# Solutions de stockage NetApp pour Apache Spark

## NetApp Solutions

NetApp
April 25, 2024

# Sommaire

# Solutions de stockage NetApp pour Apache Spark

## Tr-4570 : Solutions de stockage NetApp pour Apache Spark : architecture, cas d'utilisation et résultats des performances

Rick Huang, Karthikeyan Nagalingam, NetApp

Ce document est consacré à l'architecture Apache Spark, aux utilisations client et au portefeuille de solutions de stockage NetApp consacré à l'analytique Big Data et à l'intelligence artificielle (IA). Nous avons également présenté les résultats de plusieurs tests à l'aide des outils standard de l'IA, du machine learning (ML) et du deep learning (DL) par rapport à un système Hadoop standard qui vous permet de choisir la solution Spark adaptée. Il vous faut tout d'abord une architecture Spark, les composants appropriés et deux modes de déploiement (cluster et client).

Ce document présente également des cas d'utilisation pour résoudre les problèmes de configuration. Il présente également la gamme de solutions de stockage NetApp qui traite de l'analytique Big Data, de l'IA, DU ML et du DL avec Spark. Nous terminons ensuite avec les résultats des tests effectués à partir des cas d'utilisation propres à Spark et de la gamme de solutions NetApp Spark.

### Défis des clients

Cette section se concentre sur les défis clients liés à l'analytique Big Data et à l'IA/AM/AP dans des secteurs en croissance de données tels que le commerce, le marketing digital, la banque, la fabrication discrète, la fabrication des processus, américain et à ses services professionnels.

#### Performances imprévisibles

Les déploiements Hadoop classiques utilisent généralement du matériel ordinaire. Pour optimiser les performances, vous devez configurer le réseau, le système d'exploitation, le cluster Hadoop, les composants de l'écosystème tels que Spark et le matériel. Même si vous ajustez chaque couche, il peut être difficile d'atteindre les niveaux de performance souhaités, car Hadoop fonctionne sur du matériel générique qui n'a pas été conçu pour assurer de hautes performances dans votre environnement.

#### Pannes de nœuds et de supports

Même dans des conditions normales, le matériel de base est susceptible de subir des défaillances. Si un disque d'un nœud de données tombe en panne, le maître Hadoop considère par défaut que ce nœud est défectueux. Il copie ensuite les données spécifiques de ce nœud sur le réseau, des réplicas à un nœud en bon état. Ce processus ralentit les paquets réseau pour toutes les tâches Hadoop. Le cluster doit ensuite recopier les données et supprimer les données sur-répliquées lorsque le nœud défectueux revient à un état sain.

#### Dépendance vis-à-vis d'un fournisseur Hadoop

Les distributeurs Hadoop disposent de leur propre distribution Hadoop avec leurs propres versions, qui dépendent des clients de ces distributions. Toutefois, de nombreux clients ont besoin d'une prise en charge pour les analyses en mémoire qui n'lient pas le client à des distributions Hadoop spécifiques. Ils ont besoin de

la liberté de changer de distribution tout en bénéficiant de l'analytique.

### Manque de support pour plus d'une langue

Les clients ont souvent besoin d'un support pour plusieurs langues en plus des programmes MapReduce Java pour exécuter leurs tâches. Les options telles que SQL et les scripts offrent davantage de flexibilité pour obtenir les réponses, davantage d'options pour organiser et récupérer les données, et accélèrent le déplacement des données dans une structure analytique.

### Difficulté d'utilisation

Pendant quelques temps, certains se plaignent du fait que Hadoop est difficile à utiliser. Même si Hadoop est devenu plus simple et plus puissant à chaque nouvelle version, cette critique a persisté. Hadoop exige de comprendre les modèles de programmation Java et MapReduce, ce qui représente un véritable défi pour les administrateurs de base de données et les équipes qui disposent de compétences classiques en matière de script.

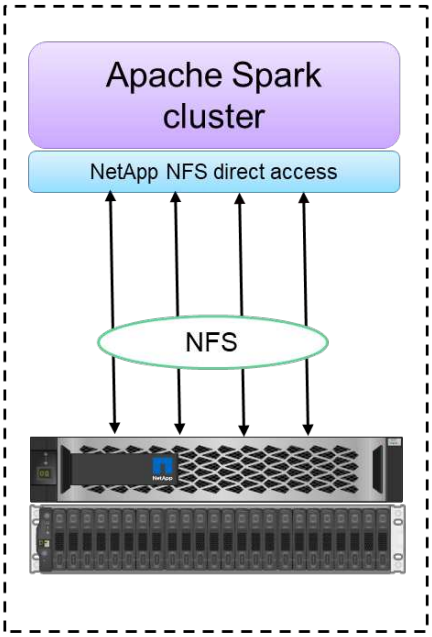
### Structures et outils complexes

Les équipes chargées de l'IA sont confrontées à plusieurs défis. Même avec des connaissances avancées en science des données, les outils et les frameworks pour différents écosystèmes et applications de déploiement ne se traduisent pas toujours par une traduction unique. Une plate-forme de data science doit s'intégrer en toute transparence aux plateformes Big Data correspondantes intégrées sur Spark et offrir la facilité de déplacement des données, des modèles réutilisables, un code prêt à l'emploi et des outils qui prennent en charge les meilleures pratiques en matière de prototypage, de validation, de gestion des versions, de partage, de réutilisation, et de déploiement rapide des modèles en production.

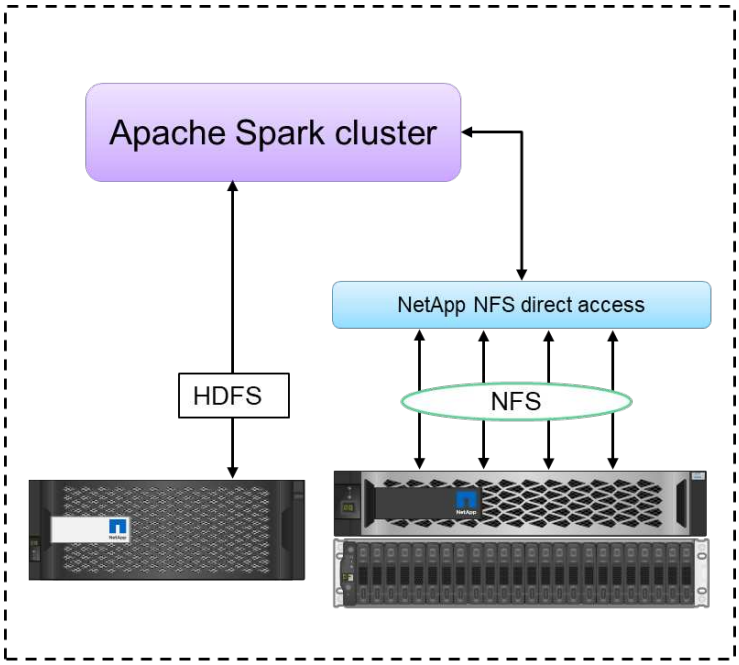
## Pourquoi choisir NetApp ?

NetApp peut améliorer votre expérience Spark de l'une des manières suivantes :

- L'accès direct NetApp NFS (voir la figure ci-dessous) permet aux clients d'exécuter des tâches d'analytique Big Data sur leurs données NFSv3 ou NFSv4 existantes ou nouvellement sollicitées sans déplacer ou copier les données. Elle empêche plusieurs copies de données et n'a plus besoin de synchroniser les données avec une source.
- Optimisation du stockage et réduction de la réplication des serveurs. Par exemple, la solution Hadoop de NetApp E-Series nécessite deux à trois réplicas des données et la solution FAS Hadoop requiert une source de données, mais aucune réplication ou copie de données. Les solutions de stockage NetApp génèrent également moins de trafic serveur à serveur.
- Amélioration des tâches Hadoop et du comportement du cluster en cas de panne de disque ou de nœud.
- De meilleures performances d'ingestion des données.




Configuration 1: NFS as primary storage




Configuration 2: HDFS and NFS in single Spark cluster

Dans le secteur financier et de la santé par exemple, ces transferts de données doivent se conformer à des obligations légales, une tâche ardue. Dans ce scénario, l'accès direct NetApp NFS analyse les données financières et de santé à partir de leur emplacement d'origine. L'autre avantage clé est que l'accès direct NetApp NFS simplifie la protection des données Hadoop grâce aux commandes Hadoop natives et à l'activation des workflows de protection des données avec la gamme complète de solutions NetApp de gestion des données.

L'accès direct NetApp NFS propose deux types d'options de déploiement pour les clusters Hadoop/Spark :

- Par défaut, les clusters Hadoop ou Spark utilisent le système HDFS (Hadoop Distributed File System) pour le stockage des données et le système de fichiers par défaut. NetApp NFS direct peut remplacer le système HDFS par défaut par un stockage NFS comme système de fichiers par défaut, permettant ainsi une analytique directe sur les données NFS.
- Dans une autre option de déploiement, l'accès direct NetApp NFS prend en charge la configuration de NFS en tant que stockage supplémentaire et HDFS dans un cluster Hadoop ou Spark unique. Dans ce cas, le client peut partager des données via les exports NFS et y accéder depuis le même cluster, ainsi que des données HDFS.

Les principaux avantages de l'accès direct NetApp NFS sont les suivants :

- L'analyse des données depuis leur emplacement actuel empêche toute tâche fastidieuse de transférer des données analytiques vers une infrastructure Hadoop telle que HDFS.
- Réduction du nombre de répliques de trois à un.
- Les utilisateurs peuvent découpler les ressources de calcul et de stockage afin de les faire évoluer de façon indépendante.
- Protection des données grâce aux fonctionnalités avancées de gestion d'ONTAP.
- Certification avec la plateforme de données Hortonworks.
- Déploiements d'analytique hybride.
- Réduction des délais de sauvegarde grâce à la fonctionnalité multithread dynamique.

Voir "Tr-4657 : solutions de données de cloud hybride NetApp - Spark et Hadoop en fonction des cas d'utilisation clients" Pour la sauvegarde de données Hadoop, la sauvegarde et la reprise d'activité depuis le cloud vers les systèmes sur site, ce qui permet le DevTest sur les données Hadoop existantes, la protection des données et la connectivité multicloud, et l'accélération des workloads d'analytique.

Les sections suivantes décrivent les fonctionnalités de stockage importantes pour les clients Spark.

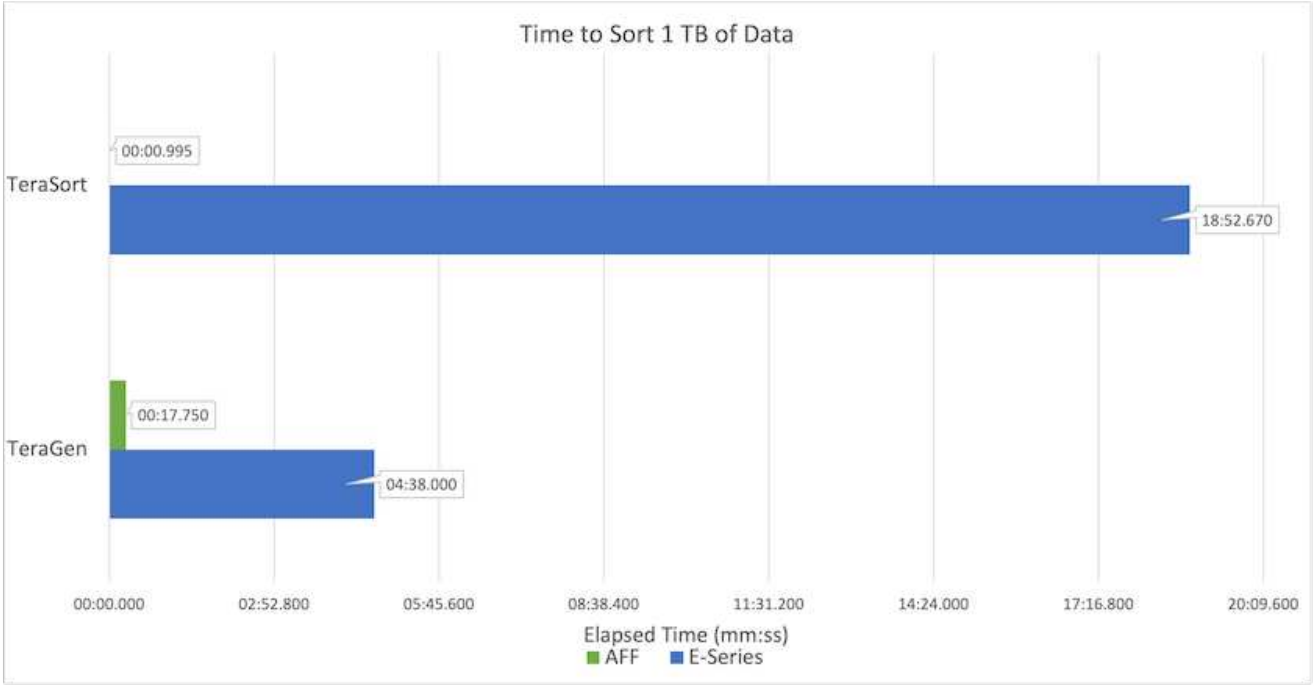
### Hiérarchisation du stockage

La hiérarchisation du stockage Hadoop permet de stocker des fichiers de différents types de stockage conformément à une règle de stockage. Les types de stockage sont notamment `hot`, `cold`, `warm`, `all_ssd`, `one_ssd`, et `lazy_persist`.

<<<<<<<< NOUS AVONS VALIDÉ la hiérarchisation du stockage Hadoop sur un contrôleur de stockage NetApp AFF et un contrôleur de stockage E-Series avec des disques SSD et SAS utilisant différentes règles de stockage. Le cluster Spark avec l'AFF-A800 dispose de quatre nœuds de traitement, tandis que le cluster avec l'E-Series en compte huit. Cette comparaison a notamment pour objectif de comparer les performances des SSD et des disques durs.

Nous avons validé la hiérarchisation du stockage Hadoop sur un contrôleur de stockage NetApp AFF et un contrôleur de stockage E-Series avec des disques SSD et SAS utilisant différentes règles de stockage. Le cluster Spark avec l'AFF-A800 dispose de quatre nœuds de traitement, tandis que le cluster avec l'E-Series en compte huit. Cela nous a principalement permis de comparer les performances des disques SSD aux disques durs. >>>>>> a51c9ddf73ca69e1120ce05edc7b0b9607b96eae

La figure suivante montre les performances des solutions NetApp pour un SSD Hadoop.




- La configuration NL-SAS de base utilisait huit nœuds de calcul et 96 disques NL-SAS. Cette configuration a généré 1 To de données en 4 minutes et 38 secondes. Voir "Tr-3969 solution NetApp E-Series pour Hadoop" pour plus d'informations sur le cluster et la configuration du stockage.
- Grâce à TeraGen, la configuration SSD a généré 1 To de données 15,6 fois plus vite que la configuration NL-SAS. De plus, la configuration SSD utilisait deux fois moins de nœuds de calcul et deux fois moins de disques (24 disques SSD au total). En fonction de la durée d'exécution des tâches, elle était presque deux fois plus rapide que la configuration NL-SAS.
- Grâce à Terasort, la configuration SSD a trié 1 To de données 1138.36 fois plus vite que la configuration NL-SAS. De plus, la configuration SSD utilisait deux fois moins de nœuds de calcul et deux fois moins de disques (24 disques SSD au total). Par conséquent, par disque, c'était environ trois fois plus rapide que la configuration NL-SAS. <<<<<<<< TÊTE
- En passant de disques rotatifs à un système 100 % Flash, le message clé est d'améliorer les performances. Le nombre de nœuds de calcul n'était pas un goulot d'étranglement. Avec le stockage 100 % Flash de NetApp, les performances d'exécution évoluent parfaitement.
- Avec NFS, les données étaient fonctionnellement équivalentes au regroupement des pools, ce qui permet de réduire le nombre de nœuds de calcul en fonction de votre charge de travail. Les utilisateurs du cluster Apache Spark n'ont pas besoin de rééquilibrer manuellement les données lors de la modification du nombre de nœuds de calcul.

- En résumé, le passage des disques rotatifs au 100 % Flash améliore les performances. Le nombre de nœuds de calcul n'était pas un goulot d'étranglement. Avec le stockage 100 % Flash de NetApp, les performances d'exécution évoluent parfaitement.

- Avec NFS, les données étaient fonctionnellement équivalentes au regroupement des pools, ce qui permet de réduire le nombre de nœuds de calcul en fonction de votre charge de travail. Les utilisateurs d'un cluster Apache Spark n'ont pas besoin de rééquilibrer manuellement les données lors de la modification du nombre de nœuds de calcul. >>>>>> a51c9ddf73ca69e1120ce05edc7b0b9607b96eae

#### Évolutivité des performances - évolutivité horizontale

Pour augmenter la puissance de calcul d'un cluster Hadoop dans une solution AFF, il est possible d'ajouter des nœuds de données avec un nombre approprié de contrôleurs de stockage. NetApp recommande de démarrer avec quatre nœuds de données par baie de contrôleur de stockage, puis d'augmenter le nombre de huit nœuds de données par contrôleur de stockage, en fonction des caractéristiques des charges de travail.

AFF et FAS sont parfaits pour l'analytique sur place. Vous pouvez ajouter des gestionnaires de nœuds et, sans interrompre l'activité, un contrôleur de stockage à la demande sans interrompre l'activité. Nous proposons des fonctionnalités riches avec AFF et FAS, notamment la prise en charge des supports NVMe, l'efficacité garantie, la réduction des données, la qualité de service, l'analytique prédictive, tiering, réplication, déploiement dans le cloud et sécurité. Pour aider les clients à satisfaire leurs besoins, NetApp propose des fonctionnalités telles que l'analytique des systèmes de fichiers, les quotas et l'équilibrage de la charge intégrée sans frais de licence supplémentaires. NetApp fournit de meilleures performances que ses concurrents en termes de nombre de tâches simultanées, de latence inférieure ou d'opérations simplifiées, et un débit par seconde supérieur à celui de ses concurrents. De plus, NetApp Cloud Volumes ONTAP s'exécute sur les trois principaux fournisseurs cloud.

#### Évolutivité des performances - évolutivité verticale

Les fonctionnalités scale-up permettent d'ajouter des disques aux systèmes AFF, FAS et E-Series lorsque vous avez besoin de capacité de stockage supplémentaire. Avec Cloud Volumes ONTAP, l'évolutivité du stockage jusqu'au niveau des po est deux facteurs : Tiering des données peu utilisées vers un stockage objet à partir d'un stockage bloc et pile des licences Cloud Volumes ONTAP sans calcul supplémentaire.

#### Protocoles multiples

Les systèmes NetApp prennent en charge la plupart des protocoles pour les déploiements Hadoop, notamment SAS, iSCSI, FCP, InfiniBand, Et NFS.

#### Solutions opérationnelles et prises en charge

Les solutions Hadoop décrites dans ce document sont prises en charge par NetApp. Ces solutions sont également certifiées avec les principaux distributeurs Hadoop. Pour plus d'informations, reportez-vous à la section "MAPR" site, le "Hortonworks" Et le Cloudera "certification" et "en tant que partenaire" distants.

# Public visé

## L'univers de l'analytique et de la science des données touche plusieurs disciplines au NIVEAU DE L'INFORMATIQUE et des activités :

- Les data Scientists doivent donc pouvoir utiliser leurs outils et leurs bibliothèques de choix.
- L'ingénieur doit savoir comment les données circulent et où elles résident.
- Un ingénieur DevOps doit disposer des outils nécessaires pour intégrer les nouvelles applications d'IA et DE ML dans son pipeline d'intégration et de livraison continues.
- Les administrateurs et architectes cloud doivent être en mesure de configurer et de gérer des ressources de cloud hybride.

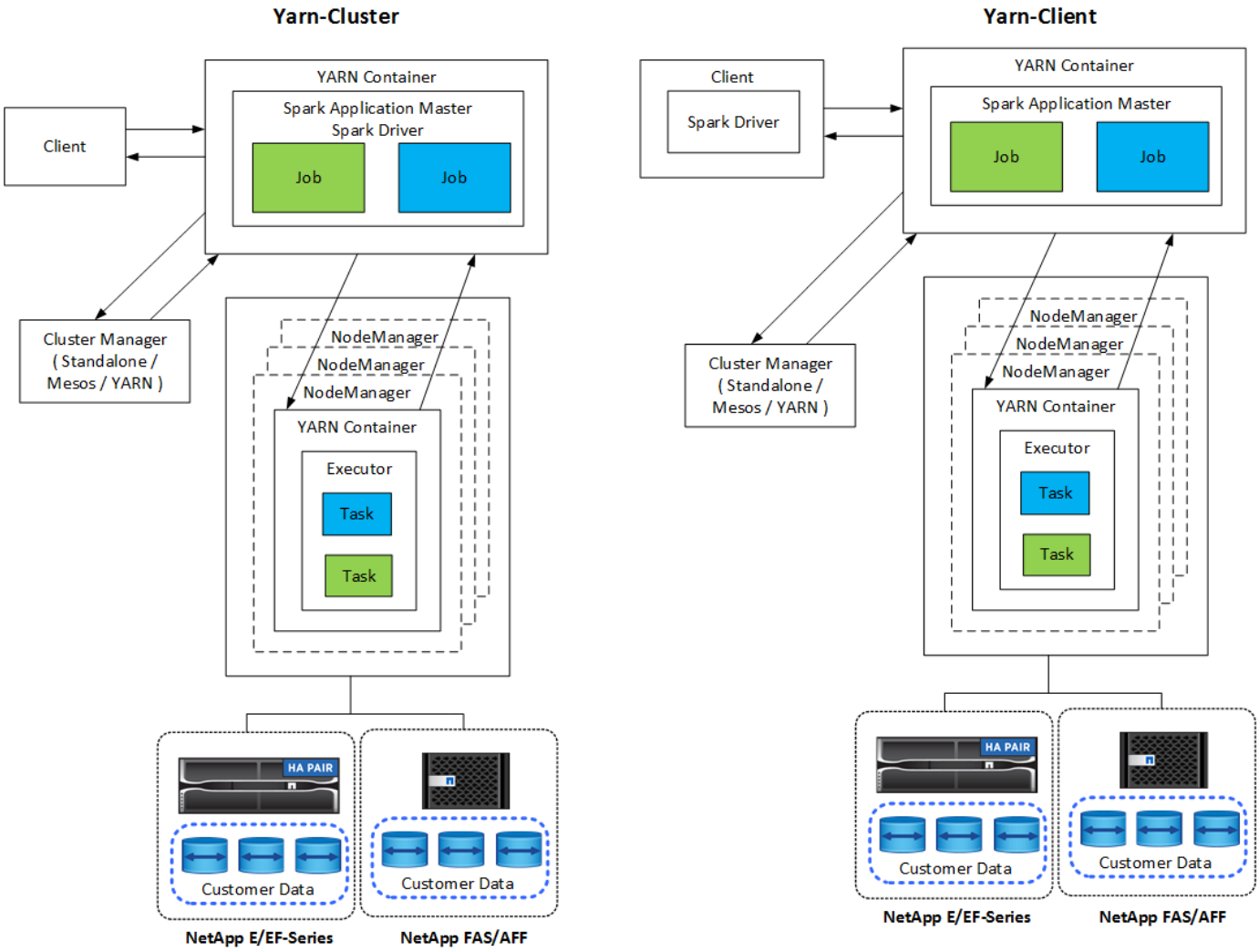
- Les utilisateurs professionnels veulent avoir accès à des applications d'analytique, d'IA, DE ML et de DL.

Dans ce rapport technique, nous explique comment NetApp AFF, E-Series, StorageGRID, NFS direct, Apache Spark Horovod et Keras aident chacun de ces rôles à apporter de la valeur aux entreprises.

# Technologie de la solution

Apache Spark est un framework de programmation célèbre qui permet de rédiger des applications Hadoop directement avec le système Hadoop Distributed File System (HDFS). Spark est prête pour la production et prend en charge le traitement du streaming de données, et elle est plus rapide que MapReduce. Spark propose une mise en cache des données en mémoire configurable pour une itération efficace, et le shell Spark est interactif pour l'apprentissage et l'exploration des données. Avec Spark, vous pouvez créer des applications en Python, Scala ou Java. Les applications SPARK consistent en un ou plusieurs travaux qui ont une ou plusieurs tâches.

Chaque application Spark est dotée d'un tournevis à bougie. En mode YARN-client, le pilote s'exécute localement sur le client. En mode YARN-Cluster, le pilote s'exécute dans le cluster du maître d'application. En mode cluster, l'application continue à fonctionner même si le client se déconnecte.



Il existe trois gestionnaires de cluster :

- **Autonome.** ce gestionnaire fait partie de Spark, ce qui facilite l'installation d'un cluster.
- **Apache Mesos.** il s'agit d'un gestionnaire de cluster général qui exécute également MapReduce et d'autres applications.
- **HADOOP YARN.** il s'agit d'un gestionnaire de ressources dans Hadoop 3.

Le jeu de données distribué résilient (RDD) est le composant principal de Spark. RDD recrée les données perdues et manquantes des données stockées dans la mémoire du cluster et stocke les données initiales provenant d'un fichier ou créées par programmation. Les RDD sont créés à partir de fichiers, de données en mémoire ou d'un autre RDD. La programmation des étincelles effectue deux opérations : la transformation et les actions. Transformation crée un nouveau RDD basé sur un RDD existant. Les actions renvoient une valeur à partir d'un RDD.

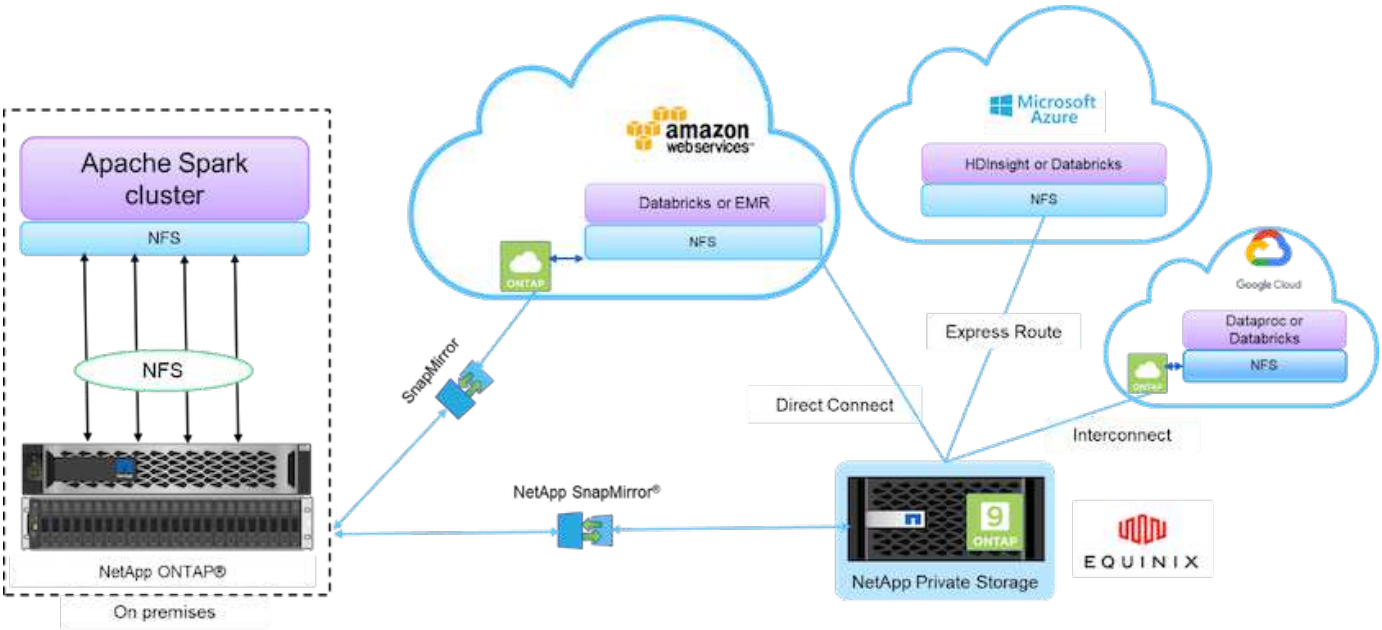
Les transformations et les actions s'appliquent également aux ensembles de données Spark et aux DataFrames. Un jeu de données est un ensemble distribué de données qui offre les avantages des RDD (fort typage, utilisation des fonctions lambda) avec les avantages du moteur d'exécution optimisé de Spark SQL. Un jeu de données peut être construit à partir d'objets JVM, puis manipulé à l'aide de transformations fonctionnelles (MAP, plantMap, filtre, etc.). Un DataFrame est un jeu de données organisé en colonnes nommées. Il est conceptuellement équivalent à une table dans une base de données relationnelle ou à une trame de données dans R/Python. DataFrames peut être construit à partir d'un large éventail de sources, telles que des fichiers de données structurés, des tables dans Hive/HBase, des bases de données externes sur site ou dans le cloud, ou des disques durs virtuels existants.

Les applications Spark incluent un ou plusieurs travaux Spark. Les travaux exécutent des tâches dans des exécuteurs et les exécuteurs exécutent des conteneurs DE FILS. Chaque exécuteur s'exécute dans un seul conteneur et des exécuteurs existent tout au long de la durée de vie d'une application. Un exécuteur est corrigé après le démarrage de l'application et LE FIL ne redimensionne pas le conteneur déjà alloué. Un exécuteur peut exécuter des tâches simultanément sur des données en mémoire.

# Présentation des solutions NetApp Spark

## Nous portefeuilles de solutions de stockage FAS/AFF, E-Series et Cloud Volumes ONTAP Nous avons validé AFF et le système de stockage E-Series avec ONTAP pour les solutions Hadoop avec Apache Spark.

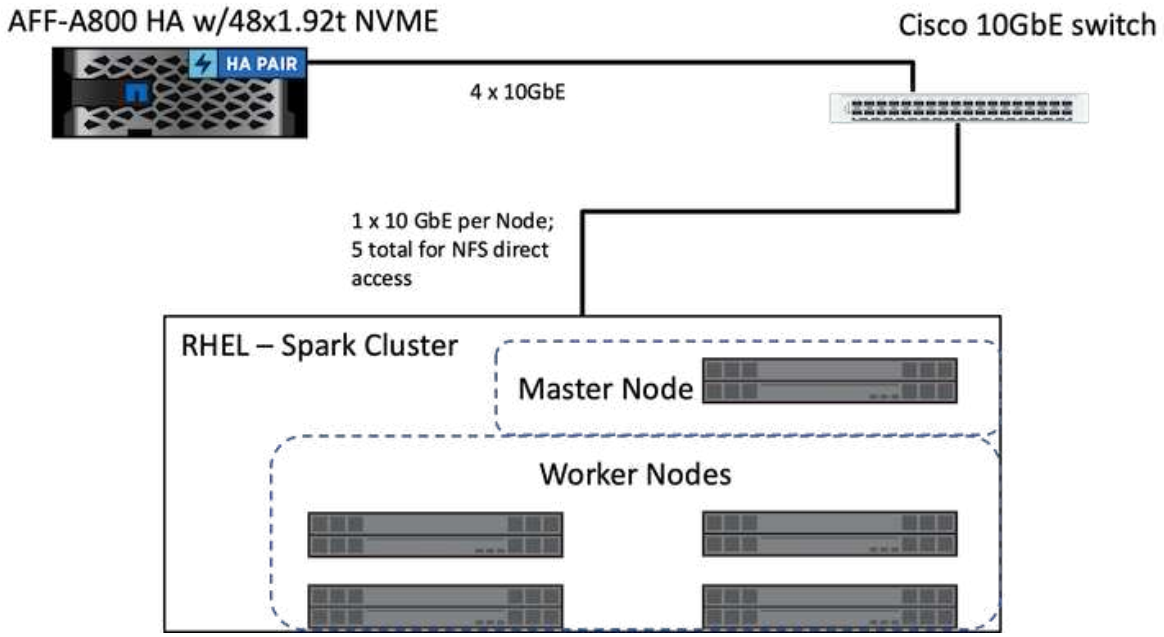
La Data Fabric optimisée par NetApp intègre des services et des applications de gestion des données (éléments de base) pour l'accès, le contrôle, la protection et la sécurité des données, comme l'illustre la figure ci-dessous.




Les éléments de base de la figure ci-dessus sont les suivants :

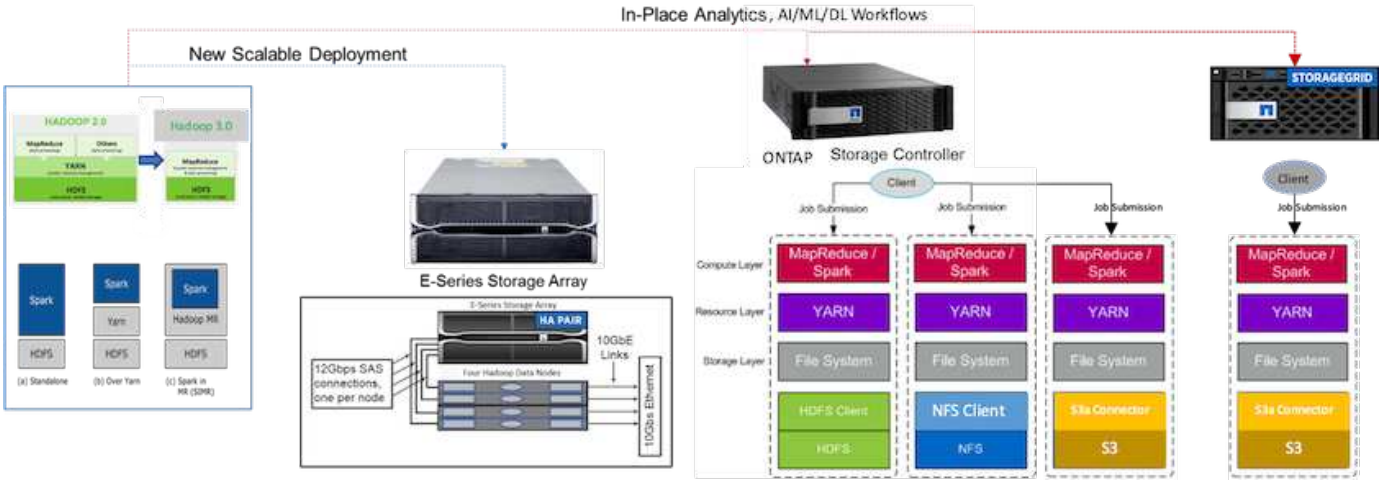
- **NetApp NFS Direct Access.** fournit les derniers clusters Hadoop et Spark avec un accès direct aux volumes NFS NetApp sans configuration logicielle ni pilote supplémentaire.
- **NetApp Cloud Volumes ONTAP et Cloud volumes Services.** stockage Software-defined basé sur ONTAP s'exécutant dans Amazon Web Services (AWS) ou Azure NetApp Files (ANF) dans les services cloud Microsoft Azure.
- **La technologie NetApp SnapMirror** fournit des fonctionnalités de protection des données entre les instances sur site et ONTAP Cloud ou NPS.
- **Fournisseurs de services cloud.** ces fournisseurs incluent AWS, Microsoft Azure, Google Cloud et IBM Cloud.
- **PaaS.** services d'analytique cloud tels qu'Amazon Elastic MapReduce (EMR) et Databricks dans AWS, ainsi que Microsoft Azure HDInsight et Azure Databricks.

La figure suivante décrit la solution Spark avec du stockage NetApp.

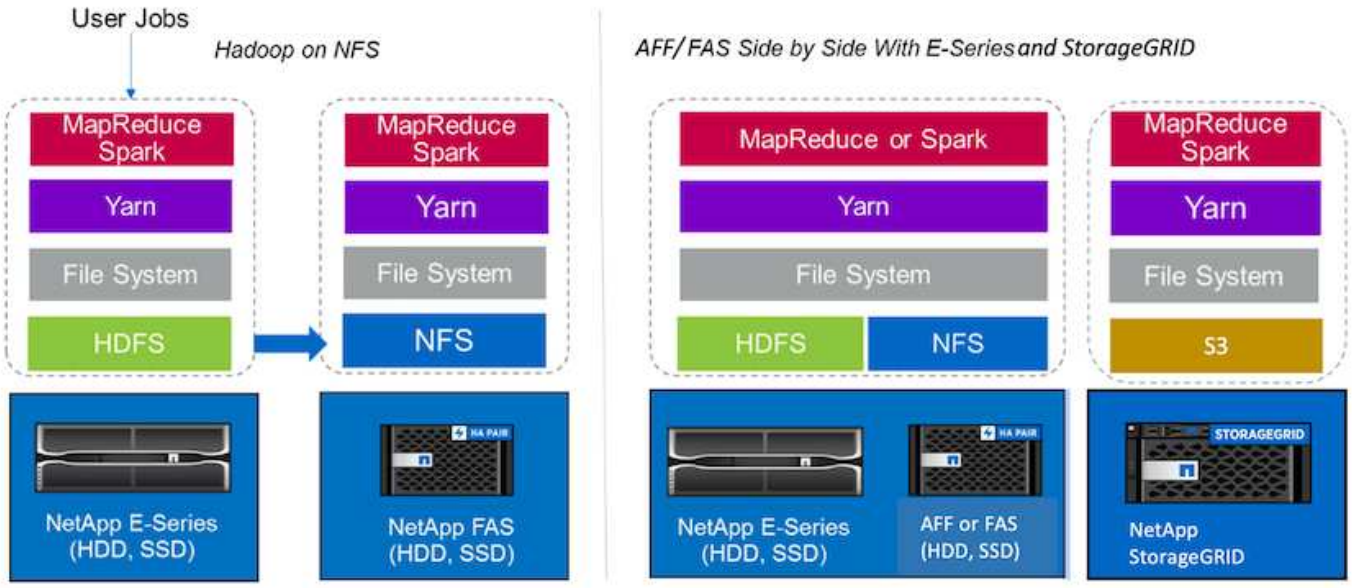



La solution ONTAP Spark utilise le protocole NetApp NFS à accès direct pour l'analytique sur place et les workflows d'IA, DE ML et de DL en utilisant un accès aux données de production existantes. Les données de production disponibles aux nœuds Hadoop sont exportées pour effectuer des tâches analytiques sur place et d'IA, DE ML et de DL. Vous pouvez accéder aux données à traiter dans les nœuds Hadoop avec l'accès direct NetApp NFS ou sans Dans Spark avec l'autonome ou `yarn` Cluster Manager vous permet de configurer un volume NFS à l'aide de `file:///<target_volume`. Nous avons validé trois cas d'utilisation avec des jeux de données différents. Les détails de ces validations sont présentés dans la section « Résultats des tests ». (xréf)

La figure suivante représente le positionnement du stockage NetApp Apache Spark/Hadoop.




Nous avons identifié les fonctionnalités uniques de la solution E-Series Spark, de la solution AFF/FAS ONTAP Spark et de la solution StorageGRID Spark, et nous avons effectué une validation et des tests détaillés. D'après nos observations, NetApp recommande la solution E-Series pour les installations nouvelles et les déploiements évolutifs. En outre, la solution AFF/FAS assure la prise en charge de l'analytique sur place, de l'IA, DU ML et du DL qui exploite les données NFS existantes, ainsi que StorageGRID pour l'IA, LE ML et le DL et l'analytique des données moderne lorsque le stockage objet est requis.




Un data Lake est un référentiel de stockage pour les datasets volumineux de forme native qui peut être utilisé pour les tâches d'analytique, d'IA, DE ML et de DL. Nous avons créé un référentiel de data Lake pour les solutions E-Series, AFF/FAS et StorageGRID SG6060 Spark. Le système E-Series fournit un accès HDFS au cluster Hadoop Spark, tandis que les données de production existantes sont accessibles via le protocole d'accès direct NFS au cluster Hadoop. Pour les datasets qui résident dans le stockage objet, NetApp StorageGRID fournit un accès sécurisé S3 et S3a.

# Récapitulatif du cas d'utilisation

Cette page décrit les différents domaines dans lesquels cette solution peut être utilisée.

## Streaming des données

Apache Spark peut traiter le streaming de données en streaming, qui est utilisé pour les processus d'extraction, de transformation et de charge (ETL), l'enrichissement des données, la détection des événements et l'analyse complexe des sessions :

- **Streaming ETL.** les données sont continuellement nettoyées et regroupées avant d'être envoyées dans les datastores. Netflix utilise Kafka et Spark pour créer une solution de surveillance des données et des recommandations de films en ligne en temps réel capable de traiter des milliards d'événements chaque jour à partir de différentes sources de données. Toutefois, le traitement par lot par ETL classique est traité différemment. Ces données sont lues d'abord, puis converties au format de la base de données avant d'être écrites dans la base de données.
- **Enrichissement des données.** la diffusion Spark enrichit les données en direct avec des données statiques pour permettre une analyse des données en temps réel. Par exemple, les annonceurs en ligne peuvent fournir des publicités ciblées personnalisées, dirigées par des informations sur le comportement des clients.
- **Détection d'événement déclencheur.** la diffusion Spark vous permet de détecter et de réagir rapidement à un comportement inhabituel qui pourrait indiquer des problèmes potentiellement graves. Par exemple, les institutions financières utilisent des déclencheurs pour détecter et arrêter les transactions de fraude, et les hôpitaux utilisent des déclencheurs pour détecter les changements sanitaires dangereux détectés dans les paramètres vitaux d'un patient.
- **Analyse de session complexe.** la diffusion en continu Spark collecte des événements tels que l'activité

de l'utilisateur après s'être connecté à un site Web ou à une application, qui sont ensuite regroupées et analysées. Par exemple, Netflix utilise cette fonctionnalité pour fournir des recommandations de films en temps réel.

Pour obtenir une configuration plus streaming, la vérification Kafka confluent et les tests de performance, consultez la section "Tr-4912 : recommandations sur les meilleures pratiques pour le stockage hiérarchisé Kafka fluide avec NetApp".

## Apprentissage machine

Le framework intégré Spark vous aide à exécuter des requêtes répétées sur des jeux de données à l'aide de la bibliothèque d'apprentissage machine (MLlib). MLlib est utilisé dans des domaines tels que la mise en grappe, la classification et la réduction de la dimensionnalité pour certaines fonctions communes de Big Data telles que l'intelligence prédictive, la segmentation des clients à des fins de marketing et l'analyse de sentiment. MLlib est utilisé dans la sécurité du réseau pour effectuer des inspections en temps réel des paquets de données pour des indications d'activité malveillante. Elle permet aux fournisseurs de sécurité de découvrir les nouvelles menaces et de garder une longueur d'avance sur les pirates informatiques tout en protégeant leurs clients en temps réel.

## L'apprentissage profond

TensorFlow est un framework de deep learning répandu dans le secteur. TensorFlow prend en charge l'entraînement distribué sur un cluster de processeur ou de processeur graphique. Cette formation distribuée permet aux utilisateurs de l'exécuter sur une grande quantité de données comportant des couches profondes.

Jusqu'à récemment, si nous souhaitions utiliser TensorFlow avec Apache Spark, nous devions effectuer tous les opérations ETL pour TensorFlow sur PySpark, puis écrire les données sur un stockage intermédiaire. Ces données seraient ensuite chargées sur le cluster TensorFlow pour le processus d'entraînement réel. Ce flux de travail exigeait que l'utilisateur conserve deux clusters différents, un pour ETL et un pour l'entraînement distribué de TensorFlow. L'exécution et la maintenance de plusieurs clusters étaient généralement fastidieuses et chronophages.

Les DataFrames et RDD dans les versions antérieures de Spark ne conviennent pas parfaitement à l'apprentissage profond, en raison de son accès aléatoire limité. La prise en charge native des frameworks de deep learning est ajoutée dans Spark 3.0 avec l'hydrogène projet. Cette approche permet de planifier des opérations non MapReduce sur le cluster Spark.

## Analyse interactive

Apache Spark est suffisamment rapide pour effectuer des requêtes exploratoires sans échantillonnage avec des langages de développement autres que Spark, y compris SQL, R et Python. Spark utilise des outils de visualisation pour traiter des données complexes et les visualiser de manière interactive. Spark avec diffusion structurée effectue des requêtes interactives sur des données en direct dans le cadre d'analyses Web qui vous permettent d'exécuter des requêtes interactives sur la session en cours d'un visiteur Web.

## Système de recommandation

Au fil des ans, les systèmes de recommandation ont apporté des changements considérables à notre vie, car les entreprises et les consommateurs ont réagi à des changements spectaculaires dans les secteurs du shopping en ligne, du divertissement en ligne et de nombreux autres secteurs. Ces systèmes constituent effectivement l'un des succès les plus évidents liés à l'IA en production. Dans de nombreux cas d'usage pratiques, les systèmes de recommandation sont combinés à l'IA ou aux chatbots de conversation avec un système backend NLP, afin d'obtenir des informations pertinentes et de produire des inférences utiles.

Aujourd'hui, de nombreux détaillants adoptent des modèles commerciaux plus récents, comme l'achat en ligne et la collecte en magasin, la collecte en ligne, le paiement à l'auto-paiement, la numérisation et l'utilisation, etc. Ces modèles occupent une place de premier plan lors de la pandémie de COVID-19 en rendant les achats plus sûrs et plus pratiques pour les consommateurs. L'IA est cruciale pour ces tendances digitales en pleine expansion, qui sont influencées par le comportement des consommateurs et inversement. Pour répondre à la demande croissante de ses clients, accroître l'expérience de leurs clients, améliorer leur efficacité opérationnelle et augmenter leur chiffre d'affaires, NetApp aide ses clients et entreprises à utiliser des algorithmes de machine learning et de deep learning pour concevoir des systèmes de recommandation plus rapides et plus précis.

Plusieurs techniques populaires sont utilisées pour formuler des recommandations, notamment le filtrage collaboratif, les systèmes basés sur le contenu, le modèle de recommandation de deep learning (DLRM) et des techniques hybrides. Auparavant, les clients utilisaient PySpark pour mettre en œuvre un filtrage collaboratif afin de créer des systèmes de recommandation. Spark MLlib implémente les moindres carrés alternatifs (ALS) pour le filtrage collaboratif, un algorithme très populaire parmi les entreprises avant la montée de DLRM.

## Le traitement du langage naturel

L'IA, rendue possible par le traitement du langage naturel (NLP), est la branche de l'IA qui aide les ordinateurs à communiquer avec les humains. Le NLP est très répandu dans tous les secteurs verticaux et nombreux cas d'utilisation, des assistants intelligents et chatbots aux applications de recherche et de texte prédictif Google. Selon un "Gartner" Prévu, d'ici 2022, 70 % des personnes interagiront quotidiennement avec des plateformes d'IA conversationnelles. Pour une conversation de haute qualité entre un humain et une machine, les réactions doivent être rapides, intelligentes et naturelles.

Les clients ont besoin d'une grande quantité de données pour traiter et former leurs modèles NLP et ASR (reconnaissance vocale automatique). Elles doivent également déplacer les données de la périphérie au cœur, et jusqu'au cloud, et elles doivent pouvoir inférence en quelques millisecondes afin d'établir une communication naturelle avec les humains. NetApp ai et Apache Spark constituent une combinaison idéale pour le calcul, le stockage, le traitement des données, l'entraînement des modèles, le réglage fin, et de déploiement des applications.

L'analyse des sentiments est un domaine d'étude au sein de NLP dans lequel des sentiments positifs, négatifs ou neutres sont extraits du texte. L'analyse de opinion comporte de nombreux cas d'utilisation, allant de la détermination de la performance des employés du centre de support lors de conversations avec les appelants à la prestation de réponses de chatbot automatisées appropriées. Il a également été utilisé pour prédire le cours des actions d'une entreprise sur la base des interactions entre les représentants de l'entreprise et le public au cours des appels trimestriels sur les bénéfices. En outre, l'analyse de sentiment peut être utilisée pour déterminer l'opinion d'un client sur les produits, les services ou l'assistance fournis par la marque.

Nous avons utilisé "Spark NLP" bibliothèque à partir de "John Snow Labs" Pour charger des pipelines pré-entraînés et des représentations d'encodeur bidirectionnelles à partir de modèles Transformers (BERT), y compris "le sentiment des informations financières" et "FinBERT", effectuer la tokenisation, la reconnaissance d'entités nommées, l'entraînement de modèle, l'ajustement et l'analyse de sentiment à grande échelle. Spark NLP est la seule bibliothèque Open Source de NLP en production qui offre des transformateurs de pointe comme BERT, ALBERT, ELECTRA, XLNet, DistillBERT, Roberta, DeBERTa, XLM- Roberta, Longex, ELMO, Encodeur de phrase universel, Google T5, MarianMT et GPT2. La bibliothèque fonctionne non seulement en Python et en R, mais aussi dans l'écosystème JVM (Java, Scala et Kotlin) à grande échelle en étendant en natif Apache Spark.

# Principales utilisations et architectures de l'IA, DU ML et du DL

Les principaux champs d'application de l'IA, DU ML et du DL peuvent être divisés en plusieurs sections :

## Les pipelines Spark NLP et l'inférence TensorFlow distribuée

La liste suivante contient les bibliothèques Open Source les plus populaires de NLP qui ont été adoptées par la communauté des sciences de la donnée sous différents niveaux de développement :

- "Boîte à outils en langage naturel (NLTK)". La boîte à outils complète pour toutes les techniques NLP. Elle est maintenue depuis le début des années 2000.
- "TextBlob". Une API Python facile à utiliser avec les outils NLP reposant sur NLTK et Pattern.
- "Stanford Core NLP". Services et paquets NLP en Java développés par le Stanford NLP Group.
- "Gensim". La modélisation des sujets pour l'homme a commencé comme un ensemble de scripts Python pour le projet de la Bibliothèque de mathématiques numériques tchèque.
- "L'espionnage". Flux de production NLP industriels de bout en bout avec Python et cython avec accélération GPU pour transformateurs.
- "Texte rapide". Une bibliothèque libre, légère et open-source de NLP pour l'apprentissage des mariages de mots et la classification des peines créée par le laboratoire de recherche sur l'IA (FAIR) de Facebook.

Spark NLP est une solution unique et unifiée pour toutes les tâches et exigences de la NLP qui permet un logiciel évolutif, haute performance et haute précision de la NLP pour les cas d'utilisation réels en production. Elle tire parti de l'apprentissage par transfert et met en œuvre les derniers algorithmes et modèles de pointe dans la recherche et dans d'autres secteurs. En raison du manque de soutien total de Spark pour les bibliothèques ci-dessus, Spark NLP a été construit sur le dessus de "Spark ML" Tirer parti du moteur de traitement de données distribué en mémoire général de Spark en tant que bibliothèque NLP de qualité professionnelle pour les flux de production critiques. Les annoter utilisent des algorithmes basés sur des règles, l'apprentissage machine et TensorFlow pour alimenter l'implémentation de l'apprentissage profond. Cela couvre les tâches NLP courantes, y compris, mais sans s'y limiter, la tokenisation, la lemmatisation, le juguler, le marquage de la partie de la parole, la reconnaissance de l'entité nommée, vérification orthographique et analyse de sentiment.

Les représentations d'encodeur bidirectionnelles des transformateurs (BERT) sont une technique d'apprentissage machine basée sur transformateur pour NLP. Elle a popularisé le concept de préformation et de réglage fin. L'architecture de transformateur de BERT provient de la traduction automatique, qui modélise les dépendances à long terme mieux que les modèles de langage RNN (réseau neuronal récurrent). Il a également introduit la tâche MLM (Modelage du langage masqué), où 15 % aléatoire de tous les jetons sont masqués et le modèle les prédit, ce qui permet une réelle bidirectionnalité.

L'analyse des sentiments financiers est difficile en raison du langage spécialisé et du manque de données étiquetées dans ce domaine. "FinBERT", Un modèle de langue basé sur une BERT pré-pré-pré-pré-pré, a été adapté au domaine "Reuters TRC2", un corpus financier, et affinée avec des données étiquetées ( "Banque de PhraseBank") pour la classification des sentiments financiers. Les chercheurs ont extrait 4, 500 phrases d'articles de presse avec des termes financiers. Puis 16 experts et maîtres étudiants ayant des antécédents financiers ont qualifié les phrases de positif, neutre et négatif. Nous avons conçu un flux de travail Spark de bout en bout pour analyser le sentiment des transcriptions d'appel des primes d'entreprise du Top 10 NASDAQ de 2016 à 2020 à l'aide de FinBERT et de deux autres pipelines pré-entraînés ( "Analyse des sentiments pour les nouvelles financières", "Expliquer le DL du document") De Spark NLP.

Le moteur de deep learning sous-jacent pour Spark NLP est TensorFlow, une plateforme open source de bout en bout pour le machine learning. Elle permet de créer des modèles facilement, de produire du ML partout et de puissants expérimentations à des fins de recherche. Par conséquent, lors de l'exécution de nos pipelines dans Spark `yarn cluster` Le mode consiste à exécuter TensorFlow distribué avec des données et une mise en parallèle des modèles sur un nœud maître, plusieurs nœuds workers et un stockage NAS monté sur le cluster.

## Horovod a distribué une formation

La validation principale de Hadoop pour les performances MapReduce est réalisée avec TeraGen, Terasort, TeraValidate et DFSIO (lecture et écriture). Les résultats de la validation TeraGen et Terasort sont présentés dans le "Tr-3969 : Solutions NetApp pour Hadoop" Pour les E-Series et dans la section « hiérarchisation du stockage » pour la AFF.

Selon les demandes des clients, nous considérons que la formation distribuée avec Spark constitue l'un des cas d'utilisation les plus importants. Dans ce document, nous avons utilisé le "Hovorod sur Spark" Pour valider les performances de Spark avec les solutions NetApp de cloud hybride, cloud et sur site à l'aide des contrôleurs de stockage FAS 100 % Flash (AFF) de NetApp, Azure NetApp Files et StorageGRID.

Le package Horovod sur Spark constitue une solution idéale pour Horovod, qui simplifie l'exécution de charges de travail d'entraînement distribuées dans les clusters Spark. Ceci permet une boucle de conception étroite dans laquelle le traitement des données, l'entraînement des modèles et l'évaluation des modèles sont réalisés dans Spark où résident les données d'entraînement et d'inférence.

Il existe deux API pour exécuter Horovod sur Spark : une API d'estimateur de haut niveau et une API d'exécution de bas niveau. Bien que les deux utilisent le même mécanisme sous-jacent pour lancer Horovod sur des exécuteurs Spark, l'API Estimator résume le traitement des données, la boucle d'entraînement des modèles, la vérification des modèles, la collecte des métriques et la formation distribuée. Nous avons utilisé les estimateurs Horovod Spark, TensorFlow et Keras pour une préparation des données de bout en bout et un workflow de formation distribué basé sur le "Vente de magasins de Gaggle Rossmann" la concurrence.

Le script `keras_spark_horovod_rossmann_estimator.py` se trouve dans la section "Scripts Python pour chaque utilisation majeure." Il contient trois parties :

- La première partie effectue diverses étapes de prétraitement des données sur un ensemble initial de fichiers CSV fournis par Kaggle et rassemblés par la communauté. Les données d'entrée sont séparées dans un kit d'entraînement par un `Validation` un sous-ensemble de données et un jeu de données test.
- La seconde partie définit un modèle DNN (réseau neuronal de deep Keras) avec une fonction d'activation sigmoïde logarithmique et un optimiseur Adam. Elle effectue également une entraînement distribué du modèle à l'aide de Horovod sur Spark.
- La troisième partie effectue des prévisions sur le dataset de test en utilisant le meilleur modèle qui minimise le jeu de validation moyenne d'erreur absolue globale. Il crée ensuite un fichier CSV de sortie.

Voir la section "« Apprentissage machine »" pour divers résultats de comparaison d'exécution.

## L'apprentissage profond multi-utilisateur utilisant Keras pour la prédiction CTR

Avec les récentes avancées des plates-formes ET des applications DE ML, une grande attention se porte maintenant sur l'apprentissage à grande échelle. Le taux de clics (CTR) est défini comme le nombre moyen de clics par cent impressions de publicités en ligne (exprimé en pourcentage). Elle est largement adoptée comme indicateur clé dans différents secteurs d'activité et champs d'application, notamment le marketing digital, la vente au détail, l'e-commerce et les fournisseurs de services. Découvrez nos "Tr-4904 : formation distribuée dans Azure - prévision de taux par clic" Pour en savoir plus sur les applications CTR et une implémentation

complète du workflow d'IA cloud avec Kubernetes, l'ETL de données distribuées et l'entraînement des modèles via DASK et CUDA ML.

Dans ce rapport technique, nous avons utilisé une variante du "Criteo Terabyte cliquez sur le jeu de données des journaux" (Voir TR-4904) pour l'apprentissage profond distribué multi-travailleurs utilisant Keras pour créer un workflow Spark avec des modèles DCN (Deep et Cross Network), en comparant ses performances en termes de fonction d'erreur de perte de journaux avec un modèle de régression logistique Spark ML de base. Le DCN capture efficacement les interactions de fonctions efficaces avec des degrés délimités, apprend des interactions hautement non linéaires, ne nécessite pas d'ingénierie des fonctions manuelles ni de recherche exhaustive et présente un faible coût de calcul.

Les données des systèmes de recommandation Web sont généralement distinctes et catégoriques, ce qui conduit à un espace important et clairsemé de problèmes pour l'exploration des fonctionnalités. Cela a limité la plupart des systèmes à grande échelle à des modèles linéaires tels que la régression logistique. Cependant, il est essentiel d'identifier les fonctionnalités prédictives fréquentes et d'explorer en même temps des caractéristiques croisées rares ou peu visibles pour faire de bonnes prévisions. Les modèles linéaires sont simples, interprétables et faciles à mettre à l'échelle, mais ils sont limités dans leur puissance expressive.

Les caractéristiques transversales, en revanche, ont été montrées significatives dans l'amélioration de l'expressivité des modèles. Malheureusement, il nécessite souvent une ingénierie des fonctionnalités manuelles ou une recherche exhaustive pour identifier ces fonctionnalités. Il est souvent difficile de généraliser les interactions de composants non visibles. L'utilisation d'un réseau neuronal transversal comme DCN évite l'ingénierie des fonctions spécifiques aux tâches en appliquant explicitement le croisement des fonctions de manière automatique. Le réseau transversal se compose de plusieurs couches, où le degré d'interactions le plus élevé est déterminé par la profondeur de la couche. Chaque couche produit des interactions d'ordre plus élevé en fonction de celles existantes et conserve les interactions des couches précédentes.

Un réseau neuronal de deep learning (DNN) promet de capturer des interactions très complexes entre plusieurs fonctionnalités. Toutefois, par rapport au DCN, il nécessite presque un ordre de grandeur plus de paramètres, est incapable de former explicitement des fonctions transversales et peut ne pas apprendre efficacement certains types d'interactions de fonctions. Le réseau transversal est efficace en termes de mémoire et facile à mettre en œuvre. L'entraînement conjoint des composants Cross et DNN capture efficacement les interactions prédictives des fonctions et fournit des performances de pointe sur le jeu de données Criteo CTR.

Un modèle DCN commence par une couche d'intégration et de superposition, suivie d'un réseau transversal et d'un réseau profond en parallèle. Elles sont ensuite suivies d'une couche de combinaison finale qui combine les sorties des deux réseaux. Vos données d'entrée peuvent être un vecteur avec des fonctions éparses et denses. Dans Spark, les deux "ml" et "mllib" les bibliothèques contiennent le type `SparseVector`. Il est donc important que les utilisateurs établissent une distinction entre les deux et soient conscients lorsqu'ils appellent leurs fonctions et méthodes respectives. Dans les systèmes de recommandation Web tels que la prédiction CTR, les entrées sont surtout des fonctions catégoriques, par exemple `'country=usa'`. Ces fonctions sont souvent codées en tant que vecteurs à chaud, par exemple `'[0,1,0, …]'`. Codage à chaud (OHE) avec `SparseVector` est utile lorsqu'il s'agit de jeux de données du monde réel avec des vocabulaires en constante évolution et en pleine croissance. Nous avons modifié des exemples dans "DeepCTR" Traiter de gros vocabulaires, créant des vecteurs d'intégration dans la couche d'intégration et de superposition de notre DCN.

Le "Criteo Afficher le jeu de données annonces" prédit le taux de clics des publicités. Il possède 13 caractéristiques entières et 26 caractéristiques catégoriques dans lesquelles chaque catégorie a une cardinalité élevée. Pour ce jeu de données, une amélioration de 0.001 dans logloperdus est pratiquement significative en raison de la grande taille d'entrée. Une légère amélioration de la précision des prévisions pour une grande base d'utilisateurs peut potentiellement conduire à une augmentation importante du chiffre d'affaires d'une entreprise. Le jeu de données contient 11 Go de journaux utilisateur sur une période de 7 jours, ce qui équivaut à environ 41 millions d'enregistrements. Nous avons utilisé Spark `dataFrame.randomSplit()function` diviser par deux les données à des fins d'entraînement (80 %), de

cross-validation (10 %) et les 10 % restants à des fins de test.

DCN a été mis en œuvre sur TensorFlow avec Keras. Il existe quatre composants principaux pour la mise en œuvre du processus de formation des modèles avec DCN :

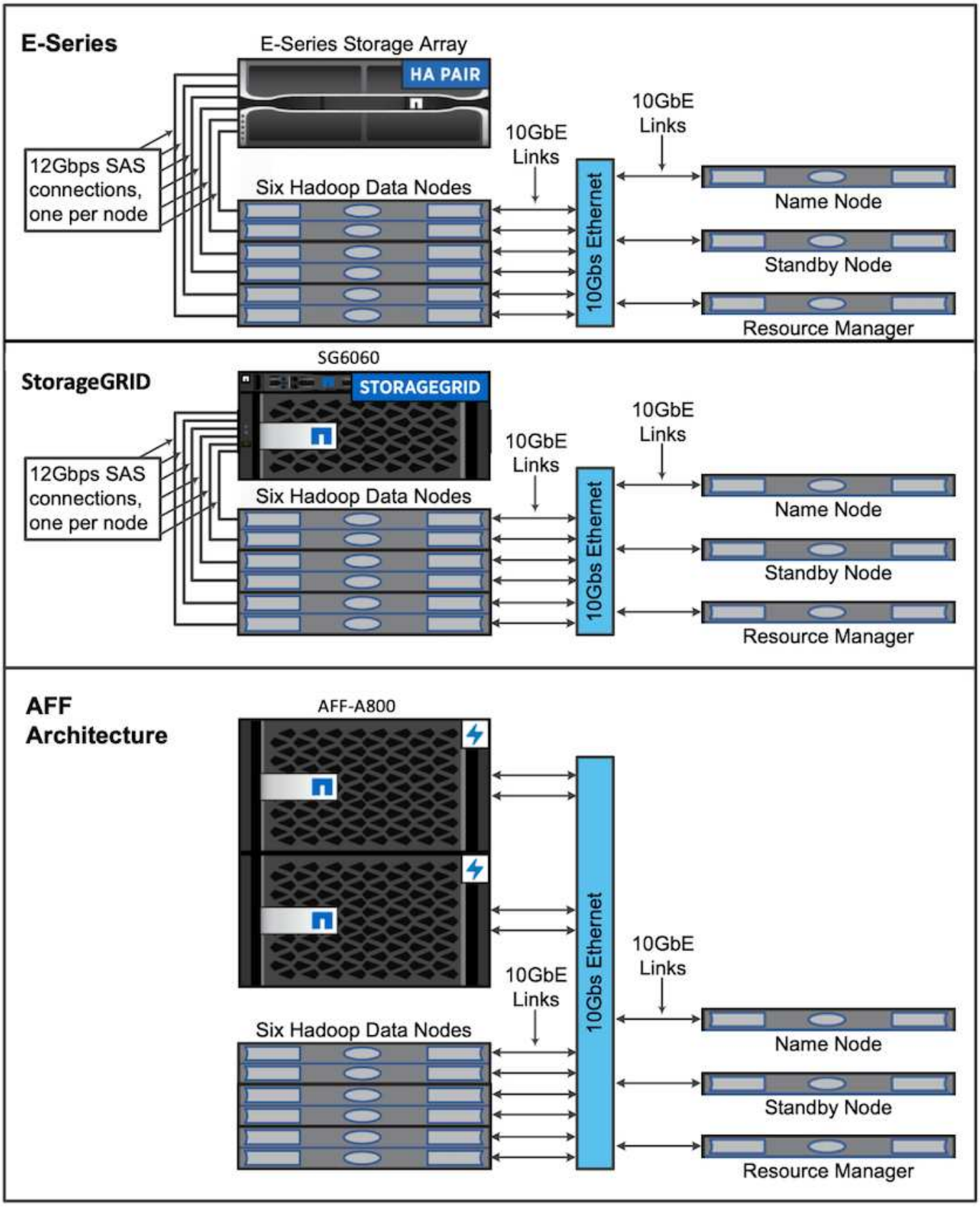
- **Traitement et incorporation de données.** les fonctions de valeur réelle sont normalisées en appliquant une transformation de journal. Pour les caractéristiques catégoriques, nous intégrons les fonctions dans les vecteurs denses de dimension 6×(cardinalité de catégorie)1/4. Le fait de concaténer tous les émudages donne un vecteur de dimension 1026.
- **Optimisation.** nous avons appliqué l'optimisation stochastique de mini-lot avec l'optimiseur Adam. La taille de batchs a été définie sur 512. La normalisation des lots a été appliquée au réseau profond et la norme de l'attache de gradient a été définie à 100.
- * Régularisation.* nous avons utilisé l'arrêt précoce, comme la régularisation L2 ou la chute n'a pas été trouvée efficace.
- **Hyperparamètres.** nous présentons des résultats basés sur une recherche de grille sur le nombre de couches masquées, la taille de couche masquée, le taux d'apprentissage initial et le nombre de couches transversales. Le nombre de couches masquées variait de 2 à 5, avec des tailles de couche cachées comprises entre 32 et 1024. Pour le DCN, le nombre de couches transversales était de 1 à 6. Le taux d'apprentissage initial a été ajusté de 0.0001 à 0.001 par incréments de 0.0001. Toutes les expériences ont appliqué un arrêt précoce à l'étape 150,000 de l'entraînement, au-delà duquel le surajustement a commencé.

En plus du DCN, nous avons également testé d'autres modèles de deep learning courants pour la prédiction CTR, notamment "DeepFM", "XDeepFM", "Int. Auto", et "DCN v2".

## Architectures utilisées pour la validation

Nous avons utilisé quatre nœuds workers et un nœud maître avec une paire HA AFF-A800. Tous les membres du cluster étaient connectés via des commutateurs réseau 10GbE.

Pour cette validation de la solution NetApp Spark, nous avons utilisé trois contrôleurs de stockage différents : E5760, E5724 et AFF-A800. Les contrôleurs de stockage E-Series ont été connectés à cinq nœuds de données avec des connexions SAS de 12 Gbit/s. Le contrôleur de stockage AFF à paire haute disponibilité fournit les volumes NFS exportés via des connexions 10 GbE vers les nœuds workers Hadoop. Les membres du cluster Hadoop ont été connectés via des connexions 10GbE dans les solutions E-Series, AFF et StorageGRID Hadoop.




## Résultats des tests

Nous avons utilisé les scripts Terasort et TeraValidate de l'outil de test TeraGen pour

mesurer la validation des performances Spark avec les configurations E5760, E5724 et AFF-A800. En outre, trois cas d'utilisation majeurs ont été testés : les pipelines Spark NLP et l'entraînement TensorFlow distribué, la formation Horovod distribuée et l'apprentissage profond multi-travailleur à l'aide de Keras pour la prédiction par CTR avec DeepFM.

Pour la validation des environnements E-Series et StorageGRID, nous avons utilisé le facteur de réplication Hadoop 2. Pour la validation AFF, nous n'utilisons qu'une seule source de données.

Le tableau suivant répertorie la configuration matérielle pour la validation des performances Spark.

| Type | Nœuds workers Hadoop | Type de disque | Disques par nœud | Contrôleur de stockage |
|---|---|---|---|---|
| SG6060 | 4 | SAS | 12 | Paire haute disponibilité unique |
| E5760 | 4 | SAS | 60 | Une seule paire HA |
| E5724 | 4 | SAS | 24 | Une seule paire HA |
| AFF800 | 4 | SSD | 6 | Une seule paire HA |

Le tableau suivant répertorie la configuration logicielle requise.

| Logiciel | Version |
|---|---|
| RHEL | 7.9 |
| Environnement d'exécution OpenJDK | 1.8.0 |
| Serveur virtuel OpenJDK 64 bits | 25.302 |
| GIT | 2.24.1 |
| GCC/G++ | 11.2.1 |
| Étincelle | 3.2.1 |
| PySpark | 3.1.2 |
| SparkNLP | 3.4.2 |
| TensorFlow | 2.9.0 |
| Keras | 2.9.0 |
| Horovod | 0.24.3 |

## Analyse du sentiment financier

Nous avons publié "Tr-4910 : analyse du sentiment issue des communications avec le client sur NetApp ai", Dans lequel un pipeline d'IA conversationnel de bout en bout a été créé à l'aide du "Kit NetApp DataOps", Stockage AFF et système NVIDIA DGX. Le pipeline exécute le traitement du signal audio par lots, la reconnaissance vocale automatique (ASR), l'apprentissage par transfert et l'analyse de sentiment en utilisant le kit DataOps, "SDK NVIDIA Riva", et le "Cadre Tao". Élargissement de l'utilisation de l'analyse des sentiments à l'industrie des services financiers, nous avons conçu un workflow SparkNLP, chargé de trois modèles BERT pour diverses tâches du NLP, telles que la reconnaissance des entités nommées et obtenu un sentiment de phrase pour les 10 meilleures entreprises du NASDAQ appels trimestriels sur les bénéfices.

Le script suivant `sentiment_analysis_spark. py` Utilise le modèle FinBERT pour traiter les transcriptions dans HDFS et produire des comptages positifs, neutres et négatifs, comme le montre le tableau suivant :

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
hdfs:///data1/Transcripts/
> ./sentiment_analysis_hdfs.log 2>&1
real13m14.300s
user557m11.319s
sys4m47.676s
```

Le tableau ci-dessous répertorie les résultats de l'analyse du sentiment au niveau des phrases pour les 10 premières entreprises du NASDAQ de 2016 à 2020.

| **Le nombre et le pourcentage de sentiments** | **Toutes les 10 entreprises** | **AAPL** | **AIRBUS** | **AMZN** | **CSCO** | **GOGL** | **INTC** | **MSFT** | **NVDA** |
|---|---|---|---|---|---|---|---|---|---|
| Comptes positifs | 7447 | 1567 | 743 | 290 | 682 | 826 | 824 | 904 | 417 |
| Compte neutre | 64067 | 6856 | 7596 | 5086 | 6650 | 5914 | 6099 | 5715 | 6189 |
| Comptes négatifs | 1787 | 253 | 213 | 84 | 189 | 97 | 282 | 202 | 89 |
| Décomptes sans catégorie | 196 | 0 | 0 | 76 | 0 | 0 | 0 | 1 | 0 |
| (nombre total) | 73497 | 8676 | 8552 | 5536 | 7521 | 6837 | 7205 | 6822 | 6695 |

En termes de pourcentages, la plupart des phrases prononcées par les PDG et les DAF sont factuelles et portent donc un sentiment neutre. Lors d'un appel sur les gains, les analystes posent des questions qui peuvent transmettre un sentiment positif ou négatif. Il est intéressant d'examiner de manière quantitative la façon dont le sentiment négatif ou positif affecte le cours des actions le même jour ou le jour suivant de la négociation.

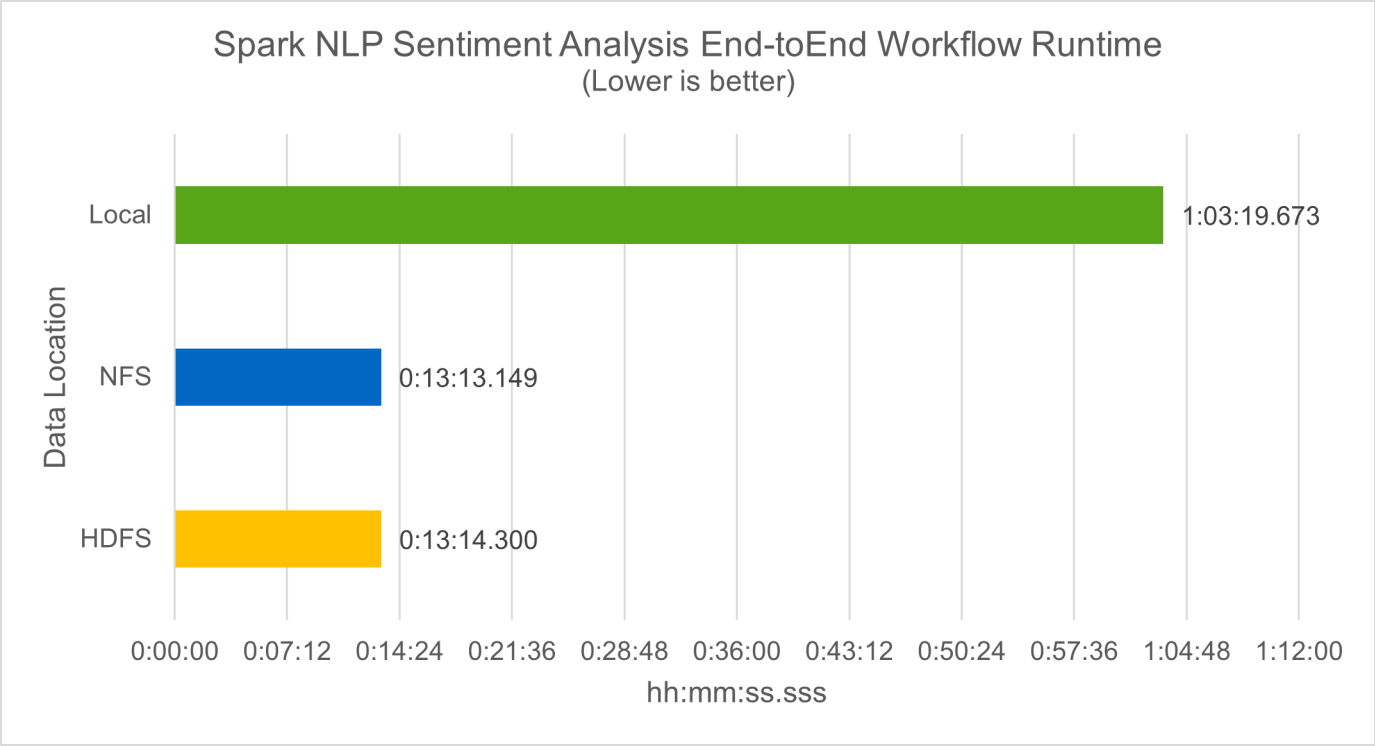
Le tableau ci-dessous répertorie les 10 premières entreprises du NASDAQ, exprimées en pourcentage, au niveau des phrases.

| Pourcentage de sentiment | Toutes les 10 entreprises | AAPL | AIRBUS | AMZN | CSCO | GOGL | INTC | MSFT | NVDA |
|---|---|---|---|---|---|---|---|---|---|
| Positif | 10.13 % | 18.06 % | 8.69 % | 5.24 % | 9.07 % | 12.08 % | 11.44 % | 13.25 % | 6.23 % |
| Neutre | 87.17 % | 79.02 % | 88.82 % | 91.87 % | 88.42 % | 86.50 % | 84.65 % | 83.77 % | 92.44 % |
| Négatif | 2.43 % | 2.92 % | 2.49 % | 1.52 % | 2.51 % | 1.42 % | 3.91 % | 2.96 % | 1.33 % |
| Sans catégorie | 0.27 % | 0 % | 0 % | 1.37 % | 0 % | 0 % | 0 % | 0.01 % | 0 % |

En termes d'exécution du flux de travail, nous avons constaté une nette amélioration de 4,6 fois par rapport à `local` Le mode vers un environnement distribué dans HDFS et une amélioration encore de 0.14 % avec NFS.

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
file:///sparkdemo/sparknlp/Transcripts/
> ./sentiment_analysis_nfs.log 2>&1
real13m13.149s
user537m50.148s
sys4m46.173s
```

Comme le montre la figure suivante, le parallélisme des données et des modèles a amélioré le traitement des données et la vitesse d'inférence des modèles TensorFlow distribués. L'emplacement des données dans NFS a permis une exécution légèrement supérieure, car le goulot d'étranglement du flux de travail correspond au téléchargement des modèles pré-entraînés. Si nous augmentons la taille des jeux de données de transcription, l'avantage du protocole NFS est plus évident.




## Formation distribuée avec la performance Horovod

La commande suivante a produit des informations d'exécution et un fichier journal dans notre cluster Spark à l'aide d'un seul `master` nœud avec 160 exécuteurs avec chacun un noyau. La mémoire de l'exécuteur était limitée à 5 Go pour éviter les erreurs de mémoire insuffisante. Voir la section "“Scripts Python pour chaque cas d'utilisation majeur”" pour obtenir plus de détails sur le traitement des données, l'entraînement du modèle et le calcul de la précision du modèle dans `keras_spark_horovod_rossmann_estimator.py`.

```
(base) [root@n138 horovod]# time spark-submit
--master local
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkusecase/horovod
--local-submission-csv /tmp/submission_0.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_local. log 2>&1
```

Le temps d'exécution résultant avec dix séries de tests d'entraînement était le suivant :

```
real43m34.608s
user12m22.057s
sys2m30.127s
```

Il fallait plus de 43 minutes pour traiter les données d'entrée, entraîner un modèle DNN, calculer la précision et produire des points de contrôle TensorFlow et un fichier CSV pour les résultats de prédiction. Nous avons limité le nombre de tests d'entraînement à 10, qui dans la pratique est souvent réglé à 100 pour assurer une précision satisfaisante du modèle. La durée d'entraînement évolue généralement de manière linéaire avec le nombre de séries de tests.

Nous avons ensuite utilisé les quatre nœuds workers disponibles dans le cluster et exécuté le même script dans `yarn` Mode avec données dans HDFS :

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir hdfs:///user/hdfs/tr-4570/experiments/horovod
--local-submission-csv /tmp/submission_1.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_yarn.log 2>&1
```

Le temps d'exécution obtenu a été amélioré comme suit :

```
real8m13.728s
user7m48.421s
sys1m26.063s
```

Avec le modèle et le parallélisme des données de Horovod dans Spark, nous avons vu une vitesse d'exécution de 5,29x `yarn` contre `local` mode avec dix séries de tests d'entraînement. Ceci est illustré dans la figure suivante avec les légendes `HDFS` et `Local`. L'entraînement du modèle DNN sous-jacent peut être accéléré au moyen de processeurs graphiques, le cas échéant. Nous prévoyons de mener ces tests et de publier les résultats dans un futur rapport technique.

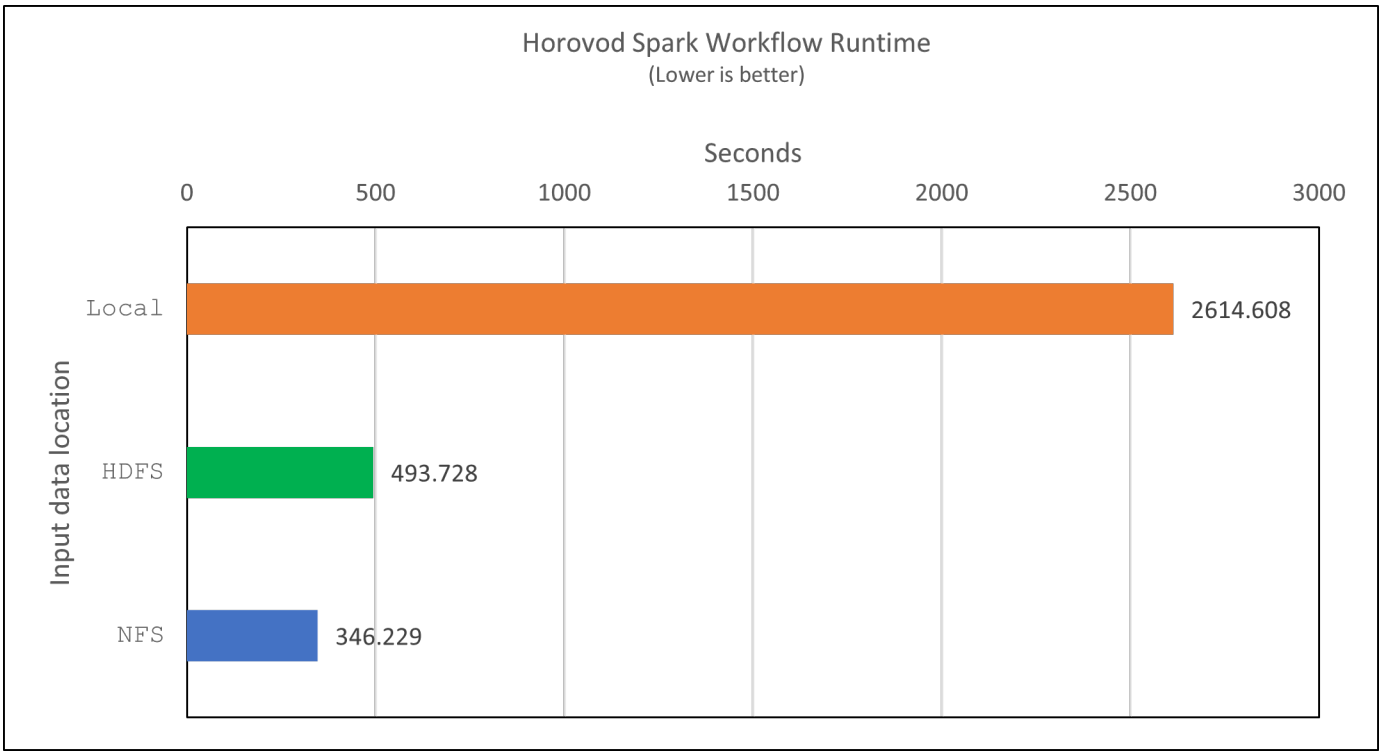
Notre prochain test a comparé les temps d'exécution avec les données d'entrée résidant dans NFS et HDFS. Le volume NFS du AFF A800 a été monté sur `/sparkdemo/horovod` Sur les cinq nœuds (un maître, quatre travailleurs) de notre cluster Spark Nous avons exécuté une commande similaire à celle des tests précédents avec `--data- dir` Paramètre maintenant pointant vers le montage NFS :

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkdemo/horovod
--local-submission-csv /tmp/submission_2.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_nfs.log 2>&1
```

Le temps d'exécution avec NFS obtenu est le suivant :

```
real 5m46.229s
user 5m35.693s
sys  1m5.615s
```

Il y a eu une accélération supplémentaire de 1,43 fois, comme le montre la figure suivante. Par conséquent, avec un système de stockage 100 % Flash NetApp connecté à leur cluster, les clients profitent des avantages du transfert et de la distribution rapides des données pour les workflows Horovod Spark, avec une vitesse de 7,5 fois supérieure à celle d'un seul nœud.



## Modèles de deep learning pour les performances de prévision CTR

Pour les systèmes de recommandation conçus pour optimiser le CTR, vous devez apprendre les interactions de fonctionnalités sophistiquées derrière les comportements utilisateur qui peuvent être calculées mathématiquement de bas en haut de gamme. Les interactions de type faible et élevé avec les fonctionnalités doivent être tout aussi importantes pour un bon modèle d'apprentissage profond, sans biasing vers l'un ou l'autre. Le Deep Factorisation machine (DeepFM), un réseau neuronal basé sur la factorisation, combine les machines d'automatisation à des fins de recommandation et d'apprentissage profond afin d'apprendre les fonctionnalités dans une nouvelle architecture de réseaux neuronaux.

Bien que les machines de factorisation conventionnelles utilisent des interactions de composants pairées en tant que produit interne de vecteurs latents entre les fonctionnalités et permettent théoriquement de capturer des informations de gros ordre, en pratique, les professionnels de l'apprentissage machine n'utilisent généralement que des interactions de fonctionnalités de second ordre du fait de la complexité élevée des calculs et du stockage. Des variantes de réseau neuronal profondes comme celle de Google "Modèles larges et profonds" en revanche, elle apprend des interactions de fonctionnalités sophistiquées dans une structure de réseau hybride en combinant un modèle à large linéaire et un modèle profond.

Il existe deux entrées pour ce modèle large et profond, l'une pour le modèle large sous-jacent et l'autre pour le plus profond, dont la dernière partie nécessite toujours une ingénierie de fonctionnalité experte et rend ainsi la technique moins généralisable pour d'autres domaines. Contrairement au modèle large et profond, DeepFM peut être efficacement formé avec des fonctions brutes sans aucune technique de fonction car sa grande partie et sa pièce profonde partagent la même entrée et le même vecteur d'intégration.

Nous avons d'abord traité le Criteo `train.txt` (11 Go) dans un fichier CSV nommé `ctr_train.csv` Stocké dans un montage NFS `/sparkdemo/tr-4570-data` à l'aide de `run_classification_criteo_spark.py` dans la section "“Scripts Python pour chaque cas d'utilisation majeur.”" Dans ce script, la fonction `process_input_file` effectue plusieurs méthodes de chaîne pour supprimer les onglets et les insérer `','` comme séparateur et `'\n'` en tant que réseau. Notez que vous n'avez besoin que de traiter l'original `train.txt` une fois, de sorte que le bloc de code soit affiché comme commentaires.

Pour les tests suivants sur les différents modèles d'apprentissage profond, nous avons utilisé `ctr_train.csv` comme fichier d'entrée. Lors des tests suivants, le fichier CSV d'entrée a été lu dans un Spark DataFrame avec un schéma contenant un champ de `'label'`, composants denses entiers `['I1', 'I2', 'I3', …, 'I13']`, et des caractéristiques parsemées `['C1', 'C2', 'C3', …, 'C26']`. Les éléments suivants `spark-submit` La commande prend dans un CSV d'entrée, forme des modèles DeepFM avec une répartition à 20 % pour la validation croisée, et sélectionne le meilleur modèle après dix séries de tests d'entraînement pour calculer la précision de prédiction sur le jeu de tests :

```
(base) [root@n138 ~]# time spark-submit --master yarn --executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py --data
-dir file:///sparkdemo/tr-4570-data >
/tmp/run_classification_criteo_spark_local.log 2>&1
```

Notez que depuis le fichier de données `ctr_train.csv` Est supérieur à 11 Go, vous devez définir une quantité suffisante `spark.driver.maxResultSize` supérieure à la taille du jeu de données pour éviter toute erreur.

```
 spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
```

Dans le ci-dessus `SparkSession.builder` configuration que nous avons également activée "Flèche Apache", Qui convertit un Spark DataFrame en un Pandas DataFrame avec le `df.toPandas()` méthode.

```
22/06/17 15:56:21 INFO scheduler.DAGScheduler: Job 2 finished: toPandas at
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py:96, took
627.126487 s
Obtained Spark DF and transformed to Pandas DF using Arrow.
```

Après la division aléatoire, le dataset d'entraînement contient plus de 36 rangées et des échantillons de 9 millions dans le dataset de test :

```
Training dataset size =  36672493
Testing dataset size =  9168124
```

Ce rapport technique étant axé sur les tests CPU sans utiliser de GPU, il est impératif de construire TensorFlow avec des indicateurs de compilateur appropriés. Cette étape évite d'appeler des bibliothèques à accélération graphique et tire pleinement parti des instructions AVX (Advanced Vector Extensions) et AVX2 de TensorFlow. Ces fonctionnalités sont conçues pour les calculs algébriques linéaires tels que l'ajout vectorisé, les multiproduits matriciels dans un entraînement DNN d'avance ou de contre-propagation. L'instruction FMA (Multiply Add) avec AVX2 utilisant des registres à virgule flottante 256 bits est idéale pour les types de code entier et de données, ce qui permet d'obtenir une vitesse de 2 fois plus élevée. Pour le code FP et les types de données, AVX2 atteint 8 % de vitesse supérieure à AVX.

```
2022-06-18 07:19:20.101478: I
tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
```

Pour créer TensorFlow à partir d'une source, NetApp vous recommande d'utiliser "Bazel". Pour notre environnement, nous avons exécuté les commandes suivantes dans l'invite du shell pour l'installation `dnf`, `dnf-plugins`, Et Bazel.

```
yum install dnf
dnf install 'dnf-command(copr)'
dnf copr enable vbatts/bazel
dnf install bazel5
```

Vous devez activer GCC 5 ou version ultérieure pour utiliser les fonctions C++17 pendant le processus de création, qui est fourni par RHEL avec la bibliothèque de collections logicielles (SCL). Les commandes suivantes s'installent `devtoolset` Et GCC 11.2.1 sur notre cluster RHEL 7.9 :

```
subscription-manager repos --enable rhel-server-rhscl-7-rpms
yum install devtoolset-11-toolchain
yum install devtoolset-11-gcc-c++
yum update
scl enable devtoolset-11 bash
. /opt/rh/devtoolset-11/enable
```

Notez que les deux dernières commandes sont en cours d'activation `devtoolset-11`, qui utilise `/opt/rh/devtoolset-11/root/usr/bin/gcc` (GCC 11.2.1). Assurez-vous également que votre `git` La version est supérieure à 1.8.3 (fournie avec RHEL 7.9). Se reporter à ceci "article" pour mise à jour `git` à 2.24.1.

Nous supposons que vous avez déjà cloné le dernier référentiel TensorFlow maître. Créez ensuite un `workspace` répertoire avec un `WORKSPACE` Fichier pour créer TensorFlow à partir de la source avec AVX, AVX2 et FMA. Exécutez le `configure` Et spécifiez l'emplacement binaire Python correct. "CUDA" Est désactivé pour nos tests car nous n'avons pas utilisé de GPU. A `.bazelrc` le fichier est généré en fonction de vos paramètres. De plus, nous avons modifié le fichier et l'ensemble `build --define=no_hdfs_support=false` Pour activer la prise en charge de HDFS. Reportez-vous à la section `.bazelrc` dans la section "“Scripts Python pour chaque cas d'utilisation majeur,”" pour obtenir une liste complète des paramètres et des indicateurs.

```
./configure
bazel build -c opt --copt=-mavx --copt=-mavx2 --copt=-mfma --copt=
-mfpmath=both -k //tensorflow/tools/pip_package:build_pip_package
```

Après avoir créé TensorFlow avec les indicateurs appropriés, exécutez le script suivant pour traiter le jeu de données Criteo Display Ads, former un modèle DeepFM et calculer la zone sous la courbe caractéristique d'exploitation du récepteur (ROC CASC) à partir des notes de prédiction.

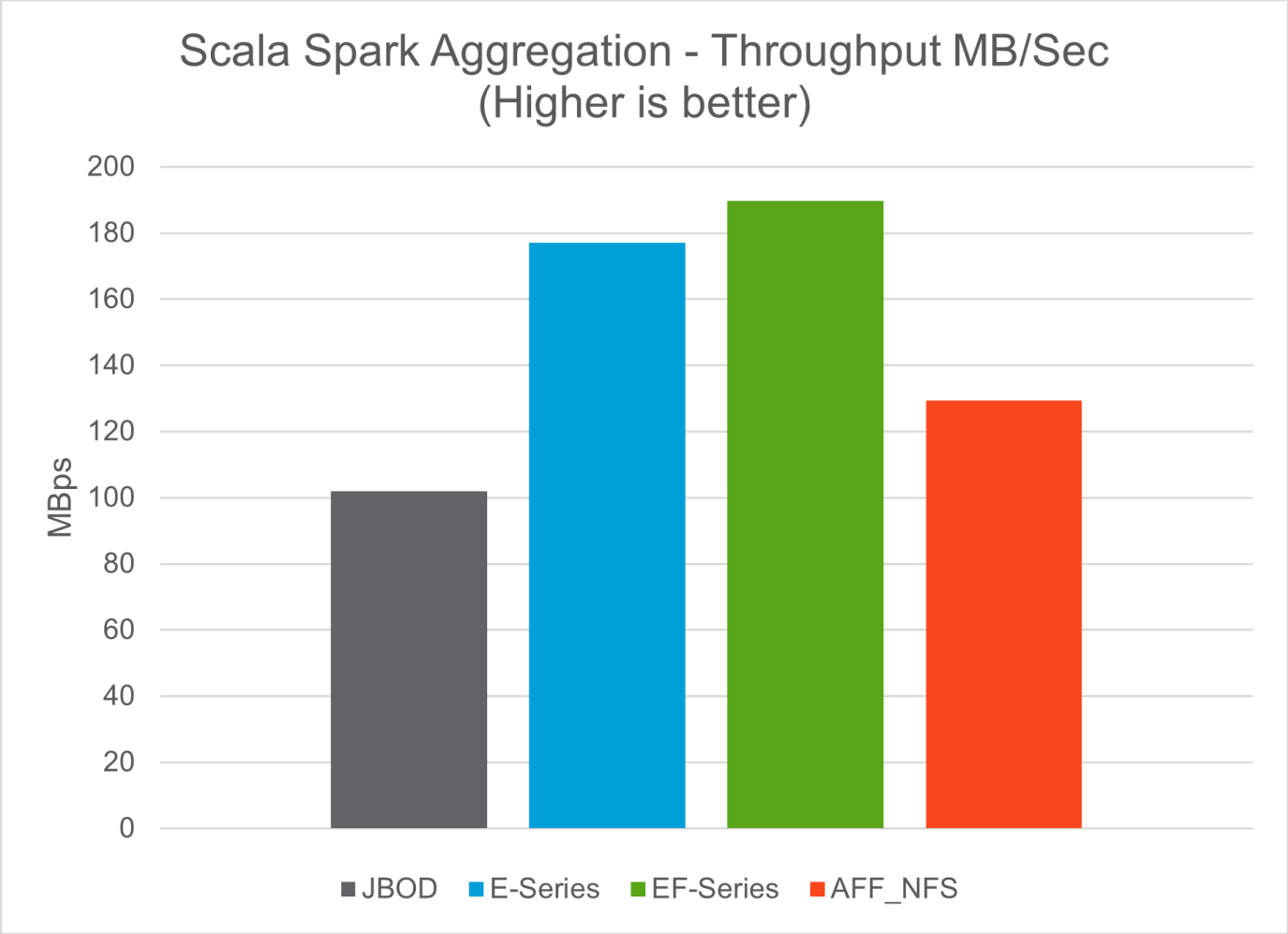
```
(base) [root@n138 examples]# ~/anaconda3/bin/spark-submit
--master yarn
--executor-memory 15g
--executor-cores 1
--num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py
--data-dir file:///sparkdemo/tr-4570-data
> . /run_classification_criteo_spark_nfs.log 2>&1
```

Après dix tests d'entraînement, nous avons obtenu le score AUC sur le jeu de données de test :

```
Epoch 1/10
125/125 - 7s - loss: 0.4976 - binary_crossentropy: 0.4974 - val_loss:
0.4629 - val_binary_crossentropy: 0.4624
Epoch 2/10
125/125 - 1s - loss: 0.3281 - binary_crossentropy: 0.3271 - val_loss:
0.5146 - val_binary_crossentropy: 0.5130
Epoch 3/10
125/125 - 1s - loss: 0.1948 - binary_crossentropy: 0.1928 - val_loss:
0.6166 - val_binary_crossentropy: 0.6144
Epoch 4/10
125/125 - 1s - loss: 0.1408 - binary_crossentropy: 0.1383 - val_loss:
0.7261 - val_binary_crossentropy: 0.7235
Epoch 5/10
125/125 - 1s - loss: 0.1129 - binary_crossentropy: 0.1102 - val_loss:
0.7961 - val_binary_crossentropy: 0.7934
Epoch 6/10
125/125 - 1s - loss: 0.0949 - binary_crossentropy: 0.0921 - val_loss:
0.9502 - val_binary_crossentropy: 0.9474
Epoch 7/10
125/125 - 1s - loss: 0.0778 - binary_crossentropy: 0.0750 - val_loss:
1.1329 - val_binary_crossentropy: 1.1301
Epoch 8/10
125/125 - 1s - loss: 0.0651 - binary_crossentropy: 0.0622 - val_loss:
1.3794 - val_binary_crossentropy: 1.3766
Epoch 9/10
125/125 - 1s - loss: 0.0555 - binary_crossentropy: 0.0527 - val_loss:
1.6115 - val_binary_crossentropy: 1.6087
Epoch 10/10
125/125 - 1s - loss: 0.0470 - binary_crossentropy: 0.0442 - val_loss:
1.6768 - val_binary_crossentropy: 1.6740
test AUC 0.6337
```

De la même manière que dans les précédents cas d'utilisation, nous avons comparé le temps d'exécution du

flux de production Spark avec des données résidant sur différents emplacements. La figure suivante montre une comparaison des prédictions CTR d'apprentissage profond pour le temps d'exécution des workflows Spark.



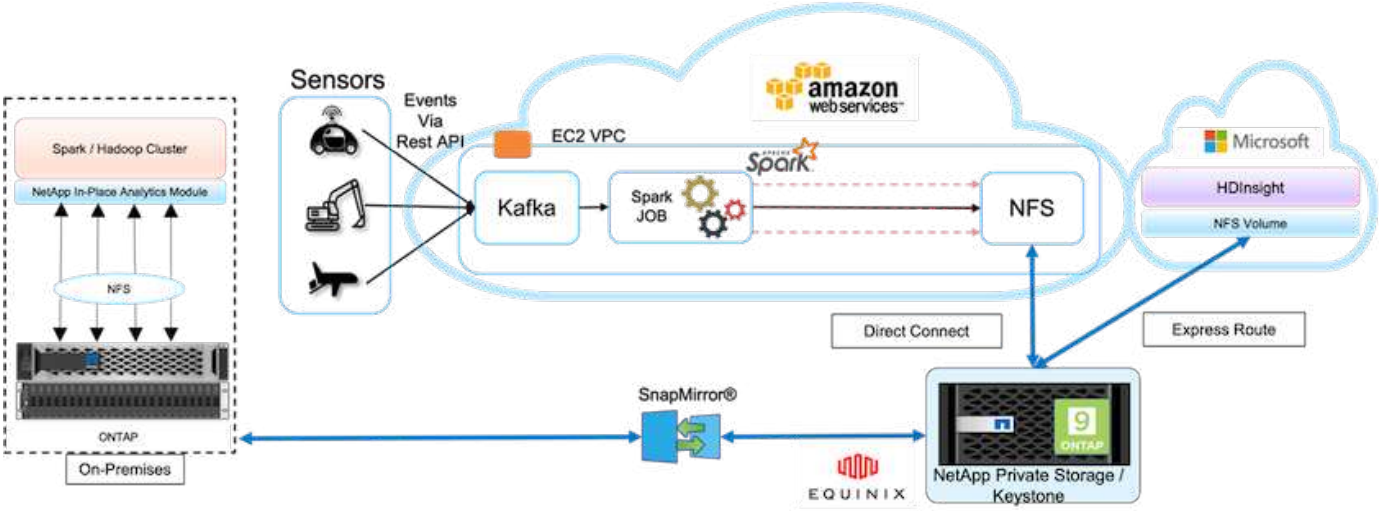

## Solution cloud hybride

Le data Center d'entreprise moderne est un cloud hybride qui connecte plusieurs environnements d'infrastructures distribuées par le biais d'un plan de gestion continue des données avec un modèle d'exploitation cohérent, sur site et/ou dans plusieurs clouds publics. Pour profiter pleinement d'un cloud hybride, vous devez pouvoir déplacer les données en toute transparence entre vos environnements sur site et multicloud sans avoir à convertir les données ou à remanier l'application.

Nous avons indiqué qu'ils commencent leur transition vers le cloud hybride, soit en déplaçant le stockage secondaire vers le cloud pour des utilisations telles que la protection des données, soit en déplaçant les workloads moins stratégiques comme le développement d'applications et le DevOps vers le cloud. Elles passent par la suite à des charges de travail plus stratégiques. L'hébergement web et de contenu, le DevOps et le développement d'applications, les bases de données, l'analytique et les applications conteneurisées font partie des workloads de cloud hybride les plus répandus. La complexité, le coût et les risques liés aux projets d'IA des entreprises ont toujours entravé l'adoption de l'IA, de l'étape expérimentale à la production.

Avec une solution de cloud hybride NetApp, les clients bénéficient d'outils intégrés de sécurité, de

gouvernance des données et de conformité avec un seul panneau de commande pour la gestion des données et des workflows dans les environnements distribués, tout en optimisant le coût total de possession en fonction de leur consommation. La figure suivante est un exemple de solution proposée par un partenaire de services clouds chargé d'offrir une connectivité multicloud pour les données d'analytique Big Data des clients.




Dans ce scénario, les données IoT reçues dans AWS de différentes sources sont stockées dans un emplacement central dans NetApp Private Storage (NPS). Le stockage NPS est connecté aux clusters Spark ou Hadoop situés dans AWS et Azure, ce qui permet aux applications d'analytique Big Data exécutées dans plusieurs clouds d'accéder aux mêmes données. Voici les principaux défis et exigences de cette utilisation :

- Les clients veulent exécuter les tâches d'analytique sur les mêmes données à l'aide de plusieurs clouds.
- Les données doivent être reçues de différentes sources, telles que les environnements sur site et cloud, par le biais de différents capteurs et concentrateurs.
- La solution doit être efficace et économique.
- Le principal défi consiste à concevoir une solution économique et efficace qui offre des services d'analytique hybride entre les différents environnements sur site et cloud.

Notre solution de protection des données et de connectivité multicloud résout le problème des applications d'analytique cloud de plusieurs hyperscalers. Comme illustré dans la figure ci-dessus, les données provenant des capteurs sont transmises et ingérées sur le cluster AWS Spark via Kafka. Les données sont stockées dans un partage NFS hébergé sur NPS, situé en dehors du fournisseur cloud au sein d'un data Center Equinix.

Étant donné que NetApp NPS est connecté à Amazon AWS et Microsoft Azure via respectivement les connexions Direct Connect et Express route, les clients peuvent utiliser le module d'analytique sur place pour accéder aux données à partir de clusters d'analytique Amazon et AWS. Par conséquent, car les systèmes de stockage sur site et NPS exécutent le logiciel ONTAP, "SnapMirror" Permet de mettre en miroir les données NPS dans le cluster sur site pour une analytique de cloud hybride entre les environnements sur site et clouds multiples.

Pour optimiser les performances, NetApp recommande généralement d'utiliser plusieurs interfaces réseau et des connexions directes ou des routes express pour accéder aux données à partir des instances cloud. Nous avons d'autres solutions de transfert de données, notamment "XCP" et "Copie et synchronisation BlueXP" Aider les clients à concevoir des clusters Spark dans le cloud hybride intégrant la cohérence applicative, sécurisés et économiques.

## Scripts Python pour chaque utilisation majeure

Les trois scripts Python suivants correspondent aux trois principaux cas d'utilisation testés. Tout d'abord `sentiment_analysis_sparknlp.py`.

```
# TR-4570 Refresh NLP testing by Rick Huang
from sys import argv
import os
import sparknlp
import pyspark.sql.functions as F
from sparknlp import Finisher
from pyspark.ml import Pipeline
from sparknlp.base import *
from sparknlp.annotator import *
from sparknlp.pretrained import PretrainedPipeline
from sparknlp import Finisher
# Start Spark Session with Spark NLP
spark = sparknlp.start()
print("Spark NLP version:")
print(sparknlp.version())
print("Apache Spark version:")
print(spark.version)
spark = sparknlp.SparkSession.builder \
     .master("yarn") \
     .appName("test_hdfs_read_write") \
     .config("spark.executor.cores", "1") \
     .config("spark.jars.packages", "com.johnsnowlabs.nlp:spark-
nlp_2.12:3.4.3")\
     .config('spark.executor.memory', '5gb') \
     .config('spark.executor.memoryOverhead','1000')\
     .config('spark.driver.memoryOverhead','1000')\
     .config("spark.sql.shuffle.partitions", "480")\
     .getOrCreate()
sc = spark.sparkContext
from pyspark.sql import SQLContext
sql = SQLContext(sc)
sqlContext = SQLContext(sc)
# Download pre-trained pipelines & sequence classifier
explain_pipeline_model = PretrainedPipeline('explain_document_dl',
lang='en').model#pipeline_sa =
PretrainedPipeline("classifierdl_bertwiki_finance_sentiment_pipeline",
lang="en")
# pipeline_finbert =
BertForSequenceClassification.loadSavedModel('/sparkusecase/bert_sequence_
classifier_finbert_en_3', spark)
```

```
sequenceClassifier = BertForSequenceClassification \
        .pretrained('bert_sequence_classifier_finbert', 'en') \
        .setInputCols(['token', 'document']) \
        .setOutputCol('class') \
        .setCaseSensitive(True) \
        .setMaxSentenceLength(512)
def process_sentence_df(data):
    # Pre-process: begin
    print("1. Begin DataFrame pre-processing...\n")
    print(f"\n\t2. Attaching DocumentAssembler Transformer to the
pipeline")
    documentAssembler = DocumentAssembler() \
        .setInputCol("text") \
        .setOutputCol("document") \
        .setCleanupMode("inplace_full")
        #.setCleanupMode("shrink", "inplace_full")
    doc_df = documentAssembler.transform(data)
    doc_df.printSchema()
    doc_df.show(truncate=50)
    # Pre-process: get rid of  blank lines
    clean_df = doc_df.withColumn("tmp", F.explode("document")) \
        .select("tmp.result").where("tmp.end !=
-1").withColumnRenamed("result", "text").dropna()
    print("[OK!] DataFrame after initial cleanup:\n")
    clean_df.printSchema()
    clean_df.show(truncate=80)
    # for FinBERT
    tokenizer = Tokenizer() \
        .setInputCols(['document']) \
        .setOutputCol('token')
    print(f"\n\t3. Attaching Tokenizer Annotator to the pipeline")
    pipeline_finbert = Pipeline(stages=[
        documentAssembler,
        tokenizer,
        sequenceClassifier
        ])
    # Use Finisher() & construct PySpark ML pipeline
    finisher = Finisher().setInputCols(["token", "lemma", "pos",
"entities"])
    print(f"\n\t4. Attaching Finisher Transformer to the pipeline")
    pipeline_ex = Pipeline() \
        .setStages([
           explain_pipeline_model,
           finisher
           ])
    print("\n\t\t\t ---- Pipeline Built Successfully ----")
```

```
    # Loading pipelines to annotate
    #result_ex_df = pipeline_ex.transform(clean_df)
    ex_model = pipeline_ex.fit(clean_df)
    annotations_finished_ex_df = ex_model.transform(clean_df)
    # result_sa_df = pipeline_sa.transform(clean_df)
    result_finbert_df = pipeline_finbert.fit(clean_df).transform(clean_df)
    print("\n\t\t\t ----Document Explain, Sentiment Analysis & FinBERT
Pipeline Fitted Successfully ----")
    # Check the result entities
    print("[OK!] Simple explain ML pipeline result:\n")
    annotations_finished_ex_df.printSchema()
    annotations_finished_ex_df.select('text',
'finished_entities').show(truncate=False)
    # Check the result sentiment from FinBERT
    print("[OK!] Sentiment Analysis FinBERT pipeline result:\n")
    result_finbert_df.printSchema()
    result_finbert_df.select('text', 'class.result').show(80, False)
    sentiment_stats(result_finbert_df)
    return
def sentiment_stats(finbert_df):
    result_df = finbert_df.select('text', 'class.result')
    sa_df = result_df.select('result')
    sa_df.groupBy('result').count().show()
    # total_lines = result_clean_df.count()
    # num_neutral = result_clean_df.where(result_clean_df.result ==
['neutral']).count()
    # num_positive = result_clean_df.where(result_clean_df.result ==
['positive']).count()
    # num_negative = result_clean_df.where(result_clean_df.result ==
['negative']).count()
    # print(f"\nRatio of neutral sentiment = {num_neutral/total_lines}")
    # print(f"Ratio of positive sentiment = {num_positive / total_lines}")
    # print(f"Ratio of negative sentiment = {num_negative /
total_lines}\n")
    return
def process_input_file(file_name):
    # Turn input file to Spark DataFrame
    print("START processing input file...")
    data_df = spark.read.text(file_name)
    data_df.show()
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    output_df.printSchema()
    return output_dfdef process_local_dir(directory):
    filelist = []
    for subdir, dirs, files in os.walk(directory):
```

```
        for filename in files:
            filepath = subdir + os.sep + filename
            print("[OK!] Will process the following files:")
            if filepath.endswith(".txt"):
                print(filepath)
                filelist.append(filepath)
    return filelist
def process_local_dir_or_file(dir_or_file):
    numfiles = 0
    if os.path.isfile(dir_or_file):
        input_df = process_input_file(dir_or_file)
        print("Obtained input_df.")
        process_sentence_df(input_df)
        print("Processed input_df")
        numfiles += 1
    else:
        filelist = process_local_dir(dir_or_file)
        for file in filelist:
            input_df = process_input_file(file)
            process_sentence_df(input_df)
            numfiles += 1
    return numfiles
def process_hdfs_dir(dir_name):
    # Turn input files to Spark DataFrame
    print("START processing input HDFS directory...")
    data_df = spark.read.option("recursiveFileLookup",
"true").text(dir_name)
    data_df.show()
    print("[DEBUG] total lines in data_df = ", data_df.count())
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    print("[DEBUG] output_df looks like: \n")
    output_df.show(40, False)
    print("[DEBUG] HDFS dir resulting data_df schema: \n")
    output_df.printSchema()
    process_sentence_df(output_df)
    print("Processed HDFS directory: ", dir_name)
    returnif __name__ == '__main__':
    try:
        if len(argv) == 2:
            print("Start processing input...\n")
    except:
        print("[ERROR] Please enter input text file or path to
process!\n")
        exit(1)
    # This is for local file, not hdfs:
```

```
    numfiles = process_local_dir_or_file(str(argv[1]))
    # For HDFS single file & directory:
    input_df = process_input_file(str(argv[1]))
    print("Obtained input_df.")
    process_sentence_df(input_df)
    print("Processed input_df")
    numfiles += 1
    # For HDFS directory of subdirectories of files:
    input_parse_list = str(argv[1]).split('/')
    print(input_parse_list)
    if input_parse_list[-2:-1] == ['Transcripts']:
        print("Start processing HDFS directory: ", str(argv[1]))
        process_hdfs_dir(str(argv[1]))
    print(f"[OK!] All done. Number of files processed = {numfiles}")
```

Le deuxième script est `keras_spark_horovod_rossmann_estimator.py`.

```
# Copyright 2022 NetApp, Inc.
# Authored by Rick Huang
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
==========================================================================
====
# The below code was modified from: https://www.kaggle.com/c/rossmann-
store-sales
import argparse
import datetime
import os
import sys
from distutils.version import LooseVersion
import pyspark.sql.types as T
import pyspark.sql.functions as F
from pyspark import SparkConf, Row
from pyspark.sql import SparkSession
```

```
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.layers import Input, Embedding, Concatenate, Dense,
Flatten, Reshape, BatchNormalization, Dropout
import horovod.spark.keras as hvd
from horovod.spark.common.backend import SparkBackend
from horovod.spark.common.store import Store
from horovod.tensorflow.keras.callbacks import BestModelCheckpoint
parser = argparse.ArgumentParser(description='Horovod Keras Spark Rossmann
Estimator Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--master',
                    help='spark cluster to use for training. If set to
None, uses current default cluster. Cluster'
                         'should be set up to provide a Spark task per
multiple CPU cores, or per GPU, e.g. by'
                         'supplying `-c <NUM_GPUS>` in Spark Standalone
mode')
parser.add_argument('--num-proc', type=int,
                    help='number of worker processes for training,
default: `spark.default.parallelism`')
parser.add_argument('--learning_rate', type=float, default=0.0001,
                    help='initial learning rate')
parser.add_argument('--batch-size', type=int, default=100,
                    help='batch size')
parser.add_argument('--epochs', type=int, default=100,
                    help='number of epochs to train')
parser.add_argument('--sample-rate', type=float,
                    help='desired sampling rate. Useful to set to low
number (e.g. 0.01) to make sure that '
                         'end-to-end process works')
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                    help='location of data on local filesystem (prefixed
with file://) or on HDFS')
parser.add_argument('--local-submission-csv', default='submission.csv',
                    help='output submission predictions CSV')
parser.add_argument('--local-checkpoint-file', default='checkpoint',
                    help='model checkpoint')
parser.add_argument('--work-dir', default='/tmp',
                    help='temporary working directory to write
intermediate files (prefix with hdfs:// to use HDFS)')
if __name__ == '__main__':
    args = parser.parse_args()
    # ================ #
    # DATA PREPARATION #
```

```
    # ================ #
    print('================')
    print('Data preparation')
    print('================')
    # Create Spark session for data preparation.
    conf = SparkConf() \
        .setAppName('Keras Spark Rossmann Estimator Example') \
        .set('spark.sql.shuffle.partitions', '480') \
        .set("spark.executor.cores", "1") \
        .set('spark.executor.memory', '5gb') \
        .set('spark.executor.memoryOverhead','1000')\
        .set('spark.driver.memoryOverhead','1000')
    if args.master:
        conf.setMaster(args.master)
    elif args.num_proc:
        conf.setMaster('local[{}]'.format(args.num_proc))
    spark = SparkSession.builder.config(conf=conf).getOrCreate()
    train_csv = spark.read.csv('%s/train.csv' % args.data_dir,
header=True)
    test_csv = spark.read.csv('%s/test.csv' % args.data_dir, header=True)
    store_csv = spark.read.csv('%s/store.csv' % args.data_dir,
header=True)
    store_states_csv = spark.read.csv('%s/store_states.csv' %
args.data_dir, header=True)
    state_names_csv = spark.read.csv('%s/state_names.csv' % args.data_dir,
header=True)
    google_trend_csv = spark.read.csv('%s/googletrend.csv' %
args.data_dir, header=True)
    weather_csv = spark.read.csv('%s/weather.csv' % args.data_dir,
header=True)
    def expand_date(df):
        df = df.withColumn('Date', df.Date.cast(T.DateType()))
        return df \
            .withColumn('Year', F.year(df.Date)) \
            .withColumn('Month', F.month(df.Date)) \
            .withColumn('Week', F.weekofyear(df.Date)) \
            .withColumn('Day', F.dayofmonth(df.Date))
    def prepare_google_trend():
        # Extract week start date and state.
        google_trend_all = google_trend_csv \
            .withColumn('Date', F.regexp_extract(google_trend_csv.week,
'(.*?) -', 1)) \
            .withColumn('State', F.regexp_extract(google_trend_csv.file,
'Rossmann_DE_(.*)', 1))
        # Map state NI -> HB,NI to align with other data sources.
        google_trend_all = google_trend_all \
```

```
            .withColumn('State', F.when(google_trend_all.State == 'NI',
'HB,NI').otherwise(google_trend_all.State))
        # Expand dates.
        return expand_date(google_trend_all)
    def add_elapsed(df, cols):
        def add_elapsed_column(col, asc):
            def fn(rows):
                last_store, last_date = None, None
                for r in rows:
                    if last_store != r.Store:
                        last_store = r.Store
                        last_date = r.Date
                    if r[col]:
                        last_date = r.Date
                    fields = r.asDict().copy()
                    fields[('After' if asc else 'Before') + col] = (r.Date
- last_date).days
                    yield Row(**fields)
            return fn
        df = df.repartition(df.Store)
        for asc in [False, True]:
            sort_col = df.Date.asc() if asc else df.Date.desc()
            rdd = df.sortWithinPartitions(df.Store.asc(), sort_col).rdd
            for col in cols:
                rdd = rdd.mapPartitions(add_elapsed_column(col, asc))
            df = rdd.toDF()
        return df
    def prepare_df(df):
        num_rows = df.count()
        # Expand dates.
        df = expand_date(df)
        df = df \
            .withColumn('Open', df.Open != '0') \
            .withColumn('Promo', df.Promo != '0') \
            .withColumn('StateHoliday', df.StateHoliday != '0') \
            .withColumn('SchoolHoliday', df.SchoolHoliday != '0')
        # Merge in store information.
        store = store_csv.join(store_states_csv, 'Store')
        df = df.join(store, 'Store')
        # Merge in Google Trend information.
        google_trend_all = prepare_google_trend()
        df = df.join(google_trend_all, ['State', 'Year',
'Week']).select(df['*'], google_trend_all.trend)
        # Merge in Google Trend for whole Germany.
        google_trend_de = google_trend_all[google_trend_all.file ==
'Rossmann_DE'].withColumnRenamed('trend', 'trend_de')
```

```
        df = df.join(google_trend_de, ['Year', 'Week']).select(df['*'],
google_trend_de.trend_de)
        # Merge in weather.
        weather = weather_csv.join(state_names_csv, weather_csv.file ==
state_names_csv.StateName)
        df = df.join(weather, ['State', 'Date'])
        # Fix null values.
        df = df \
            .withColumn('CompetitionOpenSinceYear',
F.coalesce(df.CompetitionOpenSinceYear, F.lit(1900))) \
            .withColumn('CompetitionOpenSinceMonth',
F.coalesce(df.CompetitionOpenSinceMonth, F.lit(1))) \
            .withColumn('Promo2SinceYear', F.coalesce(df.Promo2SinceYear,
F.lit(1900))) \
            .withColumn('Promo2SinceWeek', F.coalesce(df.Promo2SinceWeek,
F.lit(1)))
        # Days & months competition was open, cap to 2 years.
        df = df.withColumn('CompetitionOpenSince',
                           F.to_date(F.format_string('%s-%s-15',
df.CompetitionOpenSinceYear,

df.CompetitionOpenSinceMonth)))
        df = df.withColumn('CompetitionDaysOpen',
                           F.when(df.CompetitionOpenSinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(360 *
2), F.datediff(df.Date, df.CompetitionOpenSince))))
                           .otherwise(0))
        df = df.withColumn('CompetitionMonthsOpen',
(df.CompetitionDaysOpen / 30).cast(T.IntegerType()))
        # Days & weeks of promotion, cap to 25 weeks.
        df = df.withColumn('Promo2Since',
                           F.expr('date_add(format_string("%s-01-01",
Promo2SinceYear), (cast(Promo2SinceWeek as int) - 1) * 7)'))
        df = df.withColumn('Promo2Days',
                           F.when(df.Promo2SinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(25 *
7), F.datediff(df.Date, df.Promo2Since))))
                           .otherwise(0))
        df = df.withColumn('Promo2Weeks', (df.Promo2Days /
7).cast(T.IntegerType()))
        # Check that we did not lose any rows through inner joins.
        assert num_rows == df.count(), 'lost rows in joins'
        return df
    def build_vocabulary(df, cols):
        vocab = {}
        for col in cols:
```

```
            values = [r[0] for r in df.select(col).distinct().collect()]
            col_type = type([x for x in values if x is not None][0])
            default_value = col_type()
            vocab[col] = sorted(values, key=lambda x: x or default_value)
        return vocab
    def cast_columns(df, cols):
        for col in cols:
            df = df.withColumn(col,
F.coalesce(df[col].cast(T.FloatType()), F.lit(0.0)))
        return df
    def lookup_columns(df, vocab):
        def lookup(mapping):
            def fn(v):
                return mapping.index(v)
            return F.udf(fn, returnType=T.IntegerType())
        for col, mapping in vocab.items():
            df = df.withColumn(col, lookup(mapping)(df[col]))
        return df
    if args.sample_rate:
        train_csv = train_csv.sample(withReplacement=False,
fraction=args.sample_rate)
        test_csv = test_csv.sample(withReplacement=False,
fraction=args.sample_rate)
    # Prepare data frames from CSV files.
    train_df = prepare_df(train_csv).cache()
    test_df = prepare_df(test_csv).cache()
    # Add elapsed times from holidays & promos, the data spanning training
& test datasets.
    elapsed_cols = ['Promo', 'StateHoliday', 'SchoolHoliday']
    elapsed = add_elapsed(train_df.select('Date', 'Store', *elapsed_cols)
                          .unionAll(test_df.select('Date', 'Store',
*elapsed_cols)),
                          elapsed_cols)
    # Join with elapsed times.
    train_df = train_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(train_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    test_df = test_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(test_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    # Filter out zero sales.
    train_df = train_df.filter(train_df.Sales > 0)
    print('===================')
    print('Prepared data frame')
```

```
    print('===================')
    train_df.show()
    categorical_cols = [
        'Store', 'State', 'DayOfWeek', 'Year', 'Month', 'Day', 'Week',
'CompetitionMonthsOpen', 'Promo2Weeks', 'StoreType',
        'Assortment', 'PromoInterval', 'CompetitionOpenSinceYear',
'Promo2SinceYear', 'Events', 'Promo',
        'StateHoliday', 'SchoolHoliday'
    ]
    continuous_cols = [
        'CompetitionDistance', 'Max_TemperatureC', 'Mean_TemperatureC',
'Min_TemperatureC', 'Max_Humidity',
        'Mean_Humidity', 'Min_Humidity', 'Max_Wind_SpeedKm_h',
'Mean_Wind_SpeedKm_h', 'CloudCover', 'trend', 'trend_de',
        'BeforePromo', 'AfterPromo', 'AfterStateHoliday',
'BeforeStateHoliday', 'BeforeSchoolHoliday', 'AfterSchoolHoliday'
    ]
    all_cols = categorical_cols + continuous_cols
    # Select features.
    train_df = train_df.select(*(all_cols + ['Sales', 'Date'])).cache()
    test_df = test_df.select(*(all_cols + ['Id', 'Date'])).cache()
    # Build vocabulary of categorical columns.
    vocab = build_vocabulary(train_df.select(*categorical_cols)

.unionAll(test_df.select(*categorical_cols)).cache(),
                              categorical_cols)
    # Cast continuous columns to float & lookup categorical columns.
    train_df = cast_columns(train_df, continuous_cols + ['Sales'])
    train_df = lookup_columns(train_df, vocab)
    test_df = cast_columns(test_df, continuous_cols)
    test_df = lookup_columns(test_df, vocab)
    # Split into training & validation.
    # Test set is in 2015, use the same period in 2014 from the training
set as a validation set.
    test_min_date = test_df.agg(F.min(test_df.Date)).collect()[0][0]
    test_max_date = test_df.agg(F.max(test_df.Date)).collect()[0][0]
    one_year = datetime.timedelta(365)
    train_df = train_df.withColumn('Validation',
                                   (train_df.Date > test_min_date -
one_year) & (train_df.Date <= test_max_date - one_year))
    # Determine max Sales number.
    max_sales = train_df.agg(F.max(train_df.Sales)).collect()[0][0]
    # Convert Sales to log domain
    train_df = train_df.withColumn('Sales', F.log(train_df.Sales))
    print('===================================')
    print('Data frame with transformed columns')
```

```
    print('====================================')
    train_df.show()
    print('================')
    print('Data frame sizes')
    print('================')
    train_rows = train_df.filter(~train_df.Validation).count()
    val_rows = train_df.filter(train_df.Validation).count()
    test_rows = test_df.count()
    print('Training: %d' % train_rows)
    print('Validation: %d' % val_rows)
    print('Test: %d' % test_rows)
    # ============== #
    # MODEL TRAINING #
    # ============== #
    print('==============')
    print('Model training')
    print('==============')
    def exp_rmspe(y_true, y_pred):
        """Competition evaluation metric, expects logarithic inputs."""
        pct = tf.square((tf.exp(y_true) - tf.exp(y_pred)) /
tf.exp(y_true))
        # Compute mean excluding stores with zero denominator.
        x = tf.reduce_sum(tf.where(y_true > 0.001, pct,
tf.zeros_like(pct)))
        y = tf.reduce_sum(tf.where(y_true > 0.001, tf.ones_like(pct),
tf.zeros_like(pct)))
        return tf.sqrt(x / y)
    def act_sigmoid_scaled(x):
        """Sigmoid scaled to logarithm of maximum sales scaled by 20%."""
        return tf.nn.sigmoid(x) * tf.math.log(max_sales) * 1.2
    CUSTOM_OBJECTS = {'exp_rmspe': exp_rmspe,
                      'act_sigmoid_scaled': act_sigmoid_scaled}
    # Disable GPUs when building the model to prevent memory leaks
    if LooseVersion(tf.__version__) >= LooseVersion('2.0.0'):
        # See https://github.com/tensorflow/tensorflow/issues/33168
        os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
    else:

K.set_session(tf.Session(config=tf.ConfigProto(device_count={'GPU': 0})))
    # Build the model.
    inputs = {col: Input(shape=(1,), name=col) for col in all_cols}
    embeddings = [Embedding(len(vocab[col]), 10, input_length=1,
name='emb_' + col)(inputs[col])
                  for col in categorical_cols]
    continuous_bn = Concatenate()([Reshape((1, 1), name='reshape_' +
col)(inputs[col])
```

```
                                    for col in continuous_cols])
    continuous_bn = BatchNormalization()(continuous_bn)
    x = Concatenate()(embeddings + [continuous_bn])
    x = Flatten()(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(500, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation=act_sigmoid_scaled)(x)
    model = tf.keras.Model([inputs[f] for f in all_cols], output)
    model.summary()
    opt = tf.keras.optimizers.Adam(lr=args.learning_rate, epsilon=1e-3)
    # Checkpoint callback to specify options for the returned Keras model
    ckpt_callback = BestModelCheckpoint(monitor='val_loss', mode='auto',
save_freq='epoch')
    # Horovod: run training.
    store = Store.create(args.work_dir)
    backend = SparkBackend(num_proc=args.num_proc,
                           stdout=sys.stdout, stderr=sys.stderr,
                           prefix_output_with_timestamp=True)
    keras_estimator = hvd.KerasEstimator(backend=backend,
                                         store=store,
                                         model=model,
                                         optimizer=opt,
                                         loss='mae',
                                         metrics=[exp_rmspe],
                                         custom_objects=CUSTOM_OBJECTS,
                                         feature_cols=all_cols,
                                         label_cols=['Sales'],
                                         validation='Validation',
                                         batch_size=args.batch_size,
                                         epochs=args.epochs,
                                         verbose=2,

checkpoint_callback=ckpt_callback)
    keras_model =
keras_estimator.fit(train_df).setOutputCols(['Sales_output'])
    history = keras_model.getHistory()
    best_val_rmspe = min(history['val_exp_rmspe'])
    print('Best RMSPE: %f' % best_val_rmspe)
    # Save the trained model.
```

```
    keras_model.save(args.local_checkpoint_file)
    print('Written checkpoint to %s' % args.local_checkpoint_file)
    # ================ #
    # FINAL PREDICTION #
    # ================ #
    print('================')
    print('Final prediction')
    print('================')
    pred_df=keras_model.transform(test_df)
    pred_df.printSchema()
    pred_df.show(5)
    # Convert from log domain to real Sales numbers
    pred_df=pred_df.withColumn('Sales_pred', F.exp(pred_df.Sales_output))
    submission_df = pred_df.select(pred_df.Id.cast(T.IntegerType()),
pred_df.Sales_pred).toPandas()
    submission_df.sort_values(by=['Id']).to_csv(args.local_submission_csv,
index=False)
    print('Saved predictions to %s' % args.local_submission_csv)
    spark.stop()
```

Le troisième script est `run_classification_criteo_spark.py`.

```
import tempfile, string, random, os, uuid
import argparse, datetime, sys, shutil
import csv
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from pyspark import SparkContext
from pyspark.sql import SparkSession, SQLContext, Row, DataFrame
from pyspark.mllib import linalg as mllib_linalg
from pyspark.mllib.linalg import SparseVector as mllibSparseVector
from pyspark.mllib.linalg import VectorUDT as mllibVectorUDT
from pyspark.mllib.linalg import Vector as mllibVector, Vectors as
mllibVectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.ml import linalg as ml_linalg
from pyspark.ml.linalg import VectorUDT as mlVectorUDT
from pyspark.ml.linalg import SparseVector as mlSparseVector
from pyspark.ml.linalg import Vector as mlVector, Vectors as mlVectors
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import OneHotEncoder
from math import log
from math import exp  # exp(-t) = e^-t
```

```
from operator import add
from pyspark.sql.functions import udf, split, lit
from pyspark.sql.functions import size, sum as sqlsum
import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.sql.types import ArrayType, StructType, StructField,
LongType, StringType, IntegerType, FloatType
from pyspark.sql.functions import explode, col, log, when
from collections import defaultdict
import pandas as pd
import pyspark.pandas as ps
from sklearn.metrics import log_loss, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from deepctr.models import DeepFM
from deepctr.feature_column import SparseFeat, DenseFeat,
get_feature_names
spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
# spark.conf.set("spark.sql.execution.arrow.enabled", "true") # deprecated
print("Apache Spark version:")
print(spark.version)
sc = spark.sparkContext
sqlContext = SQLContext(sc)
parser = argparse.ArgumentParser(description='Spark DCN CTR Prediction
Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                    help='location of data on local filesystem (prefixed
with file://) or on HDFS')
def process_input_file(file_name, sparse_feat, dense_feat):
    # Need this preprocessing to turn Criteo raw file into CSV:
    print("START processing input file...")
    # only convert the file ONCE
```

```
    # sample = open(file_name)
    # sample = '\n'.join([str(x.replace('\n', '').replace('\t', ',')) for
x in sample])
    # # Add header in data file and save as CSV
    # header = ','.join(str(x) for x in (['label'] + dense_feat +
sparse_feat))
    # with open('/sparkdemo/tr-4570-data/ctr_train.csv', mode='w',
encoding="utf-8") as f:
    #     f.write(header + '\n' + sample)
    #     f.close()
    # print("Raw training file processed and saved as CSV: ", f.name)
    raw_df = sqlContext.read.option("header", True).csv(file_name)
    raw_df.show(5, False)
    raw_df.printSchema()
    # convert columns I1 to I13 from string to integers
    conv_df = raw_df.select(col('label').cast("double"),
                            *(col(i).cast("float").alias(i) for i in
raw_df.columns if i in dense_feat),
                            *(col(c) for c in raw_df.columns if c in
sparse_feat))
    print("Schema of raw_df with integer columns type changed:")
    conv_df.printSchema()
    # result_pdf = conv_df.select("*").toPandas()
    tmp_df = conv_df.na.fill(0, dense_feat)
    result_df = tmp_df.na.fill('-1', sparse_feat)
    result_df.show()
    return result_df
if __name__ == "__main__":
    args = parser.parse_args()
    # Pandas read CSV
    # data = pd.read_csv('%s/criteo_sample.txt' % args.data_dir)
    # print("Obtained Pandas df.")
    dense_features = ['I' + str(i) for i in range(1, 14)]
    sparse_features = ['C' + str(i) for i in range(1, 27)]
    # Spark read CSV
    # process_input_file('%s/train.txt' % args.data_dir, sparse_features,
dense_features) # run only ONCE
    spark_df = process_input_file('%s/data.txt' % args.data_dir,
sparse_features, dense_features) # sample data
    # spark_df = process_input_file('%s/ctr_train.csv' % args.data_dir,
sparse_features, dense_features)
    print("Obtained Spark df and filled in missing features.")
    data = spark_df
    # Pandas
    #data[sparse_features] = data[sparse_features].fillna('-1', )
    #data[dense_features] = data[dense_features].fillna(0, )
```

```
    target = ['label']
    label_npa = data.select("label").toPandas().to_numpy()
    print("label numPy array has length = ", len(label_npa)) # 45,840,617
w/ 11GB dataset
    label_npa.ravel()
    label_npa.reshape(len(label_npa), )
    # 1.Label Encoding for sparse features,and do simple Transformation
for dense features
    print("Before LabelEncoder():")
    data.printSchema()  # label: float (nullable = true)
    for feat in sparse_features:
        lbe = LabelEncoder()
        tmp_pdf = data.select(feat).toPandas().to_numpy()
        tmp_ndarray = lbe.fit_transform(tmp_pdf)
        print("After LabelEncoder(), tmp_ndarray[0] =", tmp_ndarray[0])
        # print("Data tmp PDF after lbe transformation, the output ndarray
has length = ", len(tmp_ndarray)) # 45,840,617 for 11GB dataset
        tmp_ndarray.ravel()
        tmp_ndarray.reshape(len(tmp_ndarray), )
        out_ndarray = np.column_stack([label_npa, tmp_ndarray])
        pdf = pd.DataFrame(out_ndarray, columns=['label', feat])
        s_df = spark.createDataFrame(pdf)
        s_df.printSchema() # label: double (nullable = true)
        print("Before joining data df with s_df, s_df example rows:")
        s_df.show(1, False)
        data = data.drop(feat).join(s_df, 'label').drop('label')
        print("After LabelEncoder(), data df example rows:")
        data.show(1, False)
        print("Finished processing sparse_features: ", feat)
    print("Data DF after label encoding: ")
    data.show()
    data.printSchema()
    mms = MinMaxScaler(feature_range=(0, 1))
    # data[dense_features] = mms.fit_transform(data[dense_features]) # for
Pandas df
    tmp_pdf = data.select(dense_features).toPandas().to_numpy()
    tmp_ndarray = mms.fit_transform(tmp_pdf)
    tmp_ndarray.ravel()
    tmp_ndarray.reshape(len(tmp_ndarray), len(tmp_ndarray[0]))
    out_ndarray = np.column_stack([label_npa, tmp_ndarray])
    pdf = pd.DataFrame(out_ndarray, columns=['label'] + dense_features)
    s_df = spark.createDataFrame(pdf)
    s_df.printSchema()
    data.drop(*dense_features).join(s_df, 'label').drop('label')
    print("Finished processing dense_features: ", dense_features)
    print("Data DF after MinMaxScaler: ")
```

```
    data.show()

    # 2.count #unique features for each sparse field,and record dense
feature field name
    fixlen_feature_columns = [SparseFeat(feat,
vocabulary_size=data.select(feat).distinct().count() + 1, embedding_dim=4)
                              for i, feat in enumerate(sparse_features)] +
\
                             [DenseFeat(feat, 1, ) for feat in
dense_features]
    dnn_feature_columns = fixlen_feature_columns
    linear_feature_columns = fixlen_feature_columns
    feature_names = get_feature_names(linear_feature_columns +
dnn_feature_columns)
    # 3.generate input data for model
    # train, test = train_test_split(data.toPandas(), test_size=0.2,
random_state=2020) # Pandas; might hang for 11GB data
    train, test = data.randomSplit(weights=[0.8, 0.2], seed=200)
    print("Training dataset size = ", train.count())
    print("Testing dataset size = ", test.count())
    # Pandas:
    # train_model_input = {name: train[name] for name in feature_names}
    # test_model_input = {name: test[name] for name in feature_names}
    # Spark DF:
    train_model_input = {}
    test_model_input = {}
    for name in feature_names:
        if name.startswith('I'):
            tr_pdf = train.select(name).toPandas()
            train_model_input[name] = pd.to_numeric(tr_pdf[name])
            ts_pdf = test.select(name).toPandas()
            test_model_input[name] = pd.to_numeric(ts_pdf[name])
    # 4.Define Model,train,predict and evaluate
    model = DeepFM(linear_feature_columns, dnn_feature_columns,
task='binary')
    model.compile("adam", "binary_crossentropy",
                  metrics=['binary_crossentropy'], )
    lb_pdf = train.select(target).toPandas()
    history = model.fit(train_model_input,
pd.to_numeric(lb_pdf['label']).values,
                        batch_size=256, epochs=10, verbose=2,
validation_split=0.2, )
    pred_ans = model.predict(test_model_input, batch_size=256)
    print("test LogLoss",
round(log_loss(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
```

```
    print("test AUC",
round(roc_auc_score(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
```

## Conclusion

Dans ce document, nous présenterons l'architecture Apache Spark, les utilisations client et le portefeuille de solutions de stockage NetApp en ce qui concerne le Big Data, l'analytique moderne, l'IA, LE ML et le DL. Dans le cadre de tests de validation des performances basés sur des outils de banc d'essai standard et sur la demande client, les solutions NetApp Spark offrent des performances supérieures à celles des systèmes Hadoop natifs. La combinaison des cas d'utilisation clients et des résultats de performances présentés dans ce rapport peut vous aider à choisir la solution Spark la mieux adaptée à votre déploiement.

## Où trouver des informations complémentaires

Ce rapport technique a utilisé les références suivantes :

- Architecture et composants d'Apache Spark

  "http://spark.apache.org/docs/latest/cluster-overview.html"

- Cas d'utilisation d'Apache Spark

  "https://www.qubole.com/blog/big-data/apache-spark-use-cases/"

- Les défis d'Apache

  "http://www.infoworld.com/article/2897287/big-data/5-reasons-to-turn-to-spark-for-big-data-analytics.html"

- Spark NLP

  "https://www.johnsnowlabs.com/spark-nlp/"

- BERT

  "https://arxiv.org/abs/1810.04805"

- Deep et Cross Network pour les annonces cliquez sur les prédictions

  "https://arxiv.org/abs/1708.05123"

- FlexGroup

  "http://www.netapp.com/us/media/tr-4557.pdf"

- ETL de diffusion en continu

  "https://www.infoq.com/articles/apache-spark-streaming"

- Solutions NetApp E-Series pour Hadoop

  "https://www.netapp.com/media/16420-tr-3969.pdf"

- Analyse des sentiments des communications avec NetApp ai

  "https://docs.netapp.com/us-en/netapp-solutions/pdfs/sidebar/Sentiment_analysis_with_NetApp_AI.pdf"

- Solutions NetApp d'analytique moderne

  "https://docs.netapp.com/us-en/netapp-solutions/data-analytics/index.html"

- SnapMirror

  "https://docs.netapp.com/us-en/ontap/data-protection/snapmirror-replication-concept.html"

- XCP

  https://mysupport.netapp.com/documentation/docweb/index.html?productID=63942&language=en-US

- Copie et synchronisation BlueXP

  "https://cloud.netapp.com/cloud-sync-service"

- Kit d'outils DataOps

  "https://github.com/NetApp/netapp-dataops-toolkit"