T OWARDS F ÉDÉRÉ L GAGNER CHEZ S CALE: S YSTEM ré ESIGN

Keith Bonawitz [1] Hubert Eichner [1] Wolfgang Grieskamp [1] Dzmitry Huba [1] Alex Ingerman [1] Vladimir Ivanov [1]
Chloé Kiddon [1] Jakub Konečný [1] Stefano Mazzocchi [1] H. Brendan McMahan [1] Timon Van Overveldt [1]
David Petrou [1] Daniel Ramage [1] Jason Roselander [1]

UNE BSTRACT

Federated Learning est une approche d'apprentissage automatique distribué qui permet la formation de modèles sur un vaste corpus de données décentralisées. Nous avons construit un système de production évolutif pour Federated Learning dans le domaine des appareils mobiles, basé sur TensorFlow. Dans cet article, nous décrivons la conception de haut niveau qui en résulte, esquissons certains des défis et leurs solutions, et abordons les problèmes ouverts et les orientations futures.

1 je NTRODUCTION

Apprentissage fédéré (FL) (McMahan et coll. , 2017) est une approche d'apprentissage automatique distribué qui permet de s'entraîner sur un vaste corpus de données décentralisées résidant sur des appareils tels que les téléphones mobiles. FL est un exemple de l'approche plus générale consistant à «apporter le code aux données au lieu des données au code» et aborde les problèmes fondamentaux de confidentialité, de propriété et de localisation des données. La description générale de FL a été donnée par McMahan et Ramage

(2017), et sa théorie a été explorée dans Konečný et coll.
(2016a); McMahan et coll. (2017 ; 2018).

Une décision de conception de base pour une infrastructure Federated Learning est de savoir s'il faut se concentrer sur des algorithmes d'apprentissage asynchrones ou synchrones. Bien que de nombreux travaux réussis sur l'apprentissage en profondeur aient utilisé la formation asynchrone, par exemple, Dean et coll.
(2012), il y a eu récemment une tendance constante vers la formation synchrone par lots volumineux, même dans le centre de données (Goyal et coll. , 2017 ; Smith et coll. , 2018). L'algorithme de moyennage fédéré de McMahan et coll. (2017) adopte une approche similaire. En outre, plusieurs approches pour améliorer les garanties de confidentialité pour FL, y compris la confidentialité différentielle (McManan et coll. , 2018) et l'agrégation sécurisée (Bonawitz et coll. ,

2017), nécessitent essentiellement une certaine notion de synchronisation sur un ensemble fixe de périphériques, de sorte que le côté serveur de l'algorithme d'apprentissage ne consomme qu'un simple agrégat des mises à jour de nombreux utilisateurs. Pour toutes ces raisons, nous avons choisi de nous concentrer sur la prise en charge des rondes synchrones, tout en atténuant les éventuels surcoûts de synchronisation via plusieurs techniques que nous décrivons par la suite. Notre système est donc prêt à exécuter des algorithmes de style SGD en gros lots ainsi que Federated Averaging, l'algorithme principal que nous exécutons en production; le pseudo-code est donné en annexe B par souci d'exhaustivité.

Dans cet article, nous rapportons une conception de système pour de tels algorithmes dans le domaine des téléphones mobiles (Android). Ce travail n'en est qu'à ses débuts, et nous n'avons pas tous les problèmes résolus, et nous ne sommes pas non plus en mesure de donner une discussion complète de tous les composants requis. Nous essayons plutôt d'esquisser les principaux composants du système, de décrire les défis et d'identifier les problèmes en suspens, dans l'espoir que cela sera utile pour susciter d'autres recherches sur les systèmes.

Notre système permet de former un réseau neuronal profond, en utilisant TensorFlow (Abadi et coll. , 2016), sur les données stockées sur le téléphone qui ne quitteront jamais l'appareil. Les pondérations sont combinées dans le cloud avec Federated Averaging, ce qui crée un modèle global qui est renvoyé aux téléphones pour inférence. Une implémentation de Secure Aggregation (Bonawitz et coll. ,

2017) garantit qu'au niveau mondial, les mises à jour individuelles des téléphones ne sont pas inspectables. Le système a été appliqué dans des applications à grande échelle, par exemple dans le domaine d'un clavier de téléphone.

Notre travail aborde de nombreuses questions pratiques: la disponibilité des appareils qui est en corrélation avec la distribution locale des données de manière complexe (par exemple, dépendance au fuseau horaire); connectivité des appareils non fiable et exécution interrompue; orchestration de l'exécution par étapes de verrouillage sur les appareils avec une disponibilité variable; et un stockage limité des périphériques et des ressources de calcul. Ces problèmes sont traités au niveau du protocole de communication, du périphérique et du serveur. Nous avons atteint un état de maturité suffisant pour déployer le système en production et résoudre les problèmes d'apprentissage appliqués sur des dizaines de millions d'appareils du monde réel; nous prévoyons des utilisations où le nombre d'appareils atteint des milliards.

[1] Google Inc., Mountain View, Californie, États-Unis. Correspondant à: Wolfgang Grieskamp < wgg@google.com >, Vladimir Ivanov < vlivan@google.com >, Brendan McMahan <mcma- han@google.com >.

Actes de la 2nd Conférence SysML, Palo Alto, Californie, États-Unis,
2019.

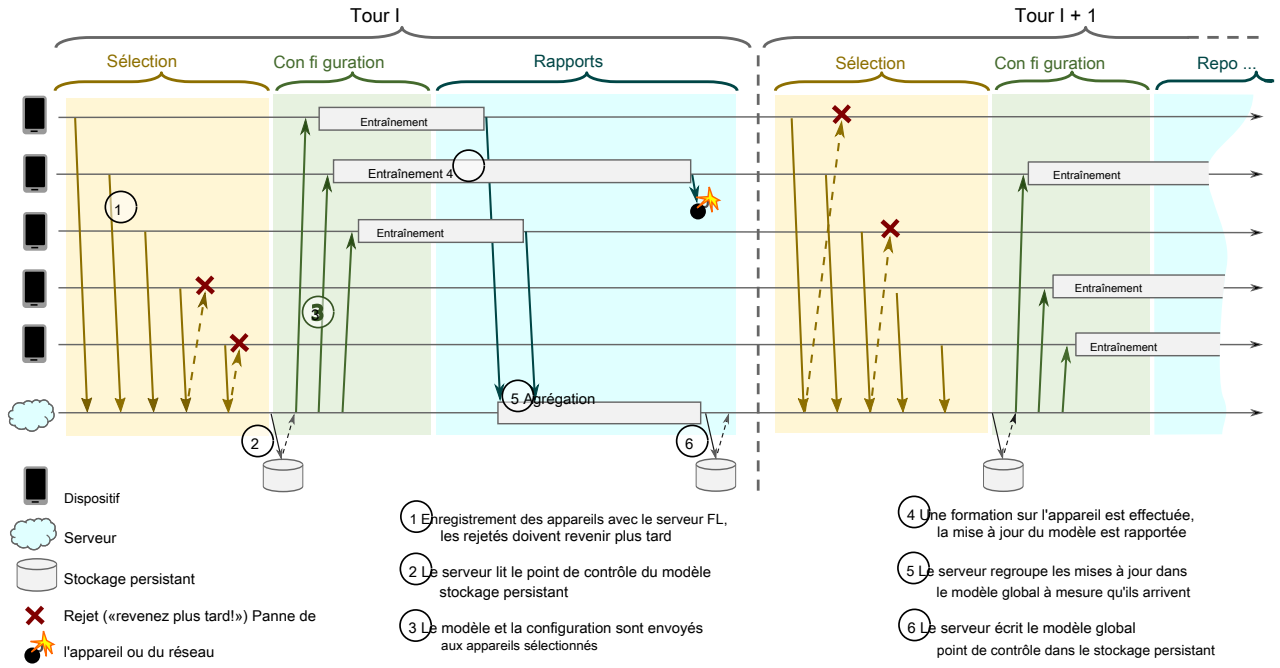

Figure 1: Protocole d'apprentissage fédéré

2 P ROTOCOL

Pour comprendre l'architecture du système, il est préférable de partir du protocole réseau.

2.1 Notions de base

Les participants au protocole sont *dispositifs (* actuellement téléphones Android) et le *Serveur FL,* qui est un service distribué basé sur le cloud. Les appareils annoncent au serveur qu'ils sont prêts à exécuter un *Tâche FL* pour un donné *Population FL.*

Une population FL est spéci fi ée par un nom globalement unique qui identi fi e le problème d'apprentissage, ou l'application, sur lequel on travaille. Une tâche FL est un calcul spéci fi que pour une population FL, tel qu'un entraînement à effectuer avec des hyperparamètres donnés, ou une évaluation de modèles entraînés sur des données de dispositif local.

À partir des dizaines de milliers de périphériques potentiels annonçant la disponibilité au serveur pendant une certaine fenêtre de temps, le serveur sélectionne un sous-ensemble de quelques centaines généralement qui sont invités à travailler sur une tâche FL spéci fi que (nous discutons de la raison de ce sous-ensemble dans Sec. 2.2). Nous appelons ce rendez-vous entre les appareils et le serveur un *rond.* Les appareils restent connectés au serveur pendant toute la durée du tour.

Le serveur indique aux périphériques sélectionnés quel calcul exécuter avec un *Plan FL,* une structure de données qui comprend un graphique TensorFlow et des instructions pour l'exécuter. Une fois qu'un tour est établi, le serveur envoie ensuite à chaque participant les paramètres du modèle global actuel et tout autre nécessaire état comme un *Point de contrôle FL (* essentiellement l'état sérialisé d'une session TensorFlow). Chaque participant effectue ensuite un calcul local basé sur l'état global et son jeu de données local, et renvoie une mise à jour sous la forme d'un point de contrôle FL au serveur. Le serveur intègre ces mises à jour dans son état global et le processus se répète.

2.2 Phases

Le protocole de communication permet aux appareils de faire progresser le modèle global singleton d'une population FL entre les cycles où chaque cycle comprend les trois phases illustrées à la Fig. 1 . Par souci de simplicité, la description ci-dessous n'inclut pas l'agrégation sécurisée, qui est décrite dans la Sec. 6 . Notez que même en l'absence d'agrégation sécurisée, tout le trafic réseau est crypté sur le câble.

Sélection Périodiquement, les appareils qui satisfont aux critères d'éligibilité (par exemple, en charge et connectés à un réseau illimité; voir Sec. 3) s'enregistrer sur le serveur en ouvrant un flux bidirectionnel. Le flux est utilisé pour suivre la vivacité et orchestrer la communication en plusieurs étapes. Le serveur sélectionne un sous-ensemble d'appareils connectés en fonction de certains objectifs comme le nombre optimal d'appareils participants (généralement quelques centaines d'appareils participent à chaque tour). Si un appareil n'est pas sélectionné pour la participation, le serveur répond avec des instructions pour se reconnecter ultérieurement. [1]

[1] Dans la mise en œuvre actuelle, la sélection se fait par simple échantillonnage de réservoir, mais le protocole se prête à des

Con fi guration Le serveur est configuré sur la base de l'ag-
mécanisme de grégation sélectionné (par exemple, simple ou sécurisé Ag-
gregation) pour les périphériques sélectionnés. Le serveur envoie le FL
plan et un point de contrôle FL avec le modèle global à chacun des
les appareils.

Rapports Le serveur attend que les appareils participants rapportent les mises à jour. Au fur et à mesure que les mises à jour sont reçues, le serveur les agrège à l'aide de la moyenne fédérée et indique aux dispositifs de rapport quand se reconnecter (voir aussi Sec. 2,3). Si suffisamment d'appareils rapportent à temps, le tour sera terminé avec succès et le serveur mettra à jour son modèle global, sinon le tour est abandonné.

Comme le montre la Fig. 1 , les périphériques en retard qui ne rapportent pas dans le temps ou ne réagissent pas à la configuration par le serveur seront simplement ignorés. Le protocole a une certaine tolérance pour ces abandons, qui est configurable par tâche FL.

Les phases de sélection et de rapport sont spéci fi ées par un ensemble de paramètres qui engendrent des fenêtres temporelles flexibles. Par exemple, pour la phase de sélection, le serveur prend en compte le nombre d'objectifs des participants de l'appareil, un délai d'expiration et un pourcentage minimal du nombre d'objectifs requis pour exécuter le tour. La phase de sélection dure jusqu'à ce que le nombre d'objectifs soit atteint ou qu'un délai d'attente se produise; dans ce dernier cas, le tour sera lancé ou abandonné selon que le nombre minimal de buts a été atteint.

2.3 Direction de l'allure

La direction de la cadence est un mécanisme de contrôle du débit régulant le schéma des connexions des appareils. Il permet au serveur FL à la fois de se dimensionner pour gérer de petites populations FL et de passer à de très grandes populations FL.

Le pilotage de l'allure est basé sur le mécanisme simple du serveur suggérant à l'appareil la fenêtre de temps optimale pour se reconnecter. L'appareil tente de respecter cela, modulo son éligibilité.

Dans le cas de petites populations FL, le pilotage de la cadence est utilisé pour garantir qu'un nombre suffisant de périphériques se connectent simultanément au serveur. Ceci est important à la fois pour le taux de progression des tâches et pour les propriétés de sécurité du protocole Secure Aggregation. Le serveur utilise un algorithme probabiliste sans état ne nécessitant aucune communication périphérique / serveur supplémentaire pour suggérer des heures de reconnexion aux périphériques rejetés de sorte que les vérifications ultérieures soient susceptibles d'arriver simultanément.

Pour les grandes populations de FL, le pilotage de l'allure est utilisé pour randomiser les heures d'enregistrement des appareils, en évitant le problème du «troupeau tonitruant» et en demandant aux appareils de se connecter aussi souvent que nécessaire pour exécuter toutes les tâches FL planifiées, mais pas plus.

---

méthodes citées qui traitent le biais de sélection.

La direction d'allure prend également en compte l'oscillation diurne dans le nombre d'appareils actifs, et est capable d'ajuster le fenêtre de temps en conséquence, évitant une activité excessive pendant heures de pointe et sans nuire aux performances FL pendant les autres périodes de la journée.

3 J EVICE

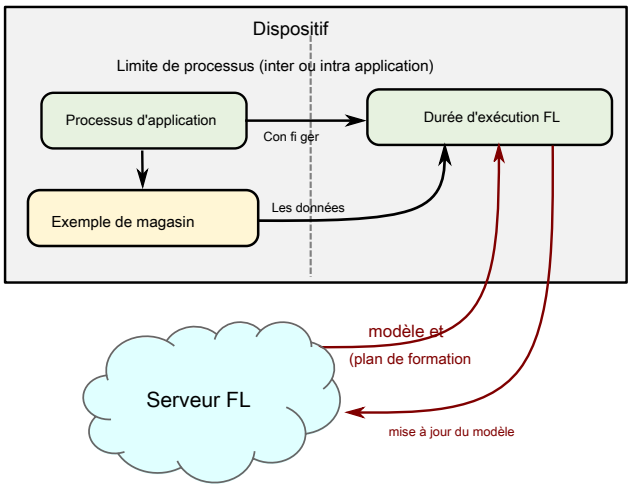

Figure 2: Architecture de l'appareil

Cette section décrit l'architecture logicielle exécutée sur un appareil participant à FL. Ceci décrit notre implémentation Android mais notez que les choix architecturaux faits ici ne sont pas particulièrement spécifiques à la plate-forme.

La première responsabilité de l'appareil dans l'apprentissage sur l'appareil est de maintenir un référentiel de données collectées localement pour l'apprentissage et l'évaluation des modèles. Les applications sont responsables de la mise à disposition de leurs données au runtime FL en tant que *exemple de magasin* en mettant en œuvre une API que nous fournissons. Un exemple de magasin d'une application peut être, par exemple, une base de données SQLite enregistrant des suggestions d'action présentées à l'utilisateur et indiquant si ces suggestions ont été acceptées ou non. Nous recommandons aux applications de limiter l'empreinte de stockage totale de leurs exemples de magasins et de supprimer automatiquement les anciennes données après un délai d'expiration prédéfini, le cas échéant. Nous fournissons des utilitaires pour faciliter ces tâches. Les données stockées sur les appareils peuvent être vulnérables aux menaces telles que les logiciels malveillants ou le démontage physique du téléphone.Nous recommandons donc aux applications de suivre les meilleures pratiques en matière de sécurité des données sur l'appareil, notamment en veillant à ce que les données soient cryptées au repos de la manière recommandée par la plate-forme.

Le moteur d'exécution FL, lorsqu'il est fourni une tâche par le serveur FL, accède à un exemple de magasin approprié pour calculer les mises à jour du modèle ou évaluer la qualité du modèle sur les données bloquées. Figure. 2
montre la relation entre l'exemple de magasin et le runtime FL. Le flux de contrôle comprend les étapes suivantes:

Configuration programmatique Une application se con fi gure

l'environnement d'exécution FL en fournissant un nom de population FL et enregistrer ses exemples de magasins. Cela programme un Travail d'exécution FL à l'aide du JobScheduler d'Android. Peut-être le exigence la plus importante pour la formation au machine learning (ML) sur les appareils des utilisateurs finaux est d'éviter tout impact sur l'expérience utilisateur, l'utilisation des données ou la durée de vie de la batterie. L'exécution FL demande que le planificateur de travaux n'appelle que le travail lorsque le téléphone est inactif, en charge et connecté à un réseau illimité tel que le WiFi. Une fois démarré, le runtime FL s'arrêtera, libérant les ressources allouées, si ces conditions ne sont plus remplies.

Invocation de tâche Lors de l'appel par le planificateur de travaux dans un processus distinct, le moteur d'exécution FL contacte le serveur FL pour annoncer qu'il est prêt à exécuter des tâches pour la population FL donnée. Le serveur décide si des tâches FL sont disponibles pour la population et renverra un plan FL ou une heure suggérée pour s'enregistrer ultérieurement.

Exécution des tâches Si l'appareil a été sélectionné, le moteur d'exécution FL reçoit le plan FL, interroge le magasin d'exemples de l'application pour les données demandées par le plan et calcule les mises à jour et les métriques de modèle déterminées par le plan.

Rapports Après l'exécution du plan FL, le runtime FL signale les mises à jour et les métriques calculées au serveur et nettoie toutes les ressources temporaires.

Comme déjà mentionné, les plans FL ne sont pas spécialisés dans la formation, mais peuvent également encoder *évaluation* Tâches - calcul de métriques de qualité à partir de données non utilisées pour la formation, analogue à l'étape de validation dans la formation du centre de données.

Notre conception permet au runtime FL de s'exécuter dans l'application qui l'a configuré ou dans un service centralisé hébergé dans une autre application. Le choix entre ces deux nécessite des changements de code minimes. La communication entre l'application, le runtime FL et le magasin d'exemples d'application comme illustré à la Fig. 2 est implémenté via le mécanisme AIDL IPC d'Android, qui fonctionne à la fois dans une seule application et entre les applications.

Locations multiples Notre implémentation offre un *multi-locataire* architecture, prenant en charge la formation de plusieurs populations FL dans la même application (ou service). Cela permet la coordination entre plusieurs activités de formation, évitant que l'appareil ne soit surchargé par de nombreuses sessions de formation simultanées.

Attestation Nous voulons que les appareils participent à FL de manière anonyme, ce qui exclut la possibilité de les authentifier via une identité d'utilisateur. Sans vérifier l'identité de l'utilisateur, nous devons nous protéger contre les attaques pour in fl uencer le résultat FL de périphériques non authentiques. Nous le faisons en utilisant le mécanisme d'attestation à distance d'Android (Documentation Android), ce qui permet de garantir que seuls les appareils et applica- participent à FL, et nous donne une certaine protection contre empoisonnement des données (Bagdasaryan et coll. , 2018) via compromis dispositifs. D'autres formes de manipulation de modèles - comme la con- fermes de tentes utilisant des téléphones sans compromis pour diriger un modèle - sont également des domaines de préoccupation potentiels auxquels nous ne traitons pas la portée de cet article.

4 S ERVER

La conception du serveur FL est motivée par la nécessité de fonctionner sur de nombreux ordres de grandeur de taille de population et d'autres dimensions. Le serveur doit fonctionner avec des populations FL dont la taille va de dizaines d'appareils (pendant le développement) à des centaines de millions, et être capable de traiter des tournées avec un nombre de participants allant de dizaines d'appareils à des dizaines de milliers. En outre, la taille des mises à jour collectées et communiquées au cours de chaque cycle peut varier de kilo-octets à des dizaines de mégaoctets. Enfin, la quantité de trafic entrant ou sortant d'une région géographique donnée peut varier considérablement au cours d'une journée en fonction du moment où les appareils sont inactifs et en charge. Cette section détaille la conception de l'infrastructure du serveur FL compte tenu de ces exigences.

4.1 Modèle d'acteur

Le serveur FL est conçu autour du *Modèle de programmation d'acteur (* Hewitt et coll. , 1973). Les acteurs sont des primitives universelles de calcul simultané qui utilisent la transmission de messages comme seul mécanisme de communication.

Chaque acteur gère un flux de messages / événements strictement séquentiellement, conduisant à un modèle de programmation simple. L'exécution de plusieurs instances d'acteurs du même type permet une mise à l'échelle naturelle vers un grand nombre de processeurs / machines. En réponse à un message, un acteur peut prendre des décisions locales, envoyer des messages à d'autres acteurs ou créer plus d'acteurs de manière dynamique. Selon les exigences de fonction et d'évolutivité, les instances d'acteurs peuvent être colocalisées sur le même processus / machine ou réparties entre des centres de données dans plusieurs régions géographiques, en utilisant des mécanismes de configuration explicites ou automatiques. La création et le placement d'instances éphémères d'acteurs à granularité fine juste pour la durée d'une tâche FL donnée permettent une gestion dynamique des ressources et des décisions d'équilibrage de charge.

4.2 Architecture

Les principaux acteurs du système sont représentés sur la Fig. 3 .

Coordinateurs sont les acteurs de haut niveau qui permettent la synchronisation globale et l'avancement des cycles de manière synchronisée. Il existe plusieurs coordinateurs, et chacun est responsable d'une population de périphériques FL. Un coordinateur enregistre son adresse et la population FL qu'il gère dans un service de verrouillage partagé, il y a donc toujours un seul propriétaire pour chaque population FL qui est joignable par d'autres acteurs du système,

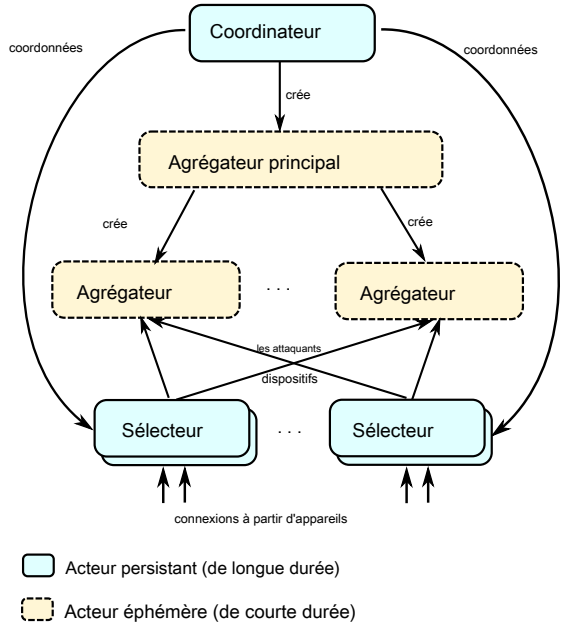

Figure 3: Acteurs dans l'architecture du serveur FL

notamment les sélecteurs. Le coordinateur reçoit des informations sur le nombre d'appareils connectés à chaque sélecteur et leur indique le nombre d'appareils à accepter pour la participation, en fonction des tâches FL planifiées. Les coordinateurs engendrent des agrégateurs maîtres pour gérer les tours de chaque tâche FL.

Sélecteurs sont responsables de l'acceptation et du transfert des connexions de l'appareil. Ils reçoivent périodiquement des informations du Coordinateur sur le nombre d'appareils nécessaires pour chaque population FL, qu'ils utilisent pour prendre des décisions locales quant à l'acceptation ou non de chaque appareil. Une fois que l'agrégateur maître et l'ensemble d'agrégateurs ont été générés, le coordinateur demande aux sélecteurs de transférer un sous-ensemble de ses périphériques connectés aux agrégateurs, permettant au coordinateur d'allouer efficacement des périphériques aux tâches FL quel que soit le nombre de périphériques disponibles. L'approche permet également aux sélecteurs d'être répartis globalement (à proximité des appareils) et de limiter la communication avec le coordinateur distant.

Agrégateurs maîtres gérer les tours de chaque tâche FL. Afin de s'adapter au nombre d'appareils et à la taille de la mise à jour, ils prennent des décisions dynamiques pour générer un ou plusieurs
Agrégateurs auquel le travail est délégué.

Aucune information pour un tour n'est écrite dans le stockage persistant tant qu'elle n'est pas entièrement agrégée par le Master Aggregator. Spécifiquement, tous les acteurs gardent leur état en mémoire et sont éphémères. Les acteurs éphémères améliorent l'évolutivité en supprimant la latence normalement encourue par le stockage distribué. L'agrégation en mémoire supprime également la possibilité d'attaques au sein du centre de données qui ciblent les journaux persistants des mises à jour par appareil, car de tels journaux n'existent pas.

4.3 Pipeline

Pendant les phases de sélection, de con fi guration et de rapport d'un tour (Sec. 2) sont séquentielles, la phase de sélection ne dépend d'aucune entrée d'un tour précédent. Cela permet une optimisation de la latence en exécutant la phase de sélection du cycle suivant du protocole en parallèle avec les phases de configuration / rapport d'un cycle précédent. Notre architecture système permet un tel pipelining sans ajouter de complexité supplémentaire, car le parallélisme est obtenu simplement grâce à des acteurs de Selector exécutant le processus de sélection en continu.

4.4 Modes de défaillance

Dans tous les cas d'échec, le système continuera à progresser, soit en terminant le tour en cours, soit en recommençant à partir des résultats du tour précédemment engagé. Dans de nombreux cas, la perte d'un acteur n'empêchera pas la ronde de réussir. Par exemple, si un agrégateur ou un sélecteur tombe en panne, seuls les périphériques connectés à cet acteur seront perdus. Si l'agrégateur maître échoue, le cycle en cours de la tâche FL qu'il gère échouera, mais sera ensuite redémarré par le coordinateur. Enfin, si le Coordinateur meurt, la couche Selector le détecte et le réapparaît. Étant donné que les coordinateurs sont enregistrés dans un service de verrouillage partagé, cela se produira exactement une fois.

5 A NALYTIQUE

Il existe de nombreux facteurs et solutions de sécurité dans l'interaction entre les périphériques et les serveurs. De plus, une grande partie de l'activité de la plate-forme se produit sur des appareils auxquels nous ne contrôlons ni n'avons accès.

Pour cette raison, nous nous appuyons sur l'analyse pour comprendre ce qui se passe réellement sur le terrain et surveiller les statistiques de santé des appareils. Du côté de l'appareil, nous effectuons des opérations gourmandes en calculs et devons éviter de gaspiller la batterie ou la bande passante du téléphone, ou de dégrader les performances du téléphone. Pour garantir cela, nous enregistrons plusieurs paramètres d'activité et d'intégrité dans le cloud. Par exemple: l'état de l'appareil dans lequel la formation a été activée, la fréquence et la durée de son exécution, la quantité de mémoire utilisée, les erreurs détectées, le modèle de téléphone / la version d'exécution du système d'exploitation / FL a été utilisé, etc. Ces entrées de journal ne contiennent aucune information d'identification personnelle (PII). Ils sont agrégés et présentés dans des tableaux de bord pour être analysés, puis introduits dans des moniteurs de séries chronologiques automatiques qui déclenchent des alertes en cas d'écarts importants.

Nous enregistrons également un événement pour chaque état dans un cycle de formation, et utilisons ces journaux pour générer des visualisations ASCII de la séquence des transitions d'état se produisant sur tous les appareils (voir le tableau 1 dans l'annexe). Nous répertorions ces visualisations de séquences dans nos tableaux de bord, ce qui nous permet de distinguer rapidement les différents types de problèmes.

Par exemple, la séquence «enregistrement, plan téléchargé, commencé la formation, terminé la formation, démarrage de l'importation, erreur "est visualisé comme "- v [] + * », Tandis que la séquence plus courte« check-dans, plan téléchargé, formation commencée, erreur "est" - v [* ». Le premier indique qu'un modèle s'est entraîné avec succès mais que le téléchargement des résultats a échoué (un problème de réseau), alors que le second indique qu'un cycle d'entraînement a échoué juste après le chargement du modèle (un problème de modèle).

Côté serveur, nous collectons de la même manière des informations telles que de nombreux appareils ont été acceptés et rejetés par formation tour, le calendrier des différentes phases de la ronde, à travers-mettre en termes de données téléchargées et téléchargées, d'erreurs et bientôt.

Depuis le déploiement de la plateforme, nous nous sommes appuyés sur l'ana-couche lytique à plusieurs reprises pour découvrir les problèmes et vérifier qu'ils ont été résolus. Certains des incidents que nous avons découverts étaient lié à la santé de l'appareil, par exemple la découverte de cette formation se passait alors que cela n'aurait pas dû, alors que d'autres fonctionnel, par exemple découvrir que les taux d'abandon des participants à la formation étaient beaucoup plus élevés que prévu.

La formation fédérée n'a pas d'impact sur l'expérience utilisateur, donc appareil et serveur *fonctionnel* les échecs n'ont pas impact négatif immédiat. Mais ne fonctionne pas correctement pourrait avoir des conséquences secondaires menant à une dégradation de l'utilité dation de l'appareil. L'utilité de l'appareil pour l'utilisateur est la mission critiques, et les dégradations sont dif fi ciles à localiser et faciles à diagnostiquer à tort. Utiliser des analyses précises pour empêcher les formation à un impact négatif sur l'utilité de l'appareil à l'utilisateur représente une part substantielle de notre ingénierie et les coûts d'atténuation des risques.

6 S ECURE UNE GRÉGATION

Bonawitz et coll. (2017) introduit *Agrégation sécurisée,* un protocole de calcul multi-parties sécurisé qui utilise le cryptage pour rendre les mises à jour de périphériques individuels non inspectables par un serveur, ne révélant la somme qu'après réception d'un nombre suffisant de mises à jour. Nous pouvons déployer Secure Aggregation comme une amélioration de la confidentialité du service FL qui protège contre les menaces supplémentaires dans le centre de données en garantissant que les mises à jour des appareils individuels restent cryptées même en mémoire. Officiellement, Secure Aggregation protège des attaquants «honnêtes mais curieux» qui peuvent avoir accès à la mémoire des instances d'Aggregator. Surtout, les seuls agrégats nécessaires pour l'évaluation du modèle, SGD ou Federated

Les moyennes sont des sommes (par exemple, $\bar{w}_t$ et $\bar{n}_t$ en annexe 1). [2]

Secure Aggregation est un protocole interactif à quatre tours activé pendant la phase de rapport d'un FL donné rond. À chaque tour de protocole, le serveur rassemble les messages de tous les appareils de la ronde FL, puis utilise l'ensemble d'appareils messages pour calculer une réponse indépendante à laquelle retourner chaque appareil. Le protocole est conçu pour être robuste à un sig-aucune fraction de périphériques abandonnant avant le protocole est complet. Les deux premiers tours constituent une phase de préparation, dans lequel des secrets partagés sont établis et pendant lesquels les appareils qui abandonnent ne verront pas leurs mises à jour incluses dans l'agrégation finale. Le troisième tour constitue un Com-phase mit, au cours de laquelle les appareils téléchargent cryptographiquement les mises à jour du modèle masqué et le serveur accumule une somme des mises à jour masquées. Tous les appareils qui terminent ce tour auront leur mise à jour de modèle incluse dans la dernière version du protocole mise à jour agrégée, sinon l'agrégation entière échouera. Le dernier cycle du protocole constitue une finalisation phase, au cours de laquelle les appareils révèlent suf fi samment secrets pour permettre au serveur de démasquer le modèle agrégé mise à jour. Tous les appareils validés ne sont pas nécessaires pour terminer ce tour; du moment qu'un nombre suffisant d'appareils a commencé à survivre au protocole pendant la phase de finalisation, tout le protocole réussit.

Plusieurs coûts de Secure Aggregation augmentent de manière quadratique avec le nombre d'utilisateurs, notamment le calcul coût pour le serveur. En pratique, cela limite le maximum taille d'une agrégation sécurisée à des centaines d'utilisateurs. Donc comme ne pas limiter le nombre d'utilisateurs pouvant participer à à chaque cycle de calcul fédéré, nous exécutons une instance de Agrégation sécurisée sur chaque acteur de l'agrégateur (voir Fig. 3) pour agréger les entrées des dispositifs de cet agrégateur en une somme intermédiaire; Les tâches FL définissent un paramètre k afin que toutes les mises à jour soient regroupées en toute sécurité sur des groupes de taille au moins k. Le maître agrégateur agrège ensuite les résultats des agrégateurs intermédiaires en un agrégat final pour le cycle, sans agrégation sécurisée.

7 T OOLS ET W ORKFLOW

Par rapport aux flux de travail standard des ingénieurs modèles sur les données collectées de manière centralisée, la formation sur appareil pose de multiples défis nouveaux. Premièrement, les exemples de formation individuels ne sont pas directement inspectables, ce qui nécessite un outillage pour travailler avec des données proxy dans les tests et la simulation (Sec. 7,1). Deuxièmement, les modèles ne peuvent pas être exécutés de manière interactive mais doivent être compilés dans des plans FL à déployer via le serveur FL (Sec. 7,2). Enfin, comme les plans FL s'exécutent sur des appareils réels, la consommation de ressources du modèle et la compatibilité d'exécution doivent être vérifiées automatiquement par l'infrastructure (Sec. 7,3).

La principale surface de développement des ingénieurs modèles travaillant

[2] Il est important de noter que le but de notre système est de fournir les outils nécessaires pour créer des applications préservant la confidentialité. La confidentialité est renforcée par la nature éphémère et ciblée des mises à jour FL, et peut être encore augmentée avec l'agrégation sécurisée et / ou la confidentialité différentielle - par exemple, les techniques de McMahan et coll. (2018) sont actuellement mis en œuvre. Cependant, alors que la plate-forme est conçue pour prendre en charge une variété de technologies améliorant la confidentialité, indiquant les garanties de confidentialité spéci fi ques dépendent des détails de l'application et des détails de la manière dont ces technologies sont utilisées; une telle discussion dépasse le cadre des travaux actuels.

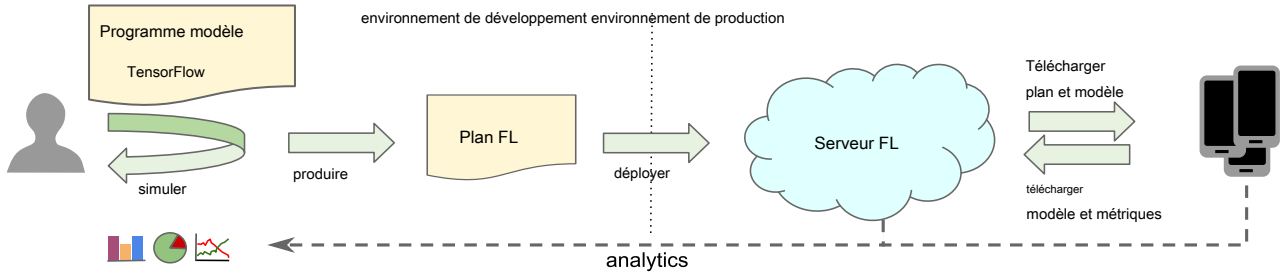

Figure 4: Flux de travail de l'ingénieur modèle

avec le système FL est un ensemble d'interfaces et d'outils Python pour dé fi nissez, testez et déployez des tâches FL basées sur TensorFlow sur Flotte d'appareils mobiles via le serveur FL. Le flux de travail d'un l'ingénieur modèle pour FL est représenté sur la Fig. 4 et décrit ci-dessous.

7.1 Modélisation et simulation

Les ingénieurs modèles commencent par dé fi nir les tâches FL qu'ils aimeraient exécuter sur une population FL donnée en Python. Notre bibliothèque permet aux ingénieurs modèles de déclarer des tâches d'apprentissage fédéré et d'évaluation à l'aide des fonctions Tensor-Flow fournies par l'ingénieur. Le rôle de ces fonctions est de mapper les tenseurs d'entrée sur les métriques de sortie telles que la perte ou la précision. Pendant le développement, les ingénieurs modèles peuvent utiliser des échantillons de données de test ou d'autres données proxy comme entrées. Une fois déployées, les entrées seront fournies à partir du magasin d'exemples sur l'appareil via le runtime FL.

Le rôle de l'infrastructure de modélisation est de permettre aux ingénieurs modèles de se concentrer sur leur modèle, en utilisant nos bibliothèques pour créer et tester les tâches FL correspondantes. Les tâches FL sont validées par rapport aux données de test et aux attentes fournies par l'ingénieur, de nature similaire aux tests unitaires. Les tests de tâche FL sont finalement nécessaires pour déployer un modèle comme décrit ci-dessous dans Sec. 7,3 .

La configuration des tâches est également écrite en Python et inclut des paramètres d'exécution tels que le nombre optimal de périphériques dans une ronde ainsi que des hyperparamètres de modèle comme le taux d'apprentissage. Les tâches FL peuvent être dé fi nies en groupes: par exemple, pour évaluer une recherche de grille sur les taux d'apprentissage. Lorsque plusieurs tâches FL sont déployées dans une population FL, le service FL choisit parmi elles en utilisant une stratégie dynamique qui permet d'alterner entre l'apprentissage et l'évaluation d'un seul modèle ou des comparaisons A / B entre les modèles.

L'exploration initiale des hyperparamètres est parfois effectuée en simulation à l'aide de données proxy. Les données proxy sont de forme similaire aux données sur l'appareil mais tirées d'une distribution différente

- par exemple, le texte de Wikipédia peut être considéré comme des données proxy pour le texte saisi sur un clavier mobile. Nos outils de modélisation permettent le déploiement de tâches FL sur un serveur FL simulé et une fl otte de jobs cloud émulant des appareils sur un grand ensemble de données proxy. La simulation exécute le même code que nous exécutons sur l'appareil et communique avec le serveur en utilisant simulé Populations FL. La simulation peut évoluer vers un grand nombre de périphériques et est parfois utilisé pour pré-entraîner des modèles sur un proxy les données avant qu'elles ne soient affinées par FL dans le champ.

7.2 Génération de plan

Chaque tâche FL est associée à un plan FL. Les plans sont automatiquement générés à partir de la combinaison du modèle et de la configuration fournie par l'ingénieur modèle. En règle générale, dans la formation d'un centre de données, les informations codées dans le plan FL seraient représentées par un programme Python qui orchestre un graphique TensorFlow. Cependant, nous n'exécutons pas Python directement sur le serveur ou les appareils. Le but du plan FL est de décrire l'orchestration souhaitée indépendamment de Python.

Un plan FL se compose de deux parties: une pour le périphérique et une pour le serveur. La partie appareil du plan FL contient, entre autres choses: le graphe TensorFlow lui-même, les critères de sélection des données d'apprentissage dans le magasin d'exemple, des instructions sur la façon de regrouper les données et le nombre d'époques à exécuter sur l'appareil, des étiquettes pour les graphique qui représente certains calculs comme le chargement et l'enregistrement de poids, etc. La partie serveur contient la logique d'agrégation, qui est codée de la même manière. Nos bibliothèques séparent automatiquement la partie du calcul d'un modèle fourni qui s'exécute sur l'appareil de la partie qui s'exécute sur le serveur (l'agrégation).

7.3 Gestion des versions, tests et déploiement

Les ingénieurs modèles travaillant dans le système fédéré sont capables de travailler de manière productive et sûre, en lançant ou en mettant fin à plusieurs expériences par jour. Mais comme chaque tâche FL peut potentiellement accaparer la RAM ou être incompatible avec les versions de TensorFlow exécutées sur la flotte, les ingénieurs s'appuient sur l'infrastructure de contrôle de version, de test et de déploiement du système FL pour des contrôles de sécurité automatisés.

Une tâche FL qui a été traduite en plan FL n'est pas acceptée par le serveur pour le déploiement à moins que certaines conditions ne soient remplies. Premièrement, il doit avoir été construit à partir de code vérifiable et révisé par des pairs. Deuxièmement, il doit avoir des prédicats de test groupés pour chaque tâche FL qui réussit en simulation. Troisièmement, les ressources consommées pendant les tests doivent se situer dans une fourchette sûre de ressources attendues pour la population cible. Et

enfin, les tests de tâches FL doivent passer sur toutes les versions du TensorFlow runtime que la tâche FL prétend prendre en charge, comme véri fi é en testant le plan de la tâche FL dans un émulateur Android.

Gestion des versions est un défi spécifique pour les machines sur appareil apprentissage. Contrairement à la formation au centre de données, où les dix Le runtime et les graphiques de sorFlow peuvent généralement être reconstruits comme nécessaire, les appareils peuvent exécuter une version de TensorFlow runtime qui a plusieurs mois de plus que ce qui est requis par le plan FL généré par les modélisateurs aujourd'hui. Par exemple, l'ancien runtime peut manquer un opérateur TensorFlow particulier, ou la signature d'un opérateur peut avoir changé d'une manière incompatible. L'infrastructure FL traite ce problème en générant *versionné* FL planifie chaque tâche. Chaque plan FL versionné est dérivé du plan FL par défaut (non révisé) en transformant son graphe de calcul pour atteindre la compatibilité avec une version déployée de TensorFlow. Les plans versionnés et non versionnés doivent passer les mêmes tests de version et sont donc traités comme sémantiquement équivalents. Nous rencontrons environ un changement incompatible qui peut être corrigé avec une transformation de graphe tous les trois mois, et un nombre légèrement plus petit qui ne peut pas être corrigé sans des solutions de contournement complexes.

7.4 Métriques

Dès qu'une tâche FL a été acceptée pour le déploiement, les périphériques qui s'enregistrent peuvent recevoir le plan approprié (versionné). Dès qu'une ronde FL se termine, les paramètres et les métriques du modèle agrégé de cette ronde sont écrits dans l'emplacement de stockage du serveur choisi par l'ingénieur modèle.

Les métriques du modèle matérialisé sont annotées avec des données supplémentaires, y compris des métadonnées telles que le nom de la tâche FL source, le numéro de ronde FL dans la tâche FL et d'autres données opérationnelles de base. Les métriques elles-mêmes sont des résumés des rapports de l'appareil au cours du cycle via des statistiques de commande approximatives et des moments comme la moyenne. Le système FL fournit des outils d'analyse aux ingénieurs modèles pour charger ces métriques dans des packages de science des données numériques Python standard pour la visualisation et l'exploration.

8 A PPLICATIONS

L'apprentissage fédéré s'applique mieux dans les situations où les données sur l'appareil sont plus pertinentes que les données qui existent sur les serveurs (par exemple, les appareils génèrent les données en premier lieu), sont sensibles à la confidentialité, ou autrement indésirables ou impossibles à transmettre aux serveurs. . Les applications actuelles de l'apprentissage fédéré sont destinées aux tâches d'apprentissage supervisé, utilisant généralement des étiquettes déduites de l'activité de l'utilisateur (par exemple, des clics ou des mots tapés).

Classement des articles sur l'appareil Une utilisation courante de l'apprentissage automatique dans les applications mobiles consiste à sélectionner et classer des éléments à partir d'un inventaire sur l'appareil. Par exemple, les applications peuvent présenter un mécanisme de recherche pour la récupération d'informations ou dans l'application navigation, par exemple recherche de paramètres sur Google Pixel devices (ai.google , 2018). En classant ces résultats sur l'appareil, appels coûteux vers le serveur (par exemple, latence, bande passante ou dimensions de la consommation d'énergie) sont éliminées, et tout informations potentiellement privées issues de la requête de recherche et la sélection de l'utilisateur reste sur l'appareil. Chaque interaction utilisateur avec la fonction de classement peut devenir un point de données étiqueté, car il est possible d'observer l'interaction de l'utilisateur avec le élément préféré dans le contexte de la liste complète des classements.

Suggestions de contenu pour les claviers intégrés Sur l'appareil les implémentations de clavier peuvent ajouter de la valeur aux utilisateurs en suggérant un contenu pertinent - par exemple, des requêtes de recherche liées au texte d'entrée. L'apprentissage fédéré peut être utilisé pour entraîner des modèles ML pour déclencher la fonction de suggestion, ainsi que pour classer les éléments qui peuvent être suggérés dans le contexte actuel. Cette approche a été adoptée par l'équipe de clavier mobile Gboard de Google, en utilisant notre système FL (Yang et coll. , 2018).

Prédiction du mot suivant Gboard a également utilisé notre plateforme FL pour former un réseau neuronal récurrent (RNN) pour la prédiction du mot suivant (Hard et coll. , 2018). Ce modèle, qui a environ 1,4 million de paramètres, converge en 3000 tours FL après avoir traité 6e8 phrases de 1,5e6 utilisateurs sur 5 jours de formation (donc chaque tour prend environ 2 à 3 minutes). [3] Il améliore le rappel du top 1 par rapport à une ligne de base *n*- modèle gramme de 13,0% à 16,4%, et correspond aux performances d'un RNN formé par le serveur qui nécessitait 1,2e8 étapes SGD. Dans les expériences A / B en direct, le modèle FL surpasse à la fois la *n*- gram et les modèles RNN formés par le serveur.

9 O PERATIONAL P ROFILE

Dans cette section, nous fournissons un bref aperçu de certaines mesures opérationnelles clés du système FL déployé, exécutant les charges de travail de production pendant plus d'un an; annexe UNE fournit des détails supplémentaires. Ces nombres ne sont que des exemples, puisque nous n'avons pas encore appliqué le FL à un ensemble d'applications suffisamment diversifié pour fournir une caractérisation complète. En outre, toutes les données ont été collectées dans le cadre du processus d'exploitation d'un système de production, plutôt que dans des conditions contrôlées explicitement à des fins de mesure. De nombreuses mesures de performances dépendent ici de l'appareil et de la vitesse du réseau (qui peut varier selon la région); Plan FL, modèle global et tailles de mise à jour (varie selon l'application); nombre d'échantillons par tour et

[3] C'est à peu près sept × plus lent que dans la formation de centre de données comparable du même modèle. Cependant, nous ne pensons pas que ce type de comparaison soit le principal - notre objectif principal est de permettre la formation sur des données qui ne sont pas disponibles dans le centre de données. En fait, pour le modèle mentionné, différentes données proxy ont été utilisées pour la formation du centre de données. Néanmoins, un temps de convergence d'horloge murale rapide est important pour permettre aux ingénieurs modèles d'itérer rapidement, et par conséquent, nous continuons d'optimiser à la fois notre système et nos algorithmes pour réduire les temps de convergence.

complexité de calcul par échantillon.

Nous avons conçu le système FL pour qu'il évolue élastiquement avec le nombre et la taille des populations de FL, potentiellement dans le milliards. Actuellement, le système gère un FL cumulatif taille de la population d'environ 10 millions d'appareils actifs par jour, couvrant plusieurs applications différentes.

Comme indiqué précédemment, à tout moment, seul un sous-ensemble de les appareils se connectent au serveur en raison de leur éligibilité et la direction du rythme. Compte tenu de cela, dans la pratique, nous observons que jusqu'à à 10 000 appareils participent simultanément. Ça vaut la peine notant que le nombre d'appareils participants dépend à l'heure (locale) de la journée (voir Fig. 5). Les appareils sont plus probablement inactif et en charge la nuit, et donc plus susceptible de participer. Nous avons observé un 4 × différence entre faible et un nombre élevé d'appareils participants sur 24 heures période pour une population centrée sur les États-Unis.

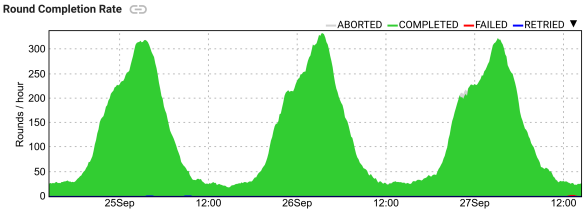

Figure 5: Taux d'achèvement du tour

Basé sur les travaux antérieurs de McMahan et coll. (2017) et des expériences que nous avons menées sur des populations de production FL, pour la plupart des modèles, recevoir des mises à jour de quelques centaines d'appareils par cycle FL est suffisant (c'est-à-dire que nous constatons une diminution des améliorations du taux de convergence de la formation sur un plus grand nombre d'appareils) . Nous observons également qu'en moyenne la part des appareils qui abandonnent en raison d'erreurs de calcul, de pannes de réseau ou de changements d'éligibilité varie entre 6% et 10%. Par conséquent, afin de compenser la perte de périphérique et de permettre aux retardataires d'être rejetés, le serveur sélectionne généralement 130% du nombre cible de périphériques pour participer initialement. Ce paramètre peut être ajusté en fonction de la distribution empirique des temps de rapport des appareils et du nombre cible de retardataires à ignorer.

10 R ELATED W ORK

Approches alternatives À notre connaissance, le système que nous avons décrit est la première implémentation de Federated Learning au niveau de la production, se concentrant principalement sur l'algorithme de Federated Averaging fonctionnant sur les téléphones mobiles. Néanmoins, il existe d'autres moyens d'apprendre à partir des données stockées sur les téléphones mobiles et d'autres contextes dans lesquels la FL en tant que concept pourrait être pertinente.

En particulier, Pihur et coll. (2018) propose un algorithme qui apprend des données des utilisateurs sans effectuer d'agrégation sur le serveur et avec des garanties de confidentialité formelles supplémentaires. Cependant, leur travail se concentre sur le mod linéaire généralisé

els, et fait valoir que leur approche est hautement évolutive en raison pour éviter la synchronisation et ne pas avoir besoin de stocker mises à jour des appareils. Notre conception de serveur décrite dans la Sec. 4 , réfute les inquiétudes concernant l'évolutivité du synchrone approche que nous utilisons, et montre en particulier que les mises à jour peuvent être traités en ligne au fur et à mesure de leur réception sans qu'il soit nécessaire de les stocker. Propositions alternatives d'algorithmes FL dans comprendre Smith et coll. (2017); Kamp et coll. (2018), qui ferait être à haut niveau compatible avec la conception du système décrit ici.

De plus, Federated Learning a déjà été proposé dans le cadre de la communication de véhicule à véhicule (Samarakoon et coll. , 2018) et applications médicales (Brisimi et coll. , 2018). Alors que le système décrit dans ce travail dans son ensemble ne s'applique pas directement à ces scénarios, de nombreux aspects il serait probablement pertinent pour l'application de production.

Nishio et Yonetani (2018) se concentre sur l'application de FL en dif- conditions environnementales férentes, à savoir où le serveur peut atteindre n'importe quel sous-ensemble de périphériques hétérogènes pour lancer une ronde, mais reçoit des mises à jour séquentiellement en raison de la limite de bande passante cellulaire. Le travail propose un algorithme de sélection conscient des ressources qui maximise le nombre de participants à un tour, qui est implémentable dans notre système.

ML distribué Des travaux importants ont été réalisés sur l'apprentissage automatique distribué, et des systèmes basés sur le cloud à grande échelle ont été décrits et sont utilisés dans la pratique. De nombreux systèmes prennent en charge plusieurs schémas de distribution, notamment *parallélisme de modèle* et *parallélisme des données,* par exemple, Dean et coll. (2012) et Low et coll. (2012). Notre système impose une approche plus structurée adaptée au domaine des appareils mobiles, qui ont une bande passante et une fiabilité bien inférieures par rapport aux nœuds de centre de données. Nous n'autorisons pas le calcul distribué arbitraire mais nous nous concentrons plutôt sur un protocole FL synchrone. Cette spécialisation de domaine nous permet, du point de vue du système, d'optimiser pour le cas d'utilisation spéci fi que.

Une approche particulièrement courante dans le datacenter est la *serveur de paramètres,* par exemple, Li et coll. (2014); Dean et coll. (2012); Abadi et coll. (2016), qui permet à un grand nombre de travailleurs de collaborer sur un modèle global partagé, le vecteur de paramètres. Dans cette ligne de travail, l'accent est mis sur une architecture de serveur ef fi cace pour traiter des vecteurs de la taille de dix [9] à dix [12]. Le serveur de paramètres fournit un état global auquel les travailleurs accèdent et mettent à jour de manière asynchrone. Notre approche ne peut intrinsèquement pas fonctionner avec un tel état global, car nous avons besoin d'un rendez-vous spéci fi que entre un ensemble de périphériques et le serveur FL pour effectuer une mise à jour synchrone avec Secure Aggregation.

MapReduce Pour les applications de centre de données, il est désormais communément admis que MapReduce (Dean et Ghemawat , 2008) n'est pas le bon cadre pour la formation ML. Pour l'espace des problèmes de FL, MapReduce est un parent proche. On peut interpréter le serveur FL comme le réducteur et les périphériques FL comme

Mappeurs. Cependant, il existe également des différences par rapport à un framework MapReduce générique. Dans notre système, les appareils FL possèdent les données sur lesquelles ils sont travail. Ce sont des acteurs totalement auto-contrôlés qui tendre et laisser les tours de calcul à volonté. À son tour, le FL le serveur recherche activement les périphériques FL disponibles et apporte seuls des sous-ensembles sélectionnés d'entre eux ensemble pour un tour de comp putation. Le serveur doit fonctionner avec le fait que de nombreux les périphériques tombent pendant le calcul, et cette disponibilité des appareils FL varie considérablement au fil du temps. Ces très spe- les exigences ci fi ques sont mieux traitées par un domaine spéci fi que framework qu'un MapReduce générique.

11 F UTURE W ORK

Biais La moyenne fédérée (McMahan et coll. , 2017) protocole suppose que tous les appareils sont également susceptibles de participer et de terminer chaque tour. En pratique, notre système introduit potentiellement un biais par le fait que les appareils ne s'entraînent que lorsqu'ils sont sur un réseau illimité et en charge. Dans certains pays, la majorité des gens ont rarement accès à un réseau illimité. De plus, nous limitons le déploiement de notre code appareil uniquement à certains téléphones, actuellement avec des versions récentes d'Android et au moins 2 Go de mémoire, une autre source de biais potentiel.

Nous abordons cette possibilité dans le système actuel comme suit: Pendant la formation FL, les modèles ne sont pas utilisés pour faire des prédictions visibles par l'utilisateur; au lieu de cela, une fois qu'un modèle est formé, il est évalué dans des expériences A / B en direct en utilisant plusieurs métriques spécifiques à l'application (tout comme avec un modèle de centre de données). Si un biais dans la participation de l'appareil ou d'autres problèmes conduisent à un modèle inférieur, il sera détecté à ce stade. Jusqu'à présent, nous n'avons pas observé ce problème dans la pratique, mais cela dépend probablement de l'application et de la population. Une quanti fi cation plus poussée de ces effets possibles sur un plus large éventail d'applications et, si nécessaire, des approches algorithmiques ou systémiques pour les atténuer, sont des orientations importantes pour les travaux futurs.

Temps de convergence Nous avons noté dans la Sec. 8 que nous observons actuellement un temps de convergence plus lent pour l'apprentissage fédéré par rapport au ML sur des données centralisées où la formation est soutenue par la puissance d'un centre de données. Les algorithmes FL actuels tels que Federated Averaging ne peuvent utiliser ef fi cacement que des centaines de dispositifs en parallèle, mais de nombreux autres sont disponibles; FL bénéficierait grandement de nouveaux algorithmes qui peuvent utiliser un parallélisme accru.

Sur le plan opérationnel, il y a aussi plus à faire. Par exemple, les fenêtres de temps pour sélectionner les appareils pour la formation et attendre leur rapport sont actuellement con fi gurées statiquement par population FL. Il doit être ajusté dynamiquement pour réduire le taux d'abandon et augmenter la fréquence des rondes. Nous devrions idéalement utiliser le ML en ligne pour régler ceci et d'autres paramètres de la configuration du protocole, en introduisant par exemple l'heure de la journée comme contexte.

Planification des appareils Actuellement, notre multi-tenant sur appareil Le planificateur utilise une simple file d'attente de travail pour déterminer séance d'entraînement à exécuter ensuite (nous évitons d'exécuter des séances d'entraînement sions sur l'appareil en parallèle en raison de leurs ressources élevées consommation). Cette approche est aveugle aux aspects comme lesquels applications que l'utilisateur utilise fréquemment. C'est possible pour nous pour finir par nous entraîner à plusieurs reprises sur des données plus anciennes (jusqu'à la date d'expiration) pour certaines applications, tout en négligeant le train- sur des données plus récentes pour les applications que l'utilisateur utilise fréquemment. Toute optimisation ici, cependant, doit être soigneusement évaluée contre les biais qu'elle peut introduire.

Bande passante Lorsque vous travaillez avec certains types de modèles, par exemple, des réseaux récurrents pour la modélisation du langage, même de petites quantités de données brutes peuvent entraîner la communication de grandes quantités d'informations (mises à jour de poids). En particulier, cela pourrait être plus que si nous téléchargions simplement les données brutes. Bien que cela puisse être considéré comme un compromis pour une meilleure confidentialité, il y a aussi beaucoup à améliorer. Pour réduire la bande passante nécessaire, nous mettons en œuvre des techniques de compression comme celles de Konečný et coll. (2016b) et

Caldas et coll. (2018). En plus de cela, nous pouvons modifier les algorithmes d'apprentissage pour obtenir des modèles en représentation quantifiée (Jacob et coll. , 2017), ce qui aura un effet synergique avec des économies de bande passante et sera important pour un déploiement ef fi cace pour l'inférence.

Calcul fédéré Nous pensons qu'il y a plus d'applications que ML pour l'architecture générale de périphérique / serveur que nous avons décrite dans cet article. Cela ressort également du fait que cet article ne contient aucune mention explicite d'une logique de ML. Au lieu de cela, nous nous référons de façon abstraite aux «plans», aux «modèles», aux «mises à jour», etc.

Nous visons à généraliser notre système de Federated Learning à *Calcul fédéré,* qui suit les mêmes principes de base que ceux décrits dans cet article, mais ne limite pas le calcul au ML avec TensorFlow, mais en général MapRe- duce comme les charges de travail. Un domaine d'application que nous voyons est dans *Analytics fédérée,* ce qui nous permettrait de surveiller les statistiques globales des appareils sans enregistrer les données brutes des appareils dans le cloud.

UNE RECONNAISSANCE

Comme la plupart des systèmes d'échelle de production, il y a beaucoup plus de contributeurs que les auteurs de cet article. Au moins, les personnes suivantes ont directement contribué à la conception et à la mise en œuvre: Galen Andrew, Blaise Agüera y Arcas, Sean Augenstein, Dave Bacon, Françoise Beaufays, Amlan Chakraborty, Arlie Davis, Stefan Dierauf, Randy Dodgen, Emily Glanz, Shiyu Hu, Ben Kreuter, Eric Loewenthal, Antonio Marcedone, Jason Hunter, Krzysztof Ostrowski, Sarvar Patel, Peter Kairouz, Kanishka Rao, Michael Re- neer, Aaron Segal, Karn Seth, Wei Huang, Nicholas Kong, Haicheng Sun, Vivian Lee, Tim Yang, Yuanbo Zhang.

R ÉFÉRENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, GS, Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. et Zheng, X. TensorFlow: Apprentissage automatique à grande échelle sur des systèmes hétérogènes. Dans *OSDI,* volume 16, pp. 265-283, 2016.

ai.google. Sous le capot du pixel 2: comment est l'IA matériel de suralimentation, 2018. URL https: // ai. google / stories / ai-in-hardware / . Récupéré Novembre 2018.

Documentation Android. API d'attestation SafetyNet. URL https://developer.android.com/ formation / filet de sécurité / attestation .

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., et Shmatikov, V. Comment faire une porte dérobée à l'apprentissage fédéré. *préimpression arXiv arXiv: 1807.00459,* 2018.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, HB, Patel, S., Ramage, D., Segal, A. et Seth, K. Agrégation sécurisée pratique pour l'apprentissage automatique préservant la confidentialité. Dans *Actes de la conférence ACM SIGSAC 2017 sur la sécurité informatique et des communications,* pp. 1175–1191. ACM, 2017.

Brisimi, TS, Chen, R., Mela, T., Olshevsky, A., Paschalidis, IC et Shi, W. Apprentissage fédéré de modèles prédictifs à partir de dossiers de santé électroniques fédérés. *Revue internationale d'informatique médicale,* 112: 59 à 67, 2018.

Caldas, S., Konecný, J., McMahan, HB et Talwalkar, A. Élargir la portée de l'apprentissage fédéré en réduisant les besoins en ressources des clients. *préimpression arXiv 1812.07210,* 2018.

Dean, J. et Ghemawat, S. MapReduce: données simplifiées traitement sur de grands clusters. *Communications de l'ACM,* 51 (1): 107-113, 2008.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, QV, Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et Ng, AY Réseaux profonds distribués à grande échelle. Dans *Progrès des systèmes de traitement de l'information neuronale,* pp. 1223-1231, 2012.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., et He, K. Grand minibatch précis SGD: imagenet d'entraînement en 1 heure. *préimpression arXiv arXiv: 1706.02677,* 2017.

Hard, A., Rao, K., Mathews, R., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C. et Ramage, D. Apprentissage fédéré pour la prédiction de clavier mobile. *préimpression arXiv 1811.03604,* 2018.

Hewitt, C., Bishop, PB et Steiger, R. Un module universel grand formalisme ACTOR pour l'intelligence artificielle. Dans *Actes de la 3e Conférence internationale conjointe sur l'intelligence artificielle. Stanford, CA, USA, 20-23 août, 1973,* pp. 235-245, 1973.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., et Kalenichenko, D. Quantification et apprentissage des réseaux de neurones pour une inférence efficace uniquement en arithmétique entière. *préimpression arXiv arXiv: 1712.05877,* 2017.

Kamp, M., Adilova, L., Sicking, J., Hüger, F., Schlicht, P., Wirtz, T., et Wrobel, S. Apprentissage en profondeur décentralisé efficace par moyennage de modèles dynamiques. *préimpression arXiv arXiv: 1807.03210,* 2018.

Konečný, J., McMahan, HB, Ramage, D., et Richtárik, P. Optimisation fédérée: apprentissage automatique distribué pour l'intelligence sur l'appareil. *préimpression arXiv arXiv: 1610.02527,* 2016a.

Konečný, J., McMahan, HB, Yu, FX, Richtárik, P., Suresh, AT et Bacon, D. Apprentissage fédéré: stratégies pour améliorer l'ef fi cacité de la communication. *arXiv préimpression arXiv: 1610.05492,* 2016b.

Li, M., Andersen, DG, Park, JW, Smola, AJ, Ahmed, A., Josifovski, V., Long, J., Shekita, EJ et Su, B.-Y. Mise à l'échelle de l'apprentissage automatique distribué avec le serveur de paramètres. Dans *11e Symposium USENIX sur la conception et l'implémentation des systèmes d'exploitation (OSDI 14),* 583-598. Association USENIX, 2014.

Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., et Hellerstein, JM Distributed graphlab: un cadre pour l'apprentissage automatique et l'exploration de données dans le cloud. *Proc. VLDB Endow.,* 5 (8): 716–727, avril 2012.

McMahan, HB et Ramage, D. Fédéré apprentissage: Apprentissage automatique collaboratif avec- sur les données de formation centralisées, avril 2017. URL https://ai.googleblog.com/2017/04/ federated-learning-collaboratif.html . Blog Google AI.

McMahan, HB, Moore, E., Ramage, D., Hampson, S., et y Arcas, BA Apprentissage efficace en communication des réseaux profonds à partir de données décentralisées. Dans *Actes de la 20e Conférence internationale sur l'intelligence artificielle et les statistiques,* pp. 1273-1282, 2017.

McMahan, HB, Ramage, D., Talwar, K., et Zhang, L.
Apprendre des modèles linguistiques récurrents différentiellement privés. Dans *Conférence internationale sur les représentations d'apprentissage (ICLR),* 2018.

Nishio, T. et Yonetani, R. Sélection des clients pour les
apprentissage avec des ressources hétérogènes en périphérie mobile.
préimpression arXiv arXiv: 1804.08333, 2018.

Pihur, V., Korolova, A., Liu, F., Sankuratripati, S., Yung,
M., Huang, D. et Zeng, R. Apprentissage automatique différentiellement
privé «dessiner et rejeter». *préimpression arXiv arXiv: 1807.04369,* 2018.

Samarakoon, S., Bennis, M., Saad, W., et Debbah, M.
Apprentissage fédéré pour des communications v2v ultra-fiables à faible latence. *préimpression arXiv arXiv: 1805.09253,* 2018.

Smith, S., jan Kindermans, P., Ying, C., et Le, QV Don't
diminuer le taux d'apprentissage, augmenter la taille du lot. Dans *Conférence internationale sur les représentations d'apprentissage (ICLR),*
2018.

Smith, V., Chiang, C.-K., Sanjabi, M., et Talwalkar, AS
Apprentissage multitâche fédéré. Dans *Progrès des systèmes de traitement de l'information neuronale,* 4424 à 4434, 2017.

Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong,
N., Ramage, D. et Beaufays, F. Apprentissage fédéré appliqué: amélioration
des suggestions de requêtes de clavier Google.
préimpression arXiv 1812.02903, 2018.

AO PERATIONAL P ROFILE ré À

Dans cette section, nous présentons les données de pro fi l opérationnel pour l'une des populations FL qui sont actuellement actives dans le système FL déployé, augmentant ainsi la discussion de la Sec. 9 . La population FL en question provient principalement du même fuseau horaire.

Figure. 6 illustre comment la disponibilité des appareils varie au cours de la journée et son impact sur le taux d'achèvement du cycle. Etant donné que le serveur FL ne planifie une tâche FL à exécuter qu'une fois qu'un nombre souhaité de périphériques est disponible et sélectionné, le taux d'achèvement du cycle oscille en synchronisation avec la disponibilité du périphérique.

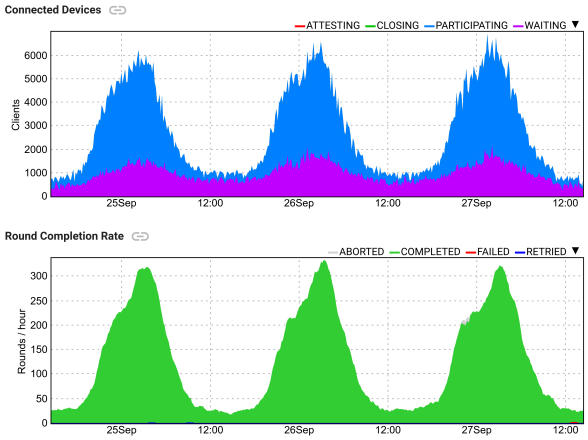

Figure 6: Un sous-ensemble des appareils connectés sur trois jours (en haut) dans les états «participant» (bleu) et «en attente» (violet). Les autres états («fermeture» et «attestation») sont trop rares pour être visibles sur ce graphique. Le taux de réussite des cycles (vert, en bas) est également indiqué, ainsi que le taux d'autres résultats («échec», «réessayer» et «abandonner») tracés sur le même graphique mais trop bas pour être visibles.

Figure. sept illustre le nombre moyen d'appareils participant à une série de tâches FL et les résultats de la participation. Notez qu'à chaque tour, le serveur FL sélectionne plus de périphériques pour la participation que vous le souhaitez pour compenser les périphériques qui abandonnent pendant l'exécution. Par conséquent, à chaque tour, il y a des périphériques qui ont été abandonnés après un nombre souhaité de périphériques terminé avec succès. Un autre aspect notable est la corrélation du taux d'abandon avec l'heure de la journée, en particulier le taux d'abandon est plus élevé pendant la journée que pendant la nuit. Cela s'explique par une probabilité plus élevée que les critères d'éligibilité de l'appareil changent en raison de l'interaction avec un appareil.

Figure. 8 montre la répartition du temps de parcours et de participation de l'appareil. Il y a deux observations notables. Tout d'abord, le temps d'exécution du cycle est à peu près égal à la majorité du temps de participation de l'appareil, ce qui s'explique par le fait que le serveur FL sélectionne plus de périphériques que nécessaire pour la participation et arrête l'exécution lorsque suffisamment de périphériques sont terminés. Deuxièmement, le temps de participation de l'appareil est plafonné.

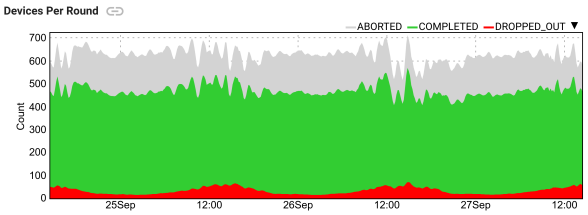

Figure 7: Nombre moyen d'appareils terminés, abandonnés et abandonnés de l'exécution du cycle

Il s'agit d'un mécanisme utilisé par le serveur FL pour gérer les périphériques retardataires; c'est-à-dire le temps d'exécution rond plafonné par le serveur.

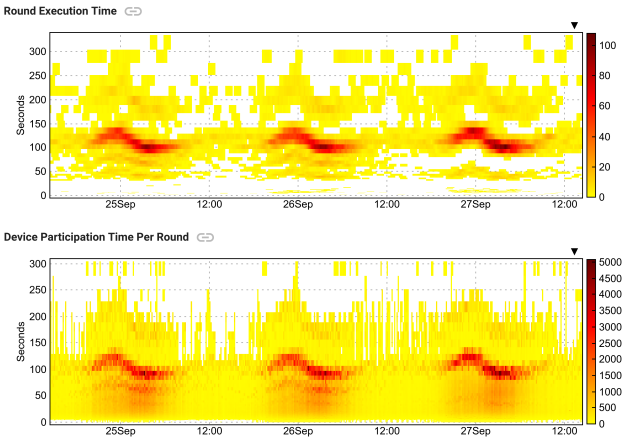

Figure 8: Exécution du cycle et temps de participation de l'appareil

Figure. 9 illustre l'asymétrie dans le trafic réseau des serveurs, en particulier que le téléchargement à partir du serveur domine le téléchargement. Il y a plusieurs aspects qui y contribuent. À savoir, chaque appareil télécharge à la fois un plan de tâches FL et un modèle global actuel (la taille du plan est comparable au modèle global) alors qu'il télécharge uniquement les mises à jour vers le modèle global; les mises à jour du modèle sont intrinsèquement plus compressibles que le modèle global.

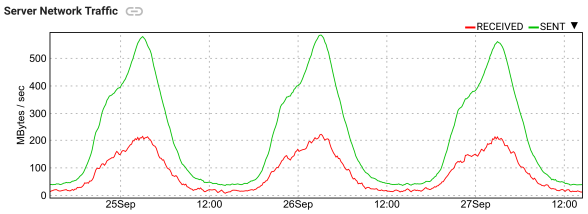

Figure 9: Traf fi c réseau du serveur

Languette. 1 affiche les visualisations de forme de session d'entraînement générées à partir des journaux d'événements d'état d'entraînement des clients. Comme indiqué, 75% des clients terminent leurs cycles de formation avec succès, 22% des clients terminent leurs cycles de formation mais voient leurs résultats rejetés par le serveur (ce sont les appareils qui font un rapport après la fenêtre de reporting déjà fermée), et 2% des clients sont interrompus avant de pouvoir

Forme de la session	Pourcentage de comptage	
- v [] + ˆ	1 116 401	75%
- v [] + #	327 478	22%
- v [!	29 771	2%

Tableau 1: Répartition des sessions de formation sur l'appareil. Légende: - = enregistrement du serveur FL, v = plan téléchargé, [= formation commencée,] = formation terminée, + = téléchargement commencé, ˆ = téléchargement terminé, # = téléchargement rejeté,! = interrompu.

pour terminer leur tour (par exemple, parce que l'appareil est sorti de l'état inactif).

BF ÉDÉRÉ UNE VERAGER

Dans cette section, nous montrons l'algorithme Federated Averaging de McMahan et coll. (2017) pour le lecteur intéressé.

Algorithme 1 FederatedAveraging mises à jour du ciblage de K clients par tour.

Le serveur exécute:

initialiser w_0

pour chaque tour $t = 1, 2, \ldots$ faire

Sélectionner 1.3 K clients éligibles pour calculer les mises à jour Attendre les mises à jour de K clients (indexés $1, \ldots, K$)

$(\Delta_k, n_k) =$ ClientUpdate (w) du client $k \in [K]$.

$\bar{w}_t = \sum_k n_k \Delta_k$ // *Somme des mises à jour pondérées*

$\bar{n}_t = \sum_k n_k$ // *Somme des poids*

$\Delta_t = \Delta_k / \bar{n}_t$ // *Mise à jour moyenne*

$w_{t+1} \leftarrow w_t + \Delta_t$

ClientUpdate (w):

$B \leftarrow$ (données locales divisées en minibatchs)

$n \leftarrow |B|$ // *Mettre à jour le poids*

$w_{init} \leftarrow w$

pour lot $b \in B$ faire

$w \leftarrow w - \eta \nabla \ell(w; b)$

$\Delta \leftarrow n \cdot (w - w_{init})$ // *Mise à jour pondérée*

// *Remarque* Δ *se prête mieux à la compression que* w

revenir (Δ, n) au serveur