abstraction layer to communicate with the `BlobStorage`. From VMs hosted in Microsoft datacenter we measured a rate from 4 Mb/s to 8Ms/s for reading (i.e., to get a blob) while for writing (put a blob) the rate is about 2Mb/s. These figures are really instable and might vary a lot. For further reading in the `BlobStorage` the reader is referred to to Durut and Rossi [42, supplementary materials] and to Calder [29]. Some experiments on I/O costs for the `BlobStorage` have been carried out by `AzureScope` [52] that produced relatively different results. We are aware that the O/C mapper of `Lokad.Cloud` may also explain the differences between these results. The Figure 5.2 provides an illustration of exchanges that can be made through the `BlobStorage`.

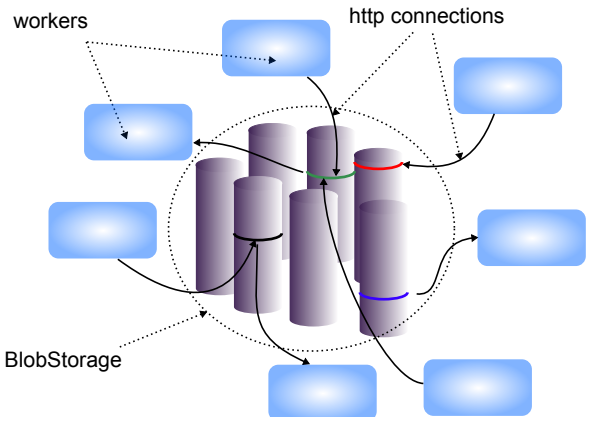

Figure 5.2: Illustration of the `BlobStorage` allowing communication between workers. The Get and Put methods are supported by `http` request and is represented with solid black lines (heading to workers for Get method and to storage for Put method).

5.4 The implementation of `CloudDALVQ`

The implementation is based on two main `QueueServices` (see previous section): the `ProcessService` and the `ReduceService`. These services are based on the main idea brought in Chapter 4: averaging concurrent local versions w^j's does not provide speed up for the convergence of the distributed system. A better approach consists in computing the sum of all displacement terms as shown in Subsection 4.3.2.

The core logic of CloudDALVQ is embedded in the implementation of the ProcessService. As a big picture, we have multiple instances of the ProcessService, M instances following notation of Chapters 3 and 4 and each instance performs a CLVQ execution. Meanwhile, the displacement terms introduced in Subsection 4.3.2, namely the $\Delta^j_{\cdot\to\cdot}$'s, are sent to the BlobStorage. The ReduceService is hosted by a dedicated worker (one instance for now). Its task consists in retrieving the multiple displacement terms and computing the shared version of the prototypes, w^{srd}. This shared version is the common reference version retrieved by the multiple instances of the ProcessService which use it to update their local version. The ReduceService implements the action described by the second inner braced equations (4.11) in Subsection 4.3.2 Chapter 4. The Figure 5.4 provides a synthetic overview of the "reducing task" assigned to the ReduceService, while Figure 5.3 gives a representation of the communication of the ProcessService instances and the ReduceService instance through the BlobStorage. The implementation of the ProcessService is more complicated than the ReduceService one and is discussed in the next paragraph.

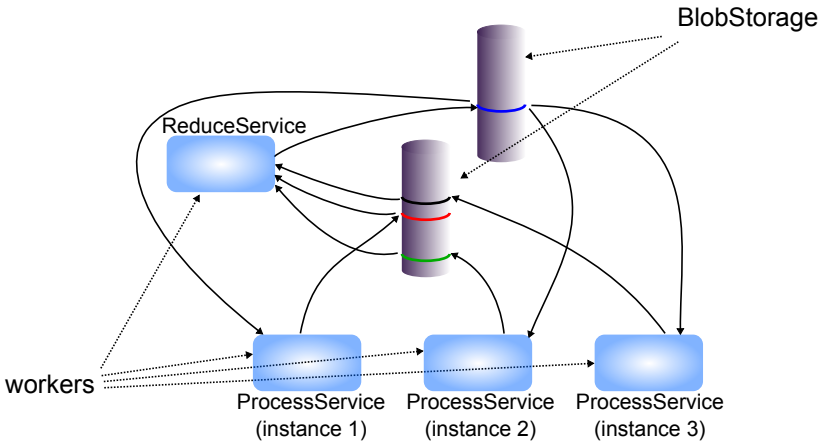

Figure 5.3: Overview of the interaction between ProcessService instances and the instance of ReduceService. All communications are made through the BlobStorage: ReduceService gets the blobs put by the ProcessService instances while they retrieve the computation of the ReduceService.

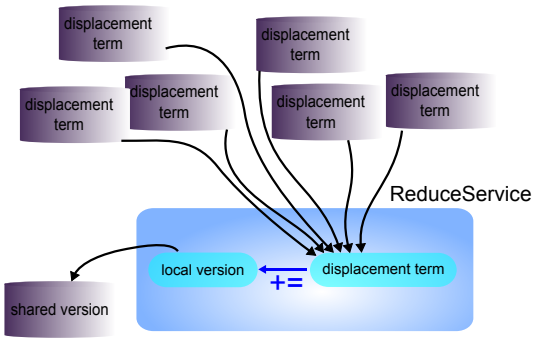

Figure 5.4: Overview of the `ReduceService` implementing the "reducing task". This `QueueService` builds the shared version by computing the sum of all displacement terms sent by `ProcessService` instances in the `BlobStorage`.

The `ProcessService` performs simultaneously three tasks using the following threads: process thread, push thread, and pull thread. The process thread is a CPU-intensive thread while push thread and pull thread are dedicated to communications with the `BlobStorage`. This multi-threaded approach avoids synchronization points and non-desired time execution penalties resulting from the I/O costs with `BlobStorage`. We put the emphasis on the fact that this parallelization is made inside the VMs, at CPU level. The multi-threaded execution of the `ProcessService` is based upon the `Task Parallel Library` provided in the `.NET framework 4.0` (see for instance Freeman [46]). More precisely, the pull thread is responsible for providing the latest shared version to process thread using a local memory buffer (read buffer). `BlobStorage`'s timestamps (etags) are used to detect new shared versions sent by the `ReduceService` and available. In other words pull thread and process thread communicate using a Producer/Consumer design pattern: pull thread is the producer whereas process thread is the consumer. For more details on Producer/Consumer design patterns we refer the reader to Grand [50]. The `.NET 4.0 BlockingCollection` class is of great help to the read buffer for managing concurrency issues. The push thread is responsible for pushing to distant blobs the displacement term updates computed by the process thread. The interaction of the push thread and the process thread also uses a Producer/Consumer design pattern with a communication buffer (write buffer). However, in this context, the process thread is the producer and the push thread is the consumer. The blobs are then consumed by the `ReduceService`: this `QueueService` is notified by messages in its Queue that new updates can be used for the shared version computation. Figure 5.5 illustrates the multi-threaded logic of the `ProcessService`.

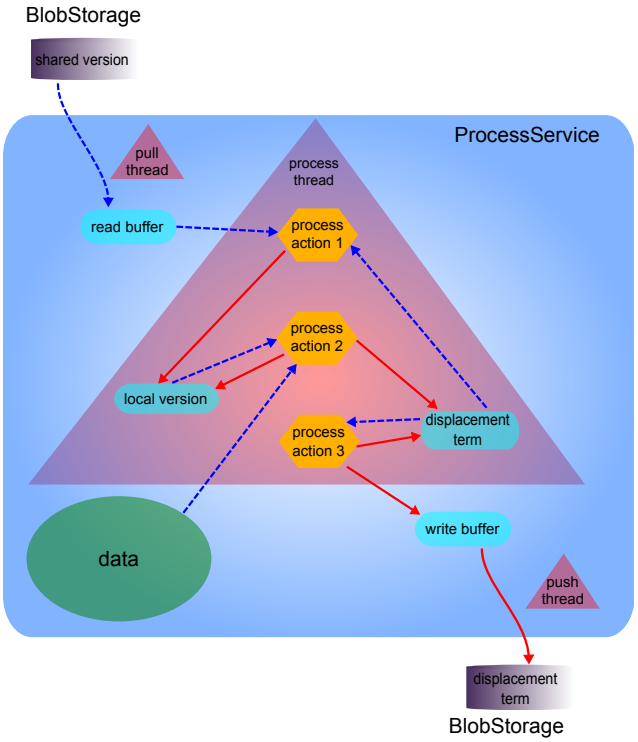

Figure 5.5: Overview of the `ProcessService`. Each triangle stands for a specific thread (process thread, push thread and pull thread). Arrows describe read/write actions: the tail of a blue dashed arrow is read by the entity at its head and the entity at the tail of a red solid arrow makes update on the entity that lies at its head. Push thread and pull thread enable communications between process thread and the `BlobStorage`. Process thread alternatively performs three actions (process action $1, 2, 3$) using a custom scheduler which does not appear in the figure. Process action 1 replaces the local version of the prototypes by the sum of the latest shared version (kept in the read buffer) and a displacement term. Process action 2 uses data to execute CLVQ iteration and updates both the local version and the displacement term. Process action 3 moves the displacement term to a dedicated buffer (write buffer) and pushes it content to the `BlobStorage`.

5.4.1 Evaluations

In the experiments of Section 5.5 and Section 5.6, the performance of clustering algorithms are measured during their executions. However, the algorithms should not be slowed down by these measurements. Consequently, `CloudDALVQ` uses the fact that the `BlobStorage` has been designed to cope with multiple reading actions. A new `QueueService` has been implemented: the `SnapshotService` and a dedicated worker is deployed to host it. The `SnapshotService` keeps making deep copies of the shared version blob—the snapshots—and stores them in other persistent blobs. We recall that the blob containing the shared version is constantly modified by the `ReduceService`. Therefore it is necessary to make copies to track back its evolution. Once the multiple `ProcessingService` instances stop processing CLVQ iterations, i.e., when the stopping criterion is met, then the evaluation processus starts: for all snapshots, many instances of the `EvaluationService` (another `QueueService`) compute the empirical distortion given by equation (4.2) (or its normalized version (4.6)). In order to perform the evaluations of the snapshots the synthetic data are generated again by using new instances of random generators. Therefore, no data is broadcasted through the network that would have been a massive bottleneck for the evaluations of the algorithms. This task is still highly CPU consuming. However, thanks to the elasticity of the VMs allocations, it can be completed with reasonable delays.

5.5 Scalability of CloudDALVQ

In this section our objective is to prove that `CloudDALVQ` brings a substantial speed up which corresponds to the gain of wall-clock time execution brought by more computing resources compared to a sequential execution. Consequently the experiments presented in this section are similar to those provided in Chapter 4. In this previous chapter we proved that a careful attention should be paid on the parallelization scheme, but without specifications on the architecture. We recall that the experiments of Chapter 4 simulated a generic distributed architecture. In this section we present some experiments made with the true cloud implementation `CloudDALVQ`.

The following settings attempt to create a synthetic but realistic situation for our algorithms. We load the local memory of the workers with data generated using the splines mixtures random generators defined in Section 4.2. Using the notation introduced in this section, each worker is endowed with $n = 10000$ vec-

torial data that are sampled spline functions with $d = 1000$. The benefit of using synthetic data for measuring the performance of clustering algorithms has been discussed in Subsection 5.4.1. The number of clusters is set to $\kappa = 1000$ while there are $G = 1500$ centers in the mixture and the number of knots for each spline χ is set to 100.

Similarly to Section 4.3, we investigate the speed up ability of `CloudDALVQ`. Remind that the number of samples in the total dataset is equal to nM. Thus, this number is varying with M (the number of `ProcessService` instances). Consequently, evaluations are performed with the normalized quantization criterion given by equation (4.6).

The following Figure 5.6 shows the normalized quantization curves for $M = 1, 2, 4, 8, 16$. We notice that `CloudDALVQ` has a good scalability property for $M = 1, 2, 4, 8$: the algorithm converges more rapidly because it has processed much more points as shown at the end of the section. Therefore, in term of wall clock time, the DALVQ with $M = 2, 4, 8$ clearly outperforms the single execution of CLVQ ($M = 1$). However we remark that troubles happen with $M = 16$. A solution has been found to overcome the problem as explained in the subsequent paragraph.

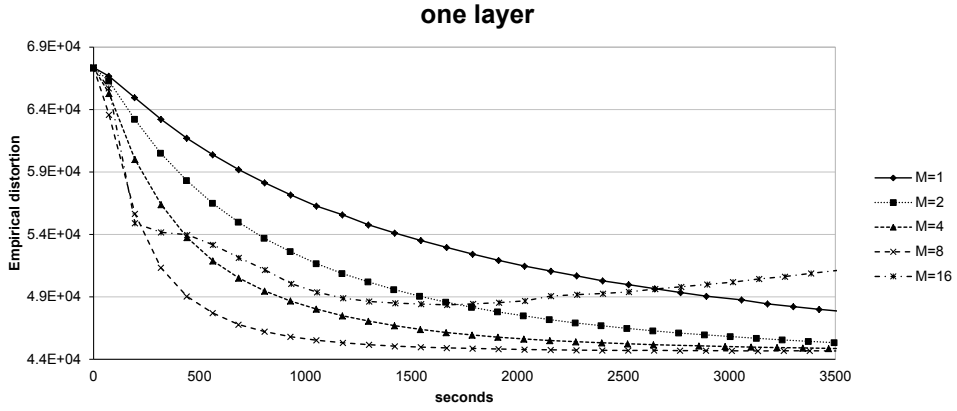

Figure 5.6: Normalized quantization curves with $M = 1, 2, 4, 8, 16$. `CloudDALVQ` has good scalability properties up to $M = 8$ instances of the `ProcessService`. Troubles appear with $M = 16$ because the `ReduceService` is overloaded.

The performance curve with $M = 16$ of Figure 5.6, shows a disastrous situation where the algorithm does not work. Actually, the troubles arise from the fact that the `ReduceService` is overloaded. Actually, too many `ProcessService` instances communicate their results (displacements terms) and fill the Queue associated to the `ReduceService` with new jobs resulting in a freezing overload. Once again the elasticity of the cloud enables us to allow more resources and to create a better scheme by distributing the reducing task workload on multiple workers. Indeed, we set an intermediate layer in the reducing process by introducing two new `QueueServices`: `PartialReduceService` and `FinalReduceService`. We deploy $\lfloor \sqrt{M} \rfloor$ instances of `PartialReduceService` that will be responsible for computing the sum of the displacement, but only for a certain fraction of `ProcessService` instances. Then, the only instance of `FinalReduceService` retrieves all intermediate sums of displacement terms made by the `PartialReduceService` instances and computes the shared version. Figure 5.8 is a synthetic representation of the new layer, which is introduced to leverage the difficulty brought by the overload of the initial `ReduceService`. In Figure 5.7 we plot the performance curves with `PartialReduceService` instances. We remark that a good speed up is obtained up to $M = 32$ `ProcessService` workers. However the algorithm does not scale up to $M = 64$. The cause of this problem has not been found yet.

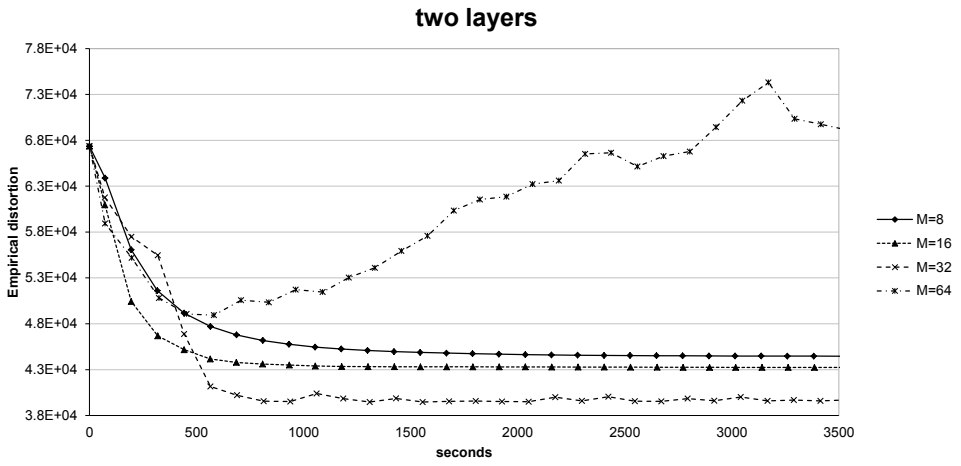

Figure 5.7: Normalized quantization curves with $M = 8, 16, 32, 64$ instances of `ProcessService` and with an extra layer for the so called "reducing task". `CloudDALVQ` has good scalability properties for the quantization performance up to $M = 32$. However, the algorithm behaves badly when the number of computing instances is raised to $M = 64$.

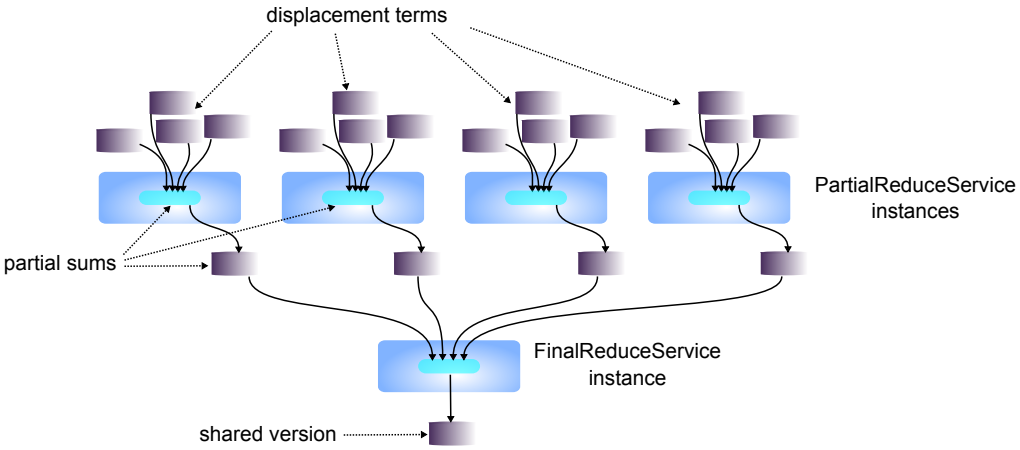

Figure 5.8: Overview of the reducing procedures with two layers: the `PartialReduceService` and the `FinalReduceService`.

In our context, the scalability is also the ability of the distributed algorithm to process growing volumes of data gracefully. The experiments reported in the Figure 5.9 investigate this processing ability more precisely, by counting the samples that have been processed for the computation of the shared version. The graph shows the results with one or two layers for the reducing task and with different values of M, namely $M = 1, 2, 4, 8$ for the first case and $M = 8, 16, 32, 64$ for the second one. The curves appear linear, which proves that the algorithms behaves well. The slopes of the multiple curves, characteristic of the processing ability of the algorithm, exhibits a good property: the slope is multiplied by 2 when the number of processing instances M is multiplied by 2. This proves that the algorithm has a good processing scalability as we hoped for. This is confirmed by Figure 5.10, showing a linear curve for the number of points processed in a given time span (3600 seconds) using different computing instances M. We remark that the scalability in terms of data processed is still good up to $M = 64$ while, the algorithm does not work well in term of accuracy (i.e., quantization level). Furthermore, the introduction of the second layer does not slow down the execution even for intermediate values of M ($M = 8$) as proved by the charts of Figure 5.9.

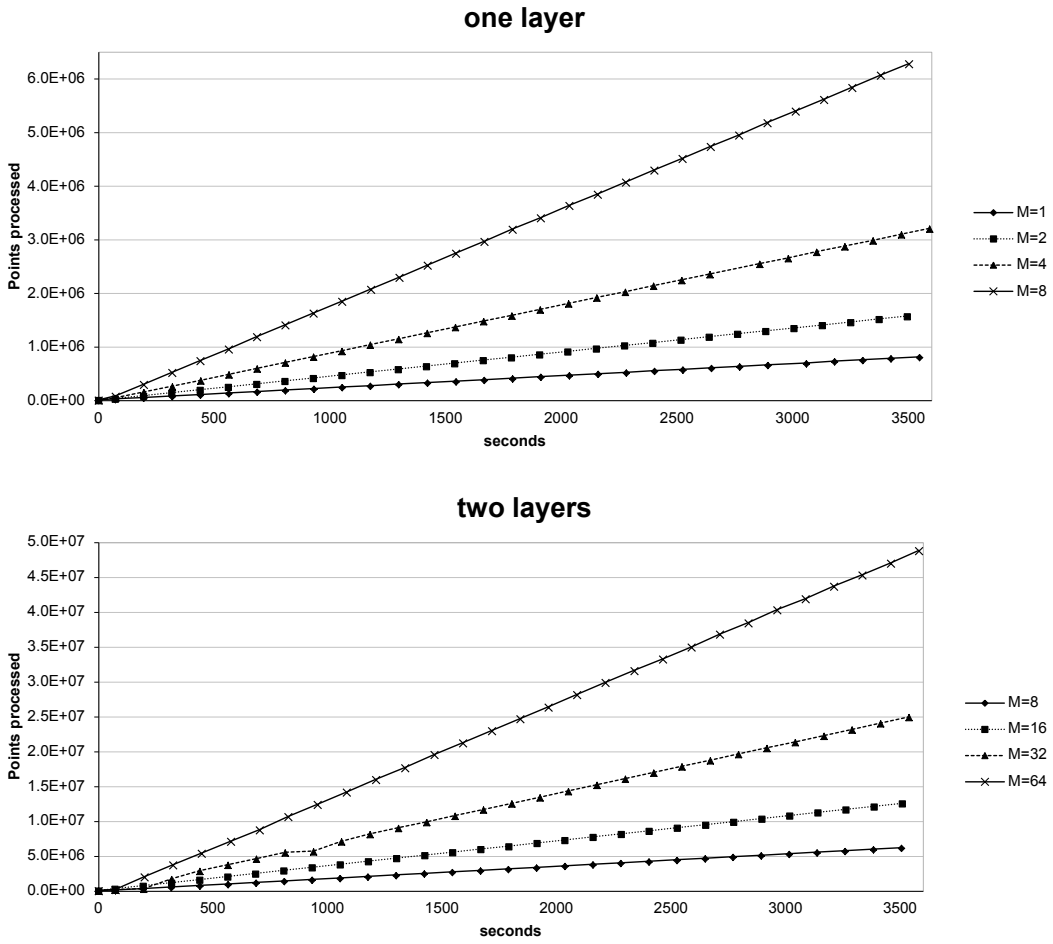

Figure 5.9: These charts plot the number of points processed with the time. The top chart shows the curves associated to $M = 1, 2, 4, 8$ with one layer for the reducing task whereas in the bottom a second layer is added and $M = 8, 16, 32, 64$.

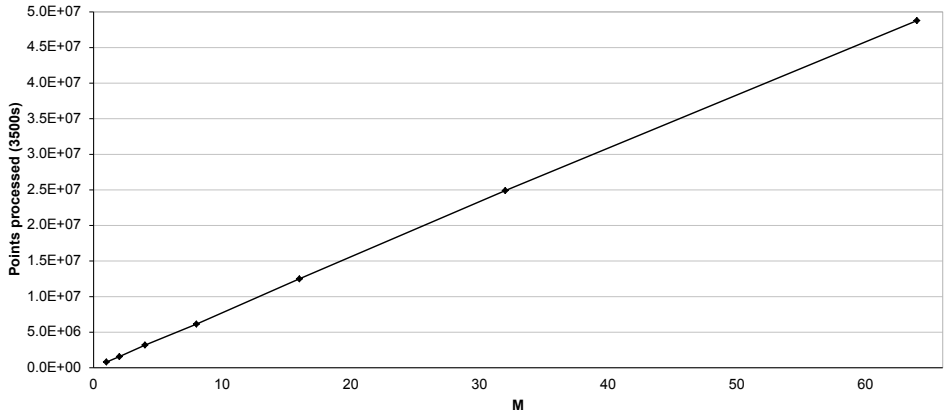

Figure 5.10: This chart plots the number of points processed in a given time span (3600 seconds) for different values of M ($M = 1, 2, 4, 8, 16, 32, 64$). The reducing task is composed of one layer for $M = 1, 2, 4$ and two layers for $M = 8, 16, 32, 64$. We observe a linear scalability in term of point processed by the system up to $M = 64$ computing instances.

5.6 Competition with batch k-means

The experiments provided in the previous section prove the benefit of using `CloudDALVQ` for large scale quantization tasks compared to the usual CLVQ algorithm. However such experiments do not prove that `CloudDALVQ` is a good clustering method itself. In this section we present a comparison of `CloudDALVQ` with the batch k-means, which is a standard clustering method. More precisely, we compare `CloudDALVQ` with the `Azure` implementation of a distributed batch k-means developed by Durut and Rossi in [42] (denoted as `CloudBatchKMeans` in the following). We work with a fixed large synthetic dataset and our goal is to provide a summary with a reduced set of prototypes. The two algorithms are compared with the evolution of the quantization error using intermediate values of the prototypes.

`CloudBatchKMeans` follows Dhillon and Modha's principles for parallel batch k-means [38]. The batch k-means consists in alternating two phases (after proper initialization). In the assignment phase, the algorithm computes the Euclidean distance between each of the data point and each of the κ prototypes. Each sample is assigned to its closest prototype. In the recalculation phase, each prototype is recomputed as the average of the data points assigned to it in the previous phase. Distance calculations are intrinsically parallel, both over the data points and the prototypes. It is therefore natural to split the computational load by allocating

disjoint subsets of data points to the workers. Each worker, called a mapper following Dean and Ghemawat's terminology [37], computes a full assignment phase for its fraction of the dataset. Next the mapper builds a new prototypes version, where each component is updated as the average of the data points assigned to this component. After that, the workers synchronize and average these prototypes versions. The averaged prototypes are broadcasted to all the mappers, which starts a new iteration on reception. As explained in [38], a parallel batch k-means on the different data subsets and a sequential execution of a batch k-means on the whole dataset perform the same computations and return the same output.

The number of mappers in `CloudBatchKMeans` is the number of processing units. Therefore it is natural to compare `CloudDALVQ` and `CloudBatchKMeans` with the same dataset and the same number of `ProcessingService` instances and mappers M. Similarly to Section 5.5 the local memory of a worker is loaded with $n = 500000$ vectorial data whose dimension d is set to 100. The benefit of using synthetic data for measuring the performance of clustering algorithms has been discussed in Subsection 5.4.1. The number of clusters is set to $\kappa = 5000$ while the number of centers in the mixture G equals 7500 and the number of knot for each splines χ equals 100. We set the learning rate ε_t (see Section 3.2) to a constant value $\varepsilon = 1.0e^{-2}$.

Figure 5.11 shows the curves of clustering performance for the two procedures in competition. We report there three experiments with various sizes of datasets nM ($M = 8, 16, 32$). We see that `CloudDALVQ` clearly outperforms `CloudBatchKmeans`. In all cases ($M = 8, 16, 32$) the algorithm takes less time to reach a similar quantization level. The quantization curve of `CloudBatchKMeans` corresponds to a step function. Indeed, the reference version used for the batch k-means evaluations is the version built during the synchronization phase and is modified only there. To conclude, in this situation `CloudDALVQ` clearly outperforms `CloudBatchKMeans`. Therefore, this implementation seems to be a competitor to existing solutions for large scale clustering jobs, thanks to its scalability and its mathematical performance.

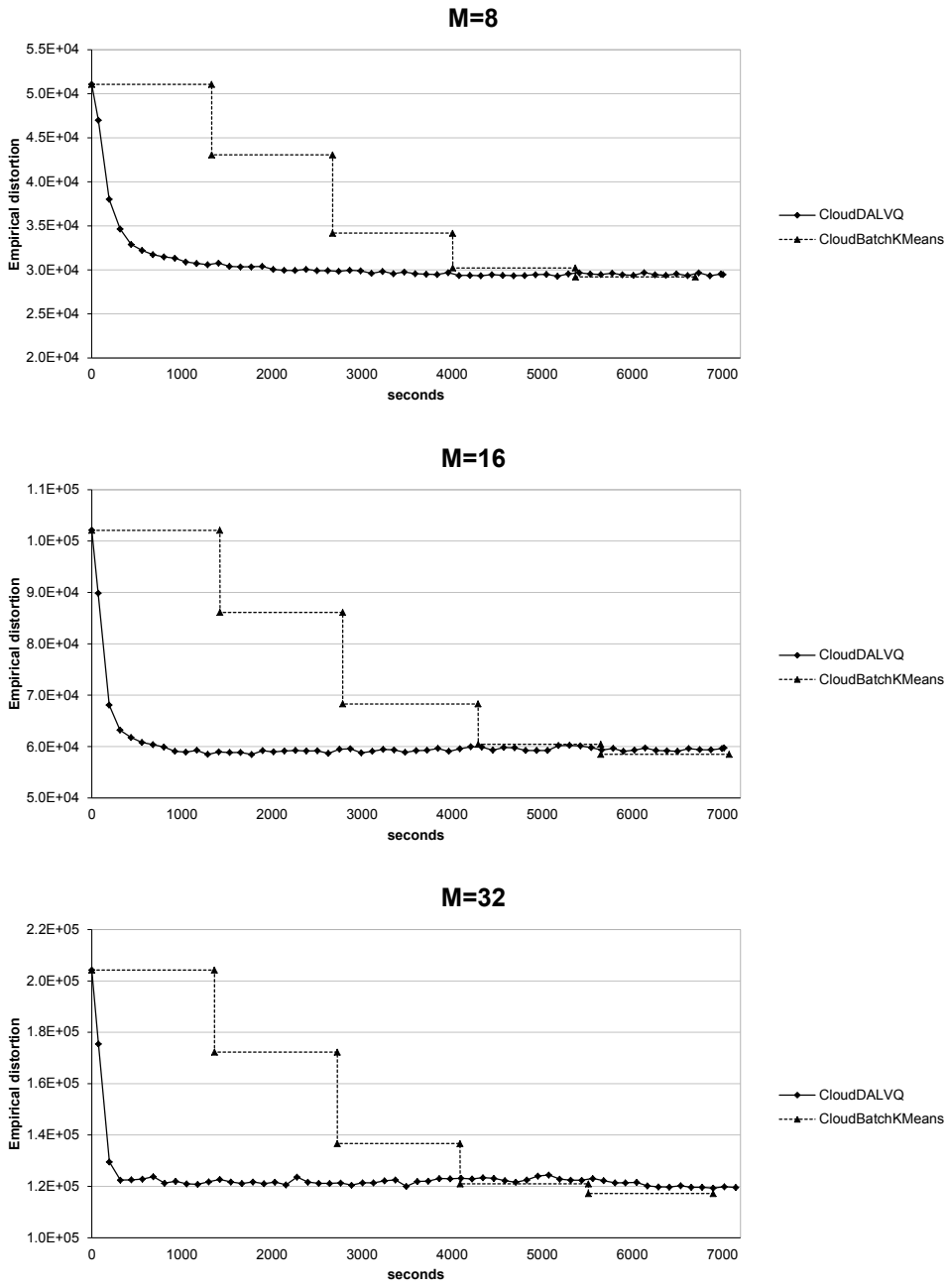

Figure 5.11: These charts report the competition between `CloudDALVQ` and `CloudBatchKMeans`. The graphs show the empirical distortion of the algorithm over the time. The empirical distortion is computed using the shared version for `CloudDALVQ` while it is computed during the synchronization phase (all mappers receiving the same prototypes) for `CloudBatchKMeans`. In these experiments `CloudDALVQ` outperforms `CloudBatchKMeans`: the same quantization level is obtained within a shorter period.

List of Tables

List of Figures

Bibliography

[1] Antos A. Improved minimax bounds on the test and training distortion of empirically designed vector quantizers. *IEEE Transactions on Information Theory*, 51:4022–4032, 2005.

[2] E. A. Abaya and G. L. Wise. Convergence of vector quantizers with applications to optimal quantization. *SIAM Journal on Applied Mathematics*, 44:183–189, 1984.

[3] C. Abraham, P. A. Cornillon, E. Matzner-Løber, and N. Molinari. Unsupervised curve clustering using b-splines. *Scandinavian Journal of Statistics*, 30:581–595, 2003.

[4] P. H. Algoet. The strong law of large numbers of sequential decisions under uncertainty. *IEEE Transactions on Information Theory*, 40:609–633, 1994.

[5] P. H. Algoet and T. M. Cover. Asymptotic optimality and asymptotic equirepartition properties of log-optimum investment. *The Annals of Probability*, 16:876–898, 1988.

[6] A. Antos, L. Györfi, and A. György. Improved convergence rates in empirical vector quantizer design. *IEEE Transactions on Information Theory*, 51:4012–4022, 2005.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, February 2009.

[8] P. L. Bartlett, T. Linder, and G. Lugosi. The minimax distortion redundancy in empirical quantizer design. *IEEE Transactions on Information Theory*, 44:1802–1813, 1998.

[9] A. Benveniste, M. Métivier, and P. Priouret. *Adaptive algorithms and stochastic approximations*. Springer-Verlag, 1990.

[10] S. Bermejo and J. Cabestany. The effect of finite sample size on on-line k-means. *Neurocomputing*, 48:511–539, 2002.

[11] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., 1989.

[12] Jeff Bezos. Amazon.com ceo jeff bezos on animoto. Blog, April 2008.

[13] G. Biau, K. Bleakley, L. Györfi, and G. Ottucsák. Nonparametric sequential prediction of time series. *Journal of Nonparametric Statistics*, 22:297–317, 2010.

[14] G. Biau, L. Devroye, and G. Lugosi. On the performance of clustering in hilbert spaces. *IEEE Transactions on Information Theory*, 54:781–790, 2008.

[15] G. Biau and B. Patra. Sequential quantile prediction of time series. *IEEE Transactions on Information Theory*, 57:1664–1674, 2011.

[16] P. Billingsley. *Probability and measure*. John Wiley & Sons Inc., 3rd edition, 1995.

[17] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Decision and Control, 2005 and 2005 European Control Conference.*, 2005.

[18] D. Bosq. *Nonparametric statistics for stochastic processes: estimation and prediction*. Springer-Verlag, 1996.

[19] L. Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, 1991.

[20] L. Bottou. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.

[21] L. Bottou and Y. Bengio. Convergence properties of the k-means algorithm. In *Advances in Neural Information Processing Systems*. MIT Press, 1995.

[22] L. Bottou and Y. LeCun. Large scale online learning. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

[23] L. Bottou and Y. LeCun. On-line learning for very large datasets. *Applied Stochastic Models in Business and Industry*, 21:137–151, 2005.

[24] P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[25] L. Breiman. The individual ergodic theorem of information theory. *Annals of mathematical statistics*, 16:809–811, 1957.

[26] P. J. Brockwell and R. A. Davis. *Time Series: theory and methods*. Springer-Verlag, 2nd edition, 1991.

[27] F. Bullo, J. Cortés, and S. Martínez. *Distributed control of robotic networks*. Princeton University Press, 2009.

[28] F. Bunea and A. B. Nobel. Sequential procedures for aggregating arbitrary estimators of a conditional mean. *IEEE Transactions on Information Theory*, 54:1725–1735, 2008.

[29] B. Calder. Windows azure storage architecture overview. Blog, December 2010.

[30] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[31] G. Choquet. *Topology*. Academic Press, 1966.

[32] P. A. Chou. The distortion of vector quantizers trained on n vectors decreases to the optimum at $o_p(1/n)$. *IEEE Transactions on Information Theory*, 8:457–457, 1994.

[33] I. D. Craig. *Virtual machines*. Springer-Verlag, 2006.

[34] F. Darema. The spmd model: past, present and future. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer-Verlag, 2001.

[35] C. de Boor. On calculating with b-splines. *Journal of Approximation Theory*, 6:50–62, 1972.

[36] C. de Boor. *A practical guide to splines*. Springer-Verlag, 1978.

[37] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[38] I. Dhillon and D. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *In Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, 2000.

[39] D. Duffie and J. Pan. An overview of value at risk. *Journal of Derivatives*, 4:7–49, 1997.

[40] J. W. Durham, R. Carli, P. Frasca, and F. Bullo. Discrete partitioning and coverage control with gossip communication. In *ASME Conference Proceedings*, 2009.

[41] R. Durrett. *Probability: theory and examples*. Duxbury Press, 1990.

[42] M. Durut and F. Rossi. k-means on azure. In *LCCC: NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds*, 2010.

[43] J. C. Fort and G. Pagès. On the a.s. convergence of the kohonen algorithm with a general neighborhood function. *The Annals of Applied Probability*, (5):1177–1216, 1995.

[44] J. C. Fort and G. Pagès. Convergence of stochastic algorithms: from the kushner-clark theorem to the lyapounov functional method. *Advances in Applied Probability*, 28:1072–1094, 1996.

[45] P. Frasca, R. Carli, and F. Bullo. Multiagent coverage algorithms with gossip communication: Control systems on the space of partitions. In *American Control Conference*, 2009.

[46] A. Freeman. *Pro.NET 4 Parallel Programming in C#*. Apress, 2010.

[47] A. Gannoun, J. Saracco, and K. Yu. Non parametric prediction by conditional median and quantiles. *Journal of Statistical Planning and Inference*, 117:207–223, 2003.

[48] A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Kluwer, 1992.

[49] S. Graf and H. Luschgy. *Foundations of quantization for probability distributions*. Springer-Verlag, 2000.

[50] M. Grand. *Patterns in Java: a catalog of reusable design patterns illustrated with uml, volume 1*. John Wiley & Sons, 2nd edition, 2002.

[51] W. H. Greub. *Linear algebra*. Springer-Verlag, 4th edition, 1975.

[52] Extreme Computing Group. Azurescope. http://azurescope.cloudapp.net/.

[53] L. Györfi, M. Kohler, A. Krzyżak, and H. Walk. *A distribution-free theory of nonparametric regression*. Springer-Verlag, 2002.

[54] L. Györfi and G. Lugosi. *Strategies for sequential prediction of stationary time series*, chapter 11, pages 225–248. Kluwer, 2001.

[55] L. Györfi, G. Lugosi, and F. Udina. Non parametric kernel based sequential investment strategies. *Mathematical finance*, 16:337–357, 2006.

[56] L. Györfi and G. Ottucsák. Sequential prediction of unbounded sationary time series. *IEEE Transactions on Information Theory*, 53:1866–1872, 2007.

[57] L. Györfi and D. Shäfer. Nonparametric prediction. In *Advances in Learning Theory: Methods, Models and Applications*. IOS Press, NATO Science Series, 2003.

[58] L. Györfi, F. Udina, and H. Walk. Experiments on universal portfolio selection using data from real markets, 2008. Technical report.

[59] L. Györfi, F. Udina, and H. Walk. Nonparametric nearest neighbor based empirical portfolio selection strategies. *Statistics and Decisions*, 26:145–157, 2008.

[60] L. Györfi, F. Udina, and H. Walk. Nonparametric nearest neighbor based empirical portfolio selection strategies. *Statistics and Decisions.*, 26:145–157, 2008.

[61] P. Hall, L. Peng, and Q. Yao. Prediction and nonparametric estimation for time series with heavy tails. *Journal of Time Series Analysis*, 23:313–331, 2002.

[62] M. Inaba, N. Katoh, and H. Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract). In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, 1994.

[63] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer-Verlag, 1999.

[64] R. Koenker. *Quantile Regression*. Cambridge University Press, 2005.

[65] R. Koenker and G. W. Basset. Regression quantiles. *Econometrica*, 46:33–50, 1978.

[66] R. Koenker and K. F. Hallock. Quantile regression. *Journal of Economic Perspectives*, 15:143–156, 2001.

[67] T. Kohonen. Analysis of a simple self-organizing process. *Biological Cybernetics*, 44:135–140, 1982.

[68] H. J. Kushner and D. S. Clark. *Stochastic approximation for constrained and unconstrained systems*. Springer-Verlag, 1978.

[69] H. Li. Working with cloud queue and blob storage. In *Introduction to Windows Azure*. Apress, 2009.

[70] T. Linder. On the training distortion of vector quantizers. *IEEE Transactions on Information Theory*, 46:1617–1623, 2000.

[71] T. Linder. Learning-theoretic methods in vector quantization. In *Lecture Notes for the Advanced School on the Principles of Nonparametric Learning*, 2001.

[72] T. Linder, G. Lugosi, and K. Zeger. Rates of convergence in the source coding theorem, in empirical quantizer design, and in universal lossy source coding. *IEEE Transactions on Information Theory*, 40:1728–1740, 1994.

[73] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 2003.

[74] G. Louppe and P. Geurts. A zealous parallel gradient descent algorithm. In *NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds*, 2010.

[75] J. B. MacQueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[76] S. G. Madrikakis, S. C. Wheelwright, and R. J. Hyndman. *Forecasting, methods and applications*. Wiley, 1998.

[77] R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured perceptron. In *HLT 2010: Human Language Technologies*, 2010.

[78] G. W. Milligan and P. D. Isaac. The validation of four ultrametric clustering algorithms. *Pattern Recognition*, 12:41–50, 1980.

[79] B. Mirkin. *Clustering for data mining: a data recovery approach*. Chapman & Hall/CRC, 2005.

[80] N. Murata. A statistical study of on-line learning. In *Online Learning and Neural Networks*, 1998.

[81] Andrew B. Nobel. On optimal sequential prediction for general processes. *IEEE Transactions on Information Theory*, 49:83–98, 2003.

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

[83] A. D. Peterson, A. P. Ghosh, and R. Maitra. A systematic evaluation of different methods for initializing the k-means clustering algorithm. Technical report, 2010.

[84] D. Pollard. Stong consistency of k-means clustering. *The annals of statistics*, 9:135–140, 1981.

[85] D. Pollard. A central limit theorem for k-means clustering. *The annals of probability*, 28:199–205, 1982.

[86] D. Pollard. Quantization and the method of k-means. *IEEE Transactions on Information Theory*, 28:199–205, 1982.

[87] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22:400–407, 1951.

[88] Fabrice Rossi, Brieuc Conan-Guez, and Aïcha El Golli. Clustering functional data with the som algorithm. In *Proceedings of ESANN 2004*, 2004.

[89] T. F. Roy. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[90] M. J. Sabin and R. M. Gray. Global convergence and empirical consistency of the generalized lloyd algorithm. *IEEE Transactions on Information Theory*, 32:148–155, 1986.

[91] J. O. Street, R. J. Carroll, and D. Ruppert. A note on computing robust regression estimates via iteratively reweighted least squares. *The American Statistician*, 42:152–154, 1988.

[92] Ichiro Takeuchi, Quoc V. Le, Timothy D. Sears, and Alewander J. Smola. Nonparametric quantile estimation. *Journal of Machine Learning Reasearch*, 7:1231–1264, 2007.

[93] J. Tsitsiklis. *Problems in Decentralized Decision Making and Computation*. PhD thesis, Department of EECS, MIT, Cambridge, USA, 1984.

[94] J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31:803–812, 1986.

[95] C. Vecchiola, S. Pandey, and R. Buyya. High-performance cloud computing: a view of scientific applications. In *ISPAN*, 2009.

[96] M. Zinkevich, A. Smola, and J. Langford. Slow learners are fast. In *Advances in Neural Information Processing Systems 22*. Curran Associates, 2009.

[97] M. Zinkevich, A. Smola, and J. Langford. Slow learners are fast. In *Advances in Neural Information Processing Systems 22*. Curran Associates, 2009.

[98] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23*, 2010.

Résumé

Les thèmes abordés dans ce manuscrit de thèse sont inspirés de problématiques de recherche rencontrées par la société Lokad, qui sont résumées dans le premier chapitre. Le Chapitre 2 est consacré à l'étude d'une méthode non paramétrique de prévision des quantiles d'une série temporelle. Nous démontrons, en particulier, que la technique proposée converge sous des hypothèses minimales. La suite des travaux porte sur des algorithmes de clustering répartis et asynchrones (DALVQ). Ainsi, le Chapitre 3 propose tout d'abord une description mathématique de ces modèles précédent, et se poursuit ensuite par leur étude théorique. Notamment, nous démontrons l'existence d'un consensus asymptotique et la convergence presque sûre de la procédure vers des points critiques de la distortion. Le chapitre suivant propose des réflexions ainsi que des expériences sur les schémas de parallélisation à mettre en place pour une réalisation effective des algorithmes de type DALVQ. Enfin, le cinquième et dernier chapitre présente une implémentation de ces méthodes sur la plate-forme de *Cloud Computing* `Microsoft Windows Azure`. Nous y étudions, entre autres thèmes, l'accélération de la convergence de l'algorithme par l'augmentation de ressources parallèles. Nous le comparons ensuite avec la méthode dite de Lloyd, elle aussi répartie et déployée sur `Windows Azure`.

Mots-clés : séries temporelles, prévision quantile, perte pinball, agrégation d'experts, k-means, quantification vectorielle, calcul réparti, asynchronisme, consensus réparti, Cloud Computing, Windows Azure.

Abstract

The subjects addressed in this thesis manuscript are inspired from research problems encountered by the company Lokad, which are summarized in the first chapter. Chapter 2 deals with a nonparametric method for forecasting the quantiles of a real-valued time series. In particular, we establish a consistency result for this technique under minimal assumptions. The remainder of the dissertation is devoted to the analysis of distributed asynchronous clustering algorithms (DALVQ). Chapter 3 first proposes a mathematical description of the models and then offers a theoretical analysis, where the existence of an asymptotical consensus and the almost sure convergence towards critical points of the distortion are proved. In the next chapter, we propose a thorough discussion as well as some experiments on parallelization schemes to be implemented for a practical deployment of DALVQ algorithms. Finally, Chapter 5 contains an effective implementation of DALVQ on the *Cloud Computing* platform `Microsoft Windows Azure`. We study, among other topics, the speed ups brought by the algorithm with more parallel computing ressources, and we compare this algorithm with the so-called Lloyd's method, which is also distributed and deployed on `Windows Azure`.

Keywords: time series, quantile forecasting, pinball loss, experts aggregation, k-means, vector quantization, distributed computing, asynchronous, distributed consensus, Cloud Computing, Windows Azure.