# Soluzioni storage NetApp per Apache Spark

NetApp Solutions

NetApp
April 26, 2024

# Sommario

# Soluzioni storage NetApp per Apache Spark

## TR-4570: Soluzioni storage NetApp per Apache Spark: Architettura, casi di utilizzo e risultati delle performance

Rick Huang, Karthikeyan Nagalingam, NetApp

Questo documento si concentra sull'architettura di Apache Spark, sui casi di utilizzo dei clienti e sul portfolio di storage NetApp relativi all'analisi dei big data e all'intelligenza artificiale (ai). Presenta inoltre diversi risultati di test utilizzando strumenti di ai standard di settore, machine learning (ML) e deep learning (DL) rispetto a un sistema Hadoop tipico, in modo da poter scegliere la soluzione Spark appropriata. Per iniziare, è necessaria un'architettura Spark, componenti appropriati e due modalità di implementazione (cluster e client).

Questo documento fornisce anche casi di utilizzo per i clienti per risolvere i problemi di configurazione e illustra una panoramica del portfolio di storage NetApp relativo all'analisi dei big data e ai, ML e DL con Spark. Concludiamo con i risultati dei test derivati dai casi di utilizzo specifici di Spark e dal portfolio di soluzioni NetApp Spark.

### Sfide per i clienti

Questa sezione si concentra sulle sfide dei clienti con analisi dei big data e ai/ML/DL nei settori di crescita dei dati come retail, digital marketing, banking, produzione discreta, produzione di processi, enti pubblici e servizi professionali.

#### Performance imprevedibili

Le implementazioni Hadoop tradizionali utilizzano generalmente hardware commodity. Per migliorare le performance, devi mettere a punto la rete, il sistema operativo, il cluster Hadoop, i componenti dell'ecosistema come Spark e l'hardware. Anche se si sintonizzano ciascun livello, può essere difficile raggiungere i livelli di performance desiderati perché Hadoop viene eseguito su hardware commodity non progettato per le performance elevate nel proprio ambiente.

#### Guasti dei supporti e dei nodi

Anche in condizioni normali, l'hardware commodity è soggetto a guasti. Se un disco su un nodo dati si guasta, il master Hadoop considera il nodo non integro per impostazione predefinita. Quindi, copia dati specifici da quel nodo in rete dalle repliche a un nodo integro. Questo processo rallenta i pacchetti di rete per qualsiasi job Hadoop. Il cluster deve quindi copiare di nuovo i dati e rimuovere i dati replicati in eccesso quando il nodo non integro torna allo stato integro.

#### Lock-in del vendor Hadoop

I distributori Hadoop dispongono di una propria distribuzione Hadoop con il proprio controllo delle versioni, che blocca il cliente a tali distribuzioni. Tuttavia, molti clienti richiedono supporto per l'analisi in-memory che non colleghi il cliente a specifiche distribuzioni Hadoop. Hanno bisogno della libertà di cambiare le distribuzioni e di portare con sé le loro analisi.

### Mancanza di supporto per più di una lingua

I clienti spesso richiedono il supporto di più lingue oltre ai programmi Java MapReduce per eseguire i propri lavori. Opzioni come SQL e script offrono maggiore flessibilità per ottenere risposte, più opzioni per l'organizzazione e il recupero dei dati e metodi più rapidi per spostare i dati in un framework di analisi.

### Difficoltà di utilizzo

Per qualche tempo, si lamenta che Hadoop è difficile da utilizzare. Anche se Hadoop è diventato più semplice e potente con ogni nuova versione, questa critica ha persistito. Hadoop richiede di comprendere i modelli di programmazione Java e MapReduce, una sfida per gli amministratori di database e le persone con competenze di scripting tradizionali.

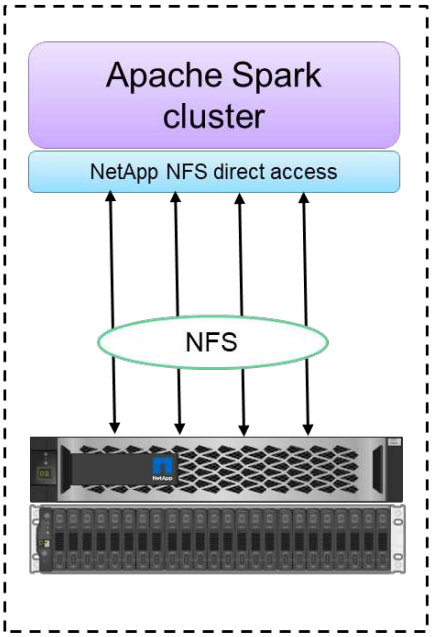
### Framework e strumenti complessi

I team ai delle aziende devono affrontare diverse sfide. Anche con una conoscenza esperta di data science, gli strumenti e i framework per diversi ecosistemi di implementazione e applicazioni potrebbero non tradursi semplicemente da uno all'altro. Una piattaforma per la scienza dei dati dovrebbe integrarsi perfettamente con le corrispondenti piattaforme per i big data basate su Spark con facilità di spostamento dei dati, modelli riutilizzabili, codice pronto all'uso e strumenti che supportino le Best practice per la prototipazione, la convalida, il controllo delle versioni, la condivisione, il riutilizzo, e implementazione rapida dei modelli in produzione.

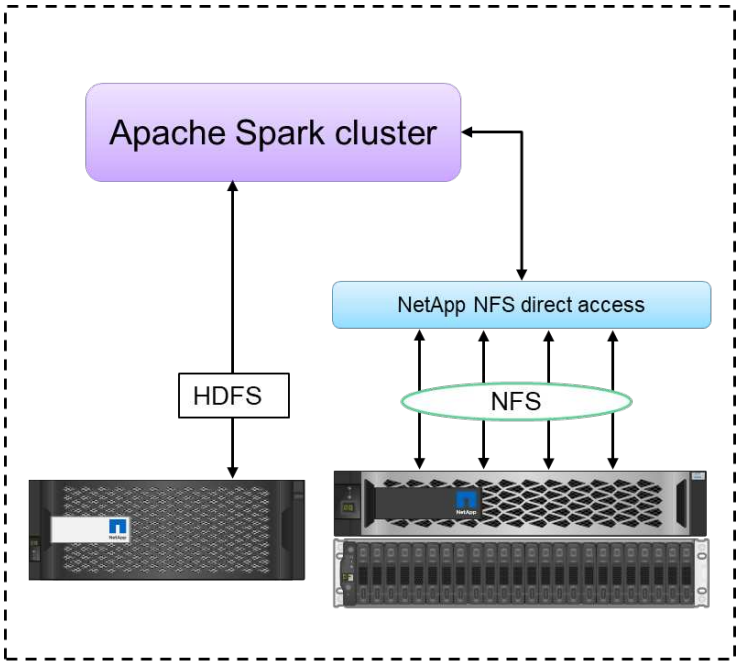
## Perché scegliere NetApp?

NetApp può migliorare la tua esperienza con Spark nei seguenti modi:

- L'accesso diretto NetApp NFS (mostrato nella figura seguente) consente ai clienti di eseguire lavori di big data analytics sui dati NFSv3 o NFSv4 esistenti o nuovi senza spostare o copiare i dati. Impedisce più copie di dati ed elimina la necessità di sincronizzare i dati con un'origine.
- Storage più efficiente e minore replica dei server. Ad esempio, la soluzione NetApp e-Series Hadoop richiede due repliche anziché tre dei dati, mentre la soluzione FAS Hadoop richiede un'origine dati ma non una replica o copie dei dati. Le soluzioni di storage NetApp producono inoltre meno traffico server-server.
- Migliore comportamento del cluster e del job Hadoop durante il guasto di disco e nodo.
- Migliori performance di acquisizione dei dati.




Configuration 1: NFS as primary storage




Configuration 2: HDFS and NFS in single Spark cluster

Ad esempio, nel settore finanziario e sanitario, lo spostamento dei dati da un luogo all'altro deve soddisfare gli obblighi legali, il che non è un compito facile. In questo scenario, l'accesso diretto NetApp NFS analizza i dati finanziari e sanitari dalla posizione originale. Un altro vantaggio chiave è che l'utilizzo dell'accesso diretto NetApp NFS semplifica la protezione dei dati Hadoop utilizzando i comandi Hadoop nativi e abilitando i flussi di lavoro per la protezione dei dati con il ricco portfolio di gestione dei dati di NetApp.

L'accesso diretto NetApp NFS offre due tipi di opzioni di implementazione per i cluster Hadoop/Spark:

- Per impostazione predefinita, i cluster Hadoop o Spark utilizzano HDFS (Distributed file System) di Hadoop per lo storage dei dati e il file system predefinito. L'accesso diretto NetApp NFS può sostituire l'HDFS predefinito con lo storage NFS come file system predefinito, consentendo l'analisi diretta dei dati NFS.
- In un'altra opzione di implementazione, l'accesso diretto NetApp NFS supporta la configurazione di NFS come storage aggiuntivo insieme a HDFS in un singolo cluster Hadoop o Spark. In questo caso, il cliente può condividere i dati attraverso le esportazioni NFS e accedervi dallo stesso cluster insieme ai dati HDFS.

I vantaggi principali dell'utilizzo dell'accesso diretto NetApp NFS includono:

- Analisi dei dati dalla posizione corrente, che impedisce il dispendioso in termini di tempo e performance dello spostamento dei dati di analisi in un'infrastruttura Hadoop come HDFS.
- Riduzione del numero di repliche da tre a uno.
- Consentendo agli utenti di separare calcolo e storage per scalare in modo indipendente.
- Protezione dei dati aziendali sfruttando le ricche funzionalità di gestione dei dati di ONTAP.
- Certificazione con la piattaforma dati Hortonworks.
- Implementazione di data analytics ibridi.
- Riduzione dei tempi di backup sfruttando la funzionalità multithread dinamica.

Vedere "TR-4657: Soluzioni dati di cloud ibrido NetApp - Spark e Hadoop in base ai casi di utilizzo dei clienti" Per il backup dei dati Hadoop, il backup e il disaster recovery dal cloud al on-premise, l'abilitazione di DevTest sui dati Hadoop esistenti, la protezione dei dati e la connettività multicloud e l'accelerazione dei carichi di

lavoro di analytics.

Le sezioni seguenti descrivono le funzionalità di storage importanti per i clienti Spark.

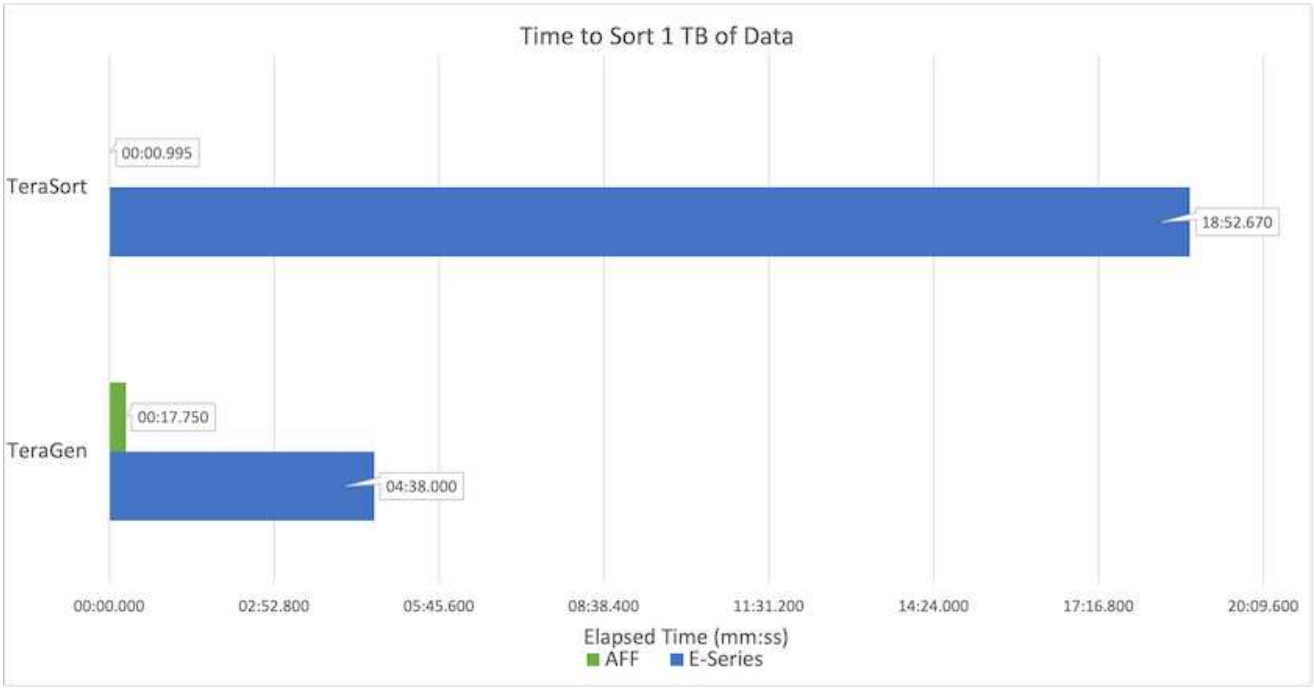
### Tiering dello storage

Con il tiering dello storage Hadoop, è possibile memorizzare file con diversi tipi di storage in conformità con una policy di storage. I tipi di storage includono `hot`, `cold`, `warm`, `all_ssd`, `one_ssd`, e. `lazy_persist`.

<<<<<<< HEAD abbiamo eseguito la convalida del tiering dello storage Hadoop su un controller di storage NetApp AFF e su un controller di storage e-Series con unità SSD e SAS con diverse policy di storage. Il cluster Spark con AFF-A800 ha quattro nodi di lavoro di calcolo, mentre il cluster con e-Series ne ha otto. Questo serve principalmente a confrontare le prestazioni dei dischi a stato solido (SSD) rispetto ai dischi rigidi (HDD).

Abbiamo eseguito la convalida del tiering dello storage Hadoop su un controller di storage NetApp AFF e su un controller di storage e-Series con unità SSD e SAS con policy di storage diverse. Il cluster Spark con AFF-A800 ha quattro nodi di lavoro di calcolo, mentre il cluster con e-Series ne ha otto. Abbiamo fatto questo principalmente per confrontare le performance dei dischi a stato solido con quelle dei dischi rigidi. >>>>>> a51c9dddf73ca69e1120ce05edc7b0b9607b96eae

La figura seguente mostra le performance delle soluzioni NetApp per un SSD Hadoop.




- La configurazione baseline NL-SAS utilizzava otto nodi di calcolo e 96 dischi NL-SAS. Questa configurazione ha generato 1 TB di dati in 4 minuti e 38 secondi. Vedere "TR-3969 soluzione NetApp e-Series per Hadoop" per informazioni dettagliate sulla configurazione del cluster e dello storage.
- Utilizzando TeraGen, la configurazione SSD ha generato 1 TB di dati a una velocità di 15,66 volte superiore rispetto alla configurazione NL-SAS. Inoltre, la configurazione SSD utilizzava la metà del numero di nodi di calcolo e la metà del numero di dischi (24 unità SSD in totale). In base al tempo di completamento del lavoro, la velocità era quasi doppia rispetto alla configurazione NL-SAS.
- Utilizzando TeraSort, la configurazione SSD ha ordinato 1 TB di dati 1138.36 volte più rapidamente della configurazione NL-SAS. Inoltre, la configurazione SSD utilizzava la metà del numero di nodi di calcolo e la metà del numero di dischi (24 unità SSD in totale). Pertanto, per disco, la velocità era circa tre volte superiore rispetto alla configurazione NL-SAS. <<<<<<< TESTA
- La transizione da dischi rotanti a all-flash migliora le performance. Il numero di nodi di calcolo non era il collo di bottiglia. Con lo storage all-flash di NetApp, le performance di runtime sono perfettamente scalabili.
- Con NFS, i dati erano funzionalmente equivalenti a quelli del pool, il che può ridurre il numero di nodi di calcolo in base al carico di lavoro. Gli utenti del cluster Apache Spark non devono ribilanciare manualmente i dati quando cambiano il numero di nodi di calcolo.

- In sintesi, la transizione dai dischi rotanti a all-flash migliora le performance. Il numero di nodi di calcolo non era il collo di bottiglia. Con lo storage all-flash NetApp, le performance di runtime sono perfettamente scalabili.

- Con NFS, i dati erano funzionalmente equivalenti a quelli del pool, il che può ridurre il numero di nodi di calcolo in base al carico di lavoro. Gli utenti del cluster Apache Spark non devono ribilanciare manualmente i dati quando cambiano il numero di nodi di calcolo. >>>>>> a51c9dddf73ca69e1120ce05edc7b0b9607b96eae

#### Scalabilità delle performance - scalabilità orizzontale

Quando è necessaria una maggiore potenza di calcolo da un cluster Hadoop in una soluzione AFF, è possibile aggiungere nodi dati con un numero appropriato di controller storage. NetApp consiglia di iniziare con quattro nodi di dati per array di controller storage e di aumentare il numero fino a otto nodi di dati per controller storage, a seconda delle caratteristiche del carico di lavoro.

AFF e FAS sono perfetti per l'analisi in-place. In base ai requisiti di calcolo, è possibile aggiungere gestori di nodi, mentre le operazioni senza interruzioni consentono di aggiungere un controller di storage on-demand senza downtime. Offriamo funzionalità complete con AFF e FAS, come SUPPORTO multimediale NVME, efficienza garantita, riduzione dei dati, QOS, analisi predittiva, tiering del cloud, replica, implementazione del cloud e sicurezza. Per aiutare i clienti a soddisfare i propri requisiti, NetApp offre funzionalità come analisi del file system, quote e bilanciamento del carico on-box senza costi di licenza aggiuntivi. NetApp offre performance migliori in termini di numero di processi simultanei, latenza inferiore, operazioni più semplici e throughput di gigabyte al secondo superiore rispetto alla concorrenza. Inoltre, NetApp Cloud Volumes ONTAP viene eseguito su tutti e tre i principali cloud provider.

#### Scalabilità delle performance - scalabilità verticale

Le funzionalità di scale-up consentono di aggiungere dischi ai sistemi AFF, FAS ed e-Series quando è necessaria una capacità di storage aggiuntiva. Con Cloud Volumes ONTAP, la scalabilità dello storage a livello di PB è una combinazione di due fattori: Il tiering dei dati utilizzati di rado per lo storage a oggetti dallo storage a blocchi e lo stacking delle licenze Cloud Volumes ONTAP senza elaborazione aggiuntiva.

#### Protocolli multipli

I sistemi NetApp supportano la maggior parte dei protocolli per le implementazioni Hadoop, tra cui SAS, iSCSI, FCP, InfiniBand, E NFS.

#### Soluzioni operative e supportate

Le soluzioni Hadoop descritte in questo documento sono supportate da NetApp. Queste soluzioni sono certificate anche con i principali distributori Hadoop. Per ulteriori informazioni, consultare "MapR" sito, il "Hortonworks" E il Cloudera "certificazione" e. "partner" siti.

# Pubblico di riferimento

## Il mondo dell'analytics e della scienza dei dati tocca diverse discipline nell'IT e nel business:

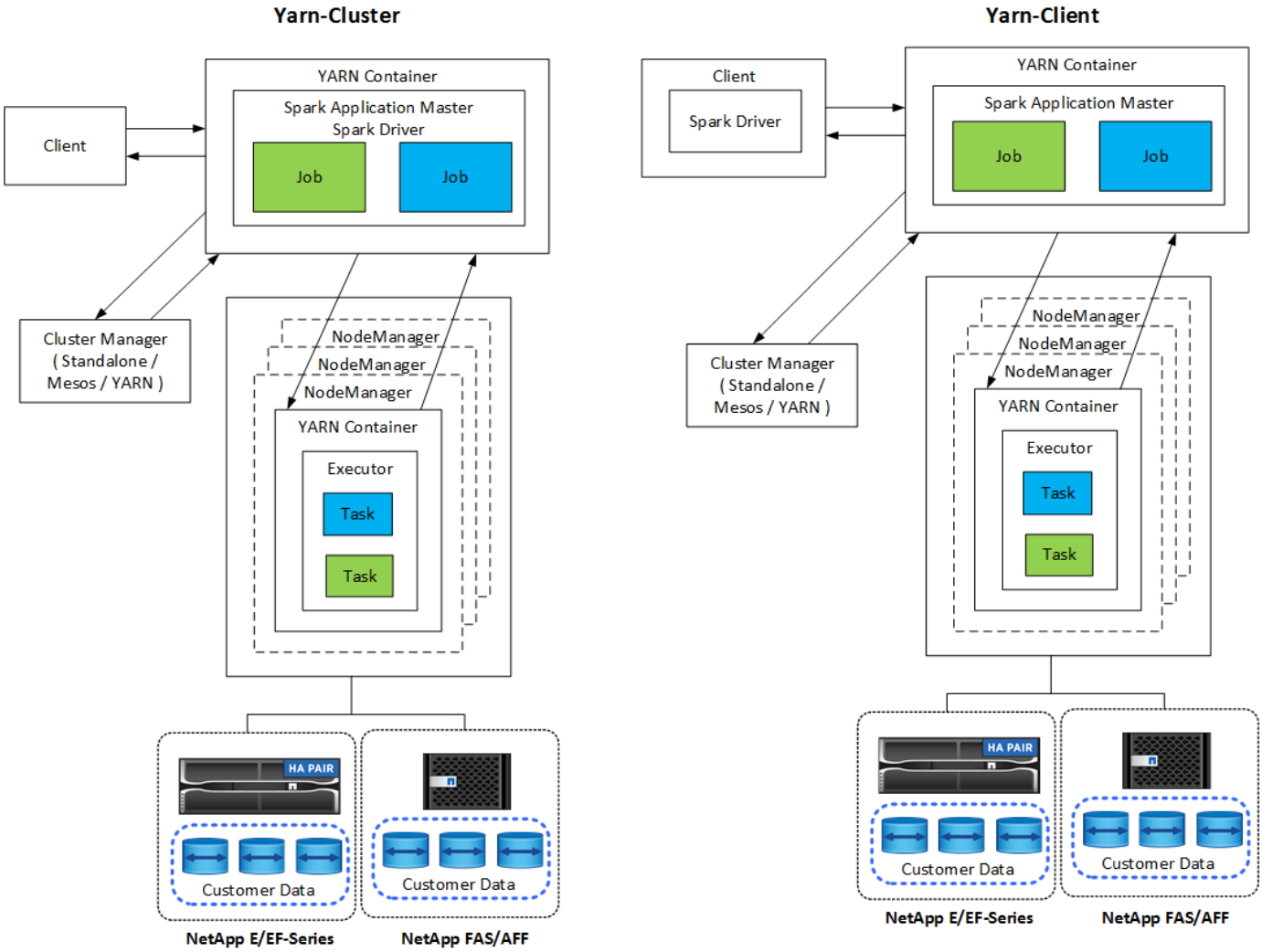
- Il data scientist ha bisogno della flessibilità necessaria per utilizzare i propri strumenti e le librerie preferite.
- Il data engineer deve sapere come i dati scorrono e dove risiedono.
- Un tecnico DevOps ha bisogno dei tool per integrare le nuove applicazioni ai e ML nelle pipeline ci e CD.
- Gli amministratori e gli architetti del cloud devono essere in grado di configurare e gestire le risorse del cloud ibrido.
- Gli utenti aziendali desiderano avere accesso alle applicazioni di analisi, ai, ML e DL.

In questo report tecnico, descriviamo come NetApp AFF, e-Series, StorageGRID, NFS direct access, Apache Spark, Horovod e keras aiutano ciascuno di questi ruoli a portare valore al business.

# Tecnologia della soluzione

Apache Spark è un popolare framework di programmazione per la scrittura di applicazioni Hadoop che funziona direttamente con Hadoop Distributed file System (HDFS). Spark è pronto per la produzione, supporta l'elaborazione dei dati in streaming ed è più veloce di MapReduce. Spark dispone di un caching dei dati in-memory configurabile per un'iterazione efficiente e la shell Spark è interattiva per l'apprendimento e l'esplorazione dei dati. Con Spark, puoi creare applicazioni in Python, Scala o Java. Le applicazioni SPARK sono costituite da uno o più lavori che hanno uno o più compiti.

Ogni applicazione Spark dispone di un driver Spark. In modalità YARN-Client, il driver viene eseguito sul client localmente. In modalità YARN-Cluster, il driver viene eseguito nel cluster sul master dell'applicazione. In modalità cluster, l'applicazione continua a funzionare anche se il client si disconnette.




Esistono tre cluster manager:

- **Standalone.** questo gestore fa parte di Spark, che semplifica la configurazione di un cluster.

- **Apache Mesos.** questo è un gestore di cluster generale che esegue anche MapReduce e altre applicazioni.
- **Hadoop YARN.** questo è un resource manager in Hadoop 3.

Il dataset distribuito resiliente (RDD) è il componente principale di Spark. RDD ricrea i dati persi e mancanti dai dati memorizzati nel cluster e memorizza i dati iniziali provenienti da un file o creati a livello di programmazione. Le RDD vengono create da file, dati in memoria o da un altro RDD. La programmazione SPARK esegue due operazioni: Trasformazione e azioni. La trasformazione crea un nuovo RDD basato su uno esistente. Le azioni restituiscono un valore da un RDD.

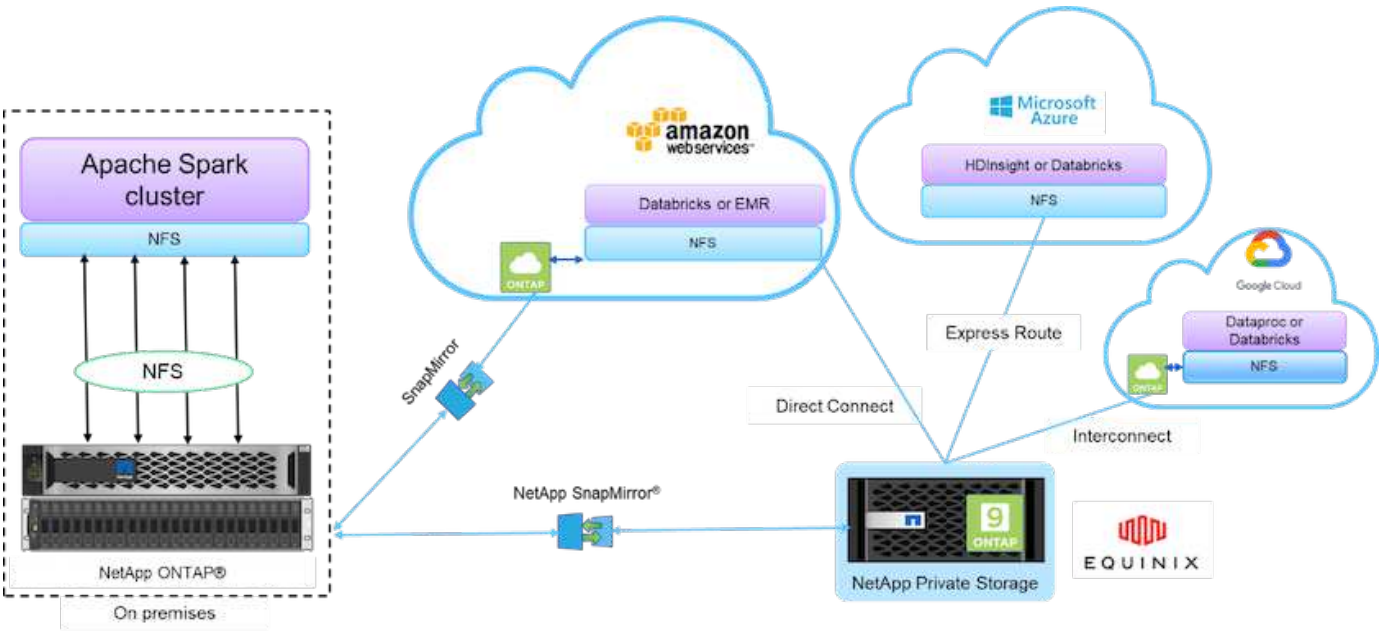
Le trasformazioni e le azioni si applicano anche ai DataSet e ai DataFrame di Spark. Un set di dati è una raccolta distribuita di dati che offre i benefici delle RDDs (tipizzazione forte, utilizzo delle funzioni lambda) con i benefici del motore di esecuzione ottimizzato di Spark SQL. È possibile costruire un dataset da oggetti JVM e manipolarlo utilizzando trasformazioni funzionali (mappa, mappa piatta, filtro e così via). Un DataFrame è un dataset organizzato in colonne denominate. È concettualmente equivalente a una tabella in un database relazionale o a un frame di dati in R/Python. I DataFrame possono essere costruiti da un'ampia gamma di origini come file di dati strutturati, tabelle in Hive/HBase, database esterni on-premise o nel cloud o RDDs esistenti.

Le applicazioni SPARK includono uno o più lavori Spark. I job eseguono task negli esecutori e gli esecutori vengono eseguiti in CONTAINER DI FILATI. Ogni esecutore viene eseguito in un singolo container e gli esecutori esistono per tutta la vita di un'applicazione. Un esecutore viene fissato dopo l'avvio dell'applicazione e IL FILATO non ridimensiona il container già allocato. Un esecutore può eseguire task contemporaneamente sui dati in-memory.

# Panoramica delle soluzioni NetApp Spark

NetApp dispone di tre portfolio di storage: FAS/AFF, e-Series e Cloud Volumes ONTAP. Abbiamo validato AFF e e-Series con il sistema di storage ONTAP per le soluzioni Hadoop con Apache Spark.

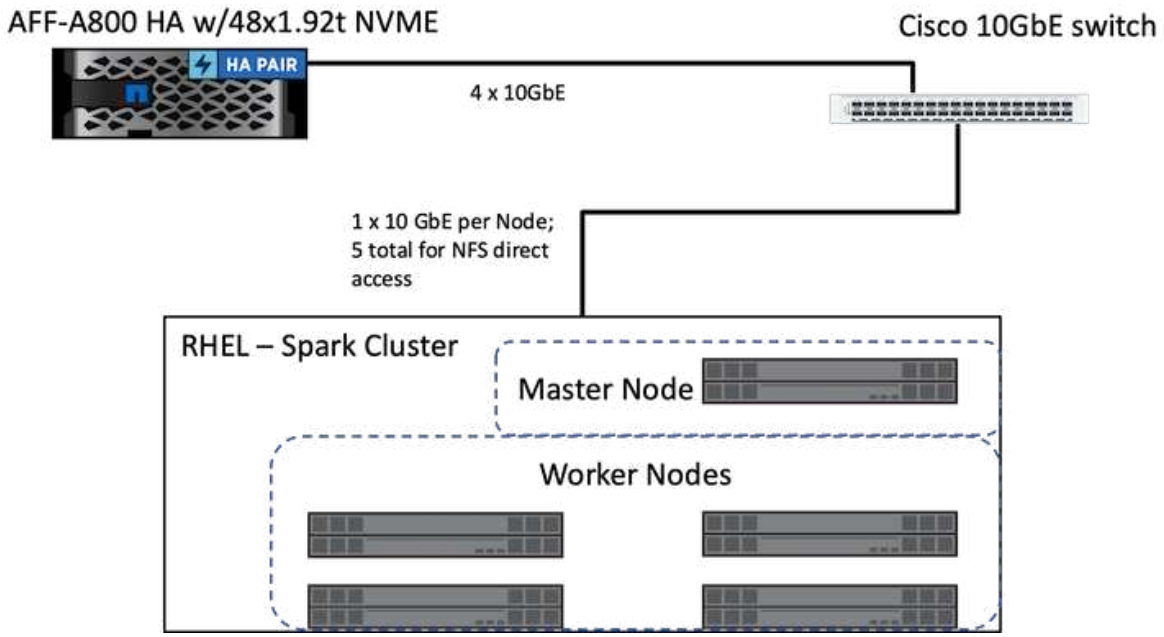
Il data fabric basato su NetApp integra i servizi e le applicazioni di gestione dei dati (building block) per l'accesso, il controllo, la protezione e la sicurezza dei dati, come mostrato nella figura seguente.



Gli elementi di base della figura precedente includono:

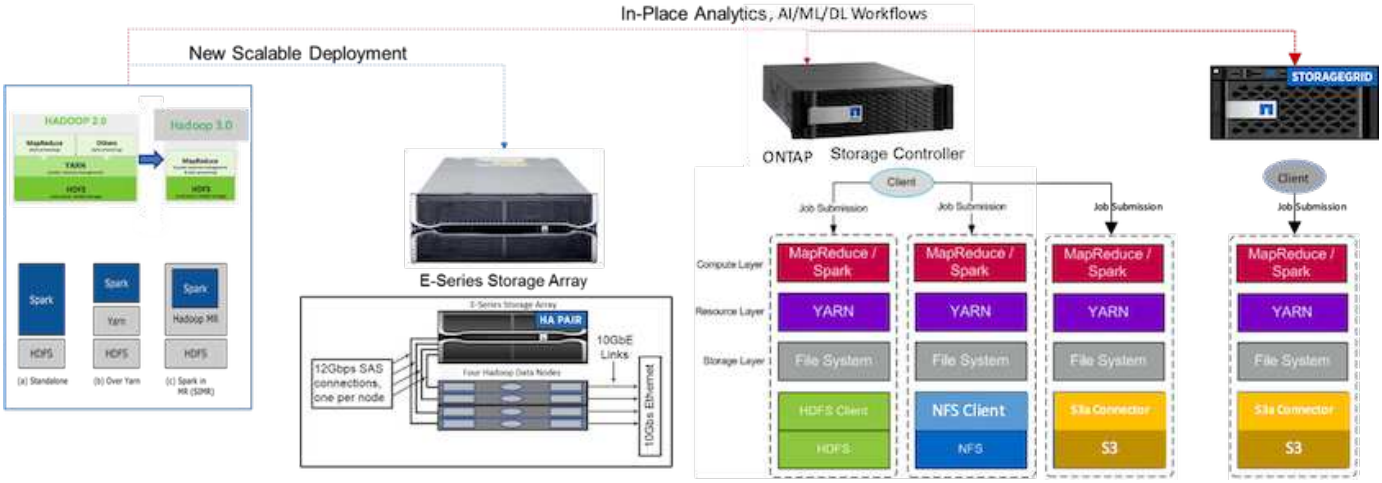
- **Accesso diretto NetApp NFS.** offre i più recenti cluster Hadoop e Spark con accesso diretto ai volumi NetApp NFS senza requisiti aggiuntivi di software o driver.
- **NetApp Cloud Volumes ONTAP e servizi di volume cloud.** storage connesso definito tramite software basato su ONTAP eseguito in AWS (Amazon Web Services) o Azure NetApp Files (ANF) nei servizi cloud Microsoft Azure.
- **La tecnologia NetApp SnapMirror.** offre funzionalità di protezione dei dati tra istanze cloud o NPS ONTAP on-premise e on-premise.
- **Cloud service provider.** questi provider includono AWS, Microsoft Azure, Google Cloud e IBM Cloud.
- **PaaS.** servizi di analisi basati sul cloud come Amazon Elastic MapReduce (EMR) e Databricks in AWS, Microsoft Azure HDInsight e Azure Databricks.

La seguente figura illustra la soluzione Spark con lo storage NetApp.

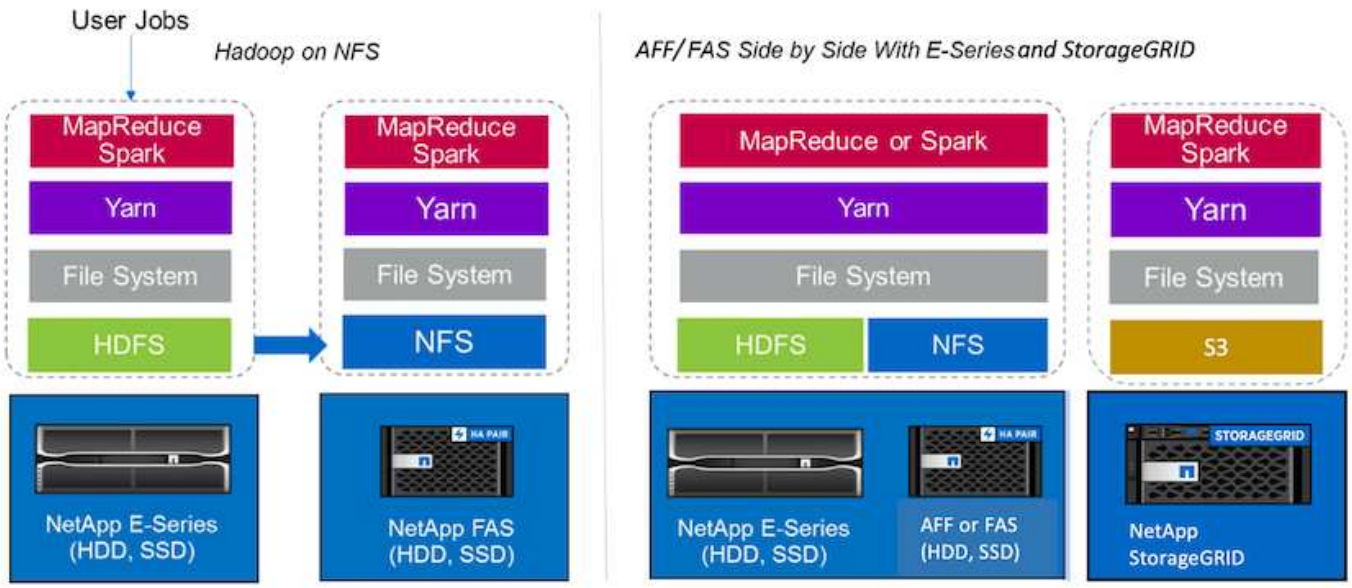



La soluzione Spark di ONTAP utilizza il protocollo di accesso diretto NetApp NFS per l'analisi in-place e i flussi di lavoro ai, ML e DL utilizzando l'accesso ai dati di produzione esistenti. I dati di produzione disponibili per i nodi Hadoop vengono esportati per eseguire lavori analitici in-place e ai, ML e DL. È possibile accedere ai dati da elaborare nei nodi Hadoop con l'accesso diretto NetApp NFS o senza di essi. In Spark con la versione standalone o. `yarn` Cluster manager, è possibile configurare un volume NFS utilizzando `file:///<target_volume`. Abbiamo validato tre casi di utilizzo con set di dati diversi. I dettagli di queste validazioni sono presentati nella sezione "risultati del test". (xref)

La seguente figura illustra il posizionamento dello storage NetApp Apache Spark/Hadoop.




Abbiamo identificato le caratteristiche esclusive della soluzione Spark e-Series, della soluzione Spark AFF/FAS ONTAP e della soluzione Spark StorageGRID, e abbiamo eseguito test e validazione dettagliati. In base alle nostre osservazioni, NetApp consiglia la soluzione e-Series per le installazioni in ambiente e le nuove implementazioni scalabili e la soluzione AFF/FAS per l'analisi in-place, i carichi di lavoro ai, ML e DL utilizzando i dati NFS esistenti e StorageGRID per ai, ML e DL e le analisi dei dati moderne quando è richiesto lo storage a oggetti.




Un data Lake è un repository di storage per set di dati di grandi dimensioni in formato nativo che può essere utilizzato per attività di analisi, ai, ML e DL. Abbiamo creato un repository di data Lake per le soluzioni e-Series, AFF/FAS e StorageGRID SG6060 Spark. Il sistema e-Series fornisce l'accesso HDFS al cluster Hadoop Spark, mentre i dati di produzione esistenti sono accessibili attraverso il protocollo di accesso diretto NFS al cluster Hadoop. Per i set di dati che risiedono nello storage a oggetti, NetApp StorageGRID offre accesso sicuro S3 e S3a.

# Riepilogo del caso d'utilizzo

Questa pagina descrive le diverse aree in cui è possibile utilizzare questa soluzione.

## Dati in streaming

Apache Spark è in grado di elaborare i dati in streaming, utilizzati per processi di estrazione, trasformazione e carico (ETL) in streaming, per l'arricchimento dei dati, per l'attivazione del rilevamento degli eventi e per analisi complesse delle sessioni:

- **Streaming ETL.** i dati vengono continuamente ripuliti e aggregati prima di essere inseriti negli archivi dati. Netflix utilizza lo streaming di Kafka e Spark per creare una soluzione di monitoraggio dei dati e consigli sui film online in tempo reale in grado di elaborare miliardi di eventi al giorno da diverse origini dati. Tuttavia, l'ETL tradizionale per l'elaborazione in batch viene trattato in modo diverso. Questi dati vengono letti per primi, quindi convertiti in un formato di database prima di essere scritti nel database.
- **Arricchimento dei dati.** lo streaming Spark arricchisce i dati live con dati statici per consentire un'analisi dei dati più in tempo reale. Ad esempio, gli inserzionisti online possono fornire annunci personalizzati e mirati, diretti in base alle informazioni sul comportamento dei clienti.
- **Rilevamento eventi trigger.** lo streaming Spark consente di rilevare e rispondere rapidamente a comportamenti anomali che potrebbero indicare problemi potenzialmente gravi. Ad esempio, gli istituti finanziari utilizzano i trigger per rilevare e arrestare le transazioni di frode, mentre gli ospedali utilizzano i trigger per rilevare i cambiamenti sanitari pericolosi rilevati nei segni vitali di un paziente.
- **Analisi complessa della sessione.** lo streaming di Spark raccoglie eventi come l'attività dell'utente dopo l'accesso a un sito Web o a un'applicazione, che vengono quindi raggruppati e analizzati. Ad esempio, Netflix utilizza questa funzionalità per fornire consigli sui filmati in tempo reale.

Per ulteriori informazioni su configurazione dei dati in streaming, verifica di Confluent Kafka e test delle performance, consulta "TR-4912: Linee guida sulle Best practice per lo storage a più livelli Confluent Kafka con NetApp".

## Apprendimento automatico

Il framework integrato Spark consente di eseguire query ripetute sui set di dati utilizzando la libreria di apprendimento automatico (MLlib). MLlib viene utilizzato in aree come clustering, classificazione e riduzione delle dimensioni per alcune funzioni comuni dei big data, come l'intelligence predittiva, la segmentazione dei clienti per scopi di marketing e l'analisi del sentimento. MLlib viene utilizzato nella sicurezza di rete per eseguire ispezioni in tempo reale dei pacchetti di dati per indicazioni di attività dannose. Aiuta i provider di sicurezza a conoscere le nuove minacce e a restare al passo con gli hacker, proteggendo i propri clienti in tempo reale.

## Apprendimento approfondito

TensorFlow è un framework di deep learning diffuso in tutto il settore. TensorFlow supporta la formazione distribuita su un cluster di CPU o GPU. Questo training distribuito consente agli utenti di eseguirlo su una grande quantità di dati con molti livelli profondi.

Fino a poco tempo fa, se volevamo utilizzare TensorFlow con Apache Spark, dovevamo eseguire tutte le ETL necessarie per TensorFlow in PySpark e quindi scrivere i dati nello storage intermedio. Tali dati verranno quindi caricati nel cluster TensorFlow per l'effettivo processo di training. Questo flusso di lavoro richiedeva all'utente di mantenere due diversi cluster, uno per ETL e uno per la formazione distribuita di TensorFlow. L'esecuzione e la manutenzione di più cluster erano in genere noiose e dispendiose in termini di tempo.

DataFrame e RDD nelle versioni precedenti di Spark non erano adatti per l'apprendimento approfondito perché l'accesso casuale era limitato. In Spark 3.0 con il progetto Hydrogen, è stato aggiunto il supporto nativo per i framework di deep learning. Questo approccio consente la pianificazione non basata su MapReduce sul cluster Spark.

## Analisi interattiva

Apache Spark è abbastanza veloce da eseguire query esplorative senza campionamenti con linguaggi di sviluppo diversi da Spark, tra cui SQL, R e Python. SPARK utilizza strumenti di visualizzazione per elaborare dati complessi e visualizzarli in modo interattivo. Spark with Structured streaming esegue query interattive in base ai dati in tempo reale in analisi web che consentono di eseguire query interattive in base alla sessione corrente di un visitatore web.

## Sistema consigliato

Nel corso degli anni, i sistemi di recommender hanno apportato enormi cambiamenti alla nostra vita, in quanto aziende e consumatori hanno risposto a cambiamenti drastici nello shopping online, nell'intrattenimento online e in molti altri settori. In effetti, questi sistemi sono tra i casi di successo più evidenti dell'ai in produzione. In molti casi pratici di utilizzo, i sistemi consigliati sono combinati con i chatbot o ai conversazionali interfacciati con un backend NLP per ottenere informazioni rilevanti e produrre utili inferenze.

Oggi, molti retailer stanno adottando modelli di business più recenti, come l'acquisto online e il ritiro in negozio, il ritiro in marciapiede, il pagamento automatico, la scansione e la partenza e molto altro ancora. Questi modelli sono diventati preminenti durante la pandemia di COVID-19, rendendo gli acquisti più sicuri e convenienti per i consumatori. L'ai è fondamentale per queste tendenze digitali in crescita, che sono influenzate dal comportamento dei consumatori e viceversa. Per soddisfare le crescenti esigenze dei consumatori, aumentare l'esperienza del cliente, migliorare l'efficienza operativa e aumentare i ricavi, NetApp aiuta i clienti aziendali e le aziende a utilizzare algoritmi di apprendimento automatico e di deep learning per progettare sistemi di raccomandazione più rapidi e precisi.

Esistono diverse tecniche utilizzate per fornire consigli, tra cui il filtraggio collaborativo, i sistemi basati sui contenuti, il modello DLRM (Deep Learning recommender Model) e le tecniche ibride. In precedenza, i clienti utilizzavano PySpark per implementare il filtraggio collaborativo per la creazione di sistemi di raccomandazione. Spark MLlib implementa Alternating Least Squares (ALS) per il filtraggio collaborativo, un algoritmo molto popolare tra le aziende prima dell'ascesa di DLRM.

## Elaborazione del linguaggio naturale

L'intelligenza artificiale conversa, resa possibile dall'elaborazione del linguaggio naturale (NLP), è il ramo dell'intelligenza artificiale che aiuta i computer a comunicare con gli esseri umani. La tecnologia NLP è prevalente in tutti i mercati verticali del settore e in molti casi di utilizzo, dagli smart Assistant ai chatbot, alla ricerca con Google e al testo predittivo. Secondo a. "Gartner" Secondo le previsioni, entro il 2022, il 70% delle persone interagirà quotidianamente con le piattaforme di ai convergenti. Per una conversazione di alta qualità tra un essere umano e una macchina, le risposte devono essere rapide, intelligenti e naturali.

I clienti hanno bisogno di una grande quantità di dati per elaborare e formare i propri modelli NLP e ASR (Automatic Speech Recognition). Devono anche spostare i dati all'edge, al core e al cloud e hanno bisogno della potenza necessaria per eseguire l'inferenza in millisecondi per stabilire una comunicazione naturale con gli esseri umani. NetApp ai e Apache Spark sono la combinazione ideale per calcolo, storage, elaborazione dei dati, training sui modelli, messa a punto, e implementazione.

L'analisi del sentimento è un campo di studio all'interno di NLP in cui i sentimenti positivi, negativi o neutri vengono estratti dal testo. L'analisi del sentimento ha una varietà di casi di utilizzo, dalla determinazione delle performance dei dipendenti del centro di supporto nelle conversazioni con i chiamanti alla fornitura di risposte dei chatbot automatizzate appropriate. Inoltre, è stato utilizzato per prevedere il prezzo delle azioni di un'azienda in base alle interazioni tra i rappresentanti dell'azienda e il pubblico durante le chiamate trimestrali sui guadagni. Inoltre, l'analisi del sentimento può essere utilizzata per determinare la posizione del cliente sui prodotti, servizi o supporto forniti dal marchio.

Abbiamo utilizzato "NLP. Scintilla" libreria da "John Snow Labs" Per caricare pipeline preaddestrate e rappresentazioni di encoder bidirezionali da modelli di Transformers (BERT), tra cui "sentimento di notizie finanziarie" e. "FinBERT", esecuzione di togenizzazione, riconoscimento di entità denominate, training sui modelli, analisi di adattamento e sentimento su larga scala. Spark NLP è l'unica libreria NLP open-source in produzione che offre trasformatori all'avanguardia come BERT, ALBERT, ELECTRA, XLNet, DistillBERT, Roberta, DeBERTA, XLM-Roberta, Longformer, ELMO, Universal sentence Encoder, Google T5, MarianMT e GPT2. La libreria funziona non solo in Python e R, ma anche nell'ecosistema JVM (Java, Scala e Kotlin) su larga scala estendendo Apache Spark in modo nativo.

# Principali casi di utilizzo e architetture ai, ML e DL

I principali casi di utilizzo e la metodologia di ai, ML e DL possono essere suddivisi nelle seguenti sezioni:

## Pipeline SPARK NLP e deduzione distribuita TensorFlow

Il seguente elenco contiene le librerie NLP open-source più diffuse che sono state adottate dalla community di data science con diversi livelli di sviluppo:

- "Natural Language Toolkit (NLTK)". Il toolkit completo per tutte le tecniche NLP. È stato mantenuto fin dai primi anni 2000.
- "TextBlob". Un tool NLP facile da usare API Python costruita su NLTK e Pattern.
- "Stanford Core NLP". Servizi e pacchetti NLP in Java sviluppati da Stanford NLP Group.
- "GENsim". Topic Modeling for Humans è iniziato come una raccolta di script Python per il progetto Czech Digital Mathematics Library.
- "Spacy". Workflow NLP industriali end-to-end con Python e Cython con accelerazione GPU per i trasformatori.
- "Fasttext". Una libreria NLP gratuita, leggera e open-source per l'apprendimento delle parole e la classificazione delle frasi creata dal laboratorio ai Research (FAIR) di Facebook.

Spark NLP è una soluzione singola e unificata per tutte le attività e i requisiti NLP che consente di ottenere software scalabile, dalle performance elevate e ad alta precisione basato su NLP per casi di utilizzo in produzione reali. Sfrutta l'apprendimento del trasferimento e implementa gli algoritmi e i modelli più recenti nella ricerca e nei vari settori. A causa della mancanza di supporto completo da parte di Spark per le librerie di cui sopra, Spark NLP è stato costruito su "SPARK ML" Sfruttare il motore di elaborazione dei dati distribuiti in-memory di Spark per scopi generali come libreria NLP di livello Enterprise per flussi di lavoro di produzione mission-critical. I suoi annotatori utilizzano algoritmi basati su regole, machine learning e TensorFlow per potenziare le implementazioni di deep learning. Questo copre i comuni compiti di NLP, inclusi, a titolo esemplificativo ma non esaustivo, la tokenizzazione, la lemmatizzazione, lo stemming, il tagging part-of-speech, il riconoscimento di entità nominate, controllo ortografico e analisi del sentimento.

Le rappresentazioni di encoder bidirezionali di Transformers (BERT) sono tecniche di apprendimento automatico basate su trasformatore per NLP. Ha reso popolare il concetto di pre-training e fine tuning. L'architettura dei trasformatori di BERT è nata dalla traduzione automatica, che modella le dipendenze a lungo termine meglio dei modelli di linguaggio basati su Neural Network (RNN) ricorrenti. Ha inoltre introdotto il task Masked Language Modeling (MLM), in cui il 15% casuale di tutti i token viene mascherato e il modello li prevede, consentendo una reale bidirezionalità.

L'analisi del sentimento finanziario è complessa a causa del linguaggio specializzato e della mancanza di dati etichettati in quel dominio. "FinBERT", Un modello linguistico basato su BERT preaddestrato, è stato adattato al dominio "Reuters TRC2", un corpus finanziario, e messo a punto con i dati etichettati ( "Financial

PhraseBank") per la classificazione del sentimento finanziario. I ricercatori hanno estratto 4, 500 frasi dagli articoli di notizie con termini finanziari. Poi 16 esperti e master studenti con background finanziari hanno etichettato le frasi come positive, neutrali e negative. Abbiamo creato un workflow Spark end-to-end per analizzare il sentimento per le trascrizioni delle chiamate delle aziende Top-10 NASDAQ dal 2016 al 2020 utilizzando FinBERT e due altre pipeline preformate ( "Analisi del sentimento per le notizie finanziarie", "Spiegare il documento DL") Di Spark NLP.

Il motore di deep learning sottostante per Spark NLP è TensorFlow, una piattaforma open source end-to-end per l'apprendimento automatico che consente la creazione di modelli semplici, una produzione ML solida ovunque e potenti sperimentazioni per la ricerca. Pertanto, quando si eseguono le nostre pipeline in Spark `yarn cluster` In pratica, abbiamo eseguito TensorFlow distribuito con parallelizzazione di dati e modelli tra un nodo master e più nodi di lavoro, oltre allo storage collegato alla rete montato sul cluster.

## Formazione distribuita Horovod

La convalida Hadoop principale per le performance correlate a MapReduce viene eseguita con TeraGen, TeraSort, TeraValidate e DFSIO (lettura e scrittura). I risultati della convalida TeraGen e TeraSort sono presentati nella "TR-3969: Soluzioni NetApp per Hadoop" Per e-Series e nella sezione "Tiering dello storage" (xref) per AFF.

In base alle richieste dei clienti, riteniamo che la formazione distribuita con Spark sia uno dei casi di utilizzo più importanti. In questo documento, abbiamo utilizzato "Hovorod su Spark" Per convalidare le performance di Spark con le soluzioni di cloud ibrido, nativo e on-premise di NetApp utilizzando i controller di storage NetApp All Flash FAS (AFF), Azure NetApp Files e StorageGRID.

Il pacchetto Horovod on Spark offre un comodo wrapper intorno a Horovod che semplifica l'esecuzione di workload di training distribuiti nei cluster Spark, consentendo un loop di progettazione di modelli ristretti in cui l'elaborazione dei dati, la formazione sui modelli e la valutazione dei modelli vengono eseguite in Spark, dove risiedono i dati di formazione e deduzione.

Esistono due API per l'esecuzione di Horovod su Spark: Un'API di stima di alto livello e un'API di esecuzione di livello inferiore. Sebbene entrambi utilizzino lo stesso meccanismo sottostante per lanciare Horovod sugli esecutori di Spark, l'API Estimator astratta l'elaborazione dei dati, il loop di training del modello, il checkpoint del modello, la raccolta di metriche e il training distribuito. Abbiamo utilizzato gli stimatori di Horovod Spark, TensorFlow e keras per una preparazione dei dati end-to-end e un workflow di training distribuito basato su "Vendita al negozio Kaggle Rossmann" concorrenza.

Lo script `keras_spark_horovod_rossmann_estimator.py` si trova nella sezione "Script Python per ogni caso di utilizzo principale." Contiene tre parti:

- La prima parte esegue varie fasi di pre-elaborazione dei dati su un set iniziale di file CSV forniti da Kaggle e raccolti dalla community. I dati di input vengono separati in un set di training con un `Validation` e un dataset di test.
- La seconda parte definisce un modello di rete neurale profonda (DNN) con funzione di attivazione sigmoid logaritmica e un ottimizzatore Adam, ed esegue un training distribuito del modello utilizzando Horovod su Spark.
- La terza parte esegue la previsione sul set di dati di test utilizzando il modello migliore che riduce al minimo l'errore medio assoluto complessivo del set di convalida. Viene quindi creato un file CSV di output.

Vedere la sezione ""Apprendimento automatico"" per diversi risultati di confronto tra runtime.

## Deep learning multi-worker con keras per la previsione CTR

Con i recenti progressi nelle piattaforme E nelle applicazioni ML, è ora molto importante concentrarsi sull'apprendimento su larga scala. Il tasso di click-through (CTR) è definito come il numero medio di click-through per cento impressioni di annunci online (espresso in percentuale). È ampiamente adottato come parametro chiave in diversi mercati verticali e casi di utilizzo del settore, tra cui digital marketing, retail, e-commerce e service provider. Consulta la nostra "TR-4904: Formazione distribuita in Azure - previsione dei tassi click-through" Per ulteriori dettagli sulle applicazioni di CTR e un'implementazione del workflow ai cloud end-to-end con Kubernetes, ETL di dati distribuiti e training sui modelli con DAK e CUDA ML.

In questo report tecnico abbiamo utilizzato una variante di "Set di dati Click Logs Criteo Terabyte" (Vedere TR-4904) per l'apprendimento approfondito distribuito multi-worker che utilizza keras per creare un workflow Spark con modelli DCN (Deep and Cross Network), confrontando le sue performance in termini di funzione di errore di perdita di log con un modello di riferimento Spark ML Logistic Regression. DCN acquisisce in modo efficiente le interazioni efficaci delle funzioni di gradi limitati, apprende interazioni altamente non lineari, non richiede alcuna progettazione manuale delle funzioni o ricerca completa e ha un costo di calcolo basso.

I dati per i sistemi recommender su scala web sono per lo più discreti e categorici, il che porta a un ampio e sparso spazio di funzionalità che è difficile per l'esplorazione delle funzionalità. Questo ha limitato la maggior parte dei sistemi su larga scala a modelli lineari come la regressione logistica. Tuttavia, l'identificazione delle funzionalità spesso predittive e allo stesso tempo l'esplorazione di funzioni incrociate rare o invisibili è la chiave per fare buone previsioni. I modelli lineari sono semplici, interpretabili e facili da scalare, ma sono limitati nel loro potere espressivo.

Le funzionalità incrociate, d'altro canto, si sono dimostrate significative nel migliorare l'espressività dei modelli. Sfortunatamente, spesso richiede un'ingegneria delle funzionalità manuale o una ricerca completa per identificare tali funzionalità. La generalizzazione di interazioni di funzionalità non visibili è spesso difficile. L'utilizzo di una rete interneurale come DCN evita l'ingegneria delle funzionalità specifiche dell'attività, applicando esplicitamente il passaggio delle funzionalità in modo automatico. La rete incrociata è costituita da più livelli, in cui il più alto grado di interazione è determinato in modo probabile dalla profondità del livello. Ogni livello produce interazioni di ordine superiore in base a quelle esistenti e mantiene le interazioni dai livelli precedenti.

Una rete neurale profonda (DNN) ha la promessa di acquisire interazioni molto complesse tra le varie funzionalità. Tuttavia, rispetto alla rete DCN, richiede quasi un ordine di grandezza più parametri, non è in grado di formare funzioni incrociate in modo esplicito e potrebbe non riuscire ad apprendere in modo efficiente alcuni tipi di interazioni tra funzionalità. La rete è efficiente in termini di memoria e facile da implementare. La formazione congiunta dei componenti Cross e DNN acquisisce in modo efficiente le interazioni predittive delle funzionalità e offre performance all'avanguardia sul set di dati CTR Criteo.

Un modello DCN inizia con un livello di incorporamento e stacking, seguito da una rete incrociata e una rete profonda in parallelo. Questi a loro volta sono seguiti da un livello di combinazione finale che combina le uscite dalle due reti. I dati di input possono essere un vettore con funzioni sparse e dense. In Spark, entrambi "ml" e. "mllib" le librerie contengono il tipo `SparseVector`. È quindi importante che gli utenti distinguano i due e si ricordino quando chiamano le rispettive funzioni e metodi. Nei sistemi recommender su scala web come la previsione CTR, gli input sono per lo più caratteristiche categoriche, ad esempio `'country=usa'`. Tali caratteristiche sono spesso codificate come vettori one-hot, ad esempio, `'[0,1,0, …]'`. One-hot-encoding (OHE) con `SparseVector` è utile quando si gestiscono set di dati reali con vocabolari in continua evoluzione e in crescita. Abbiamo modificato gli esempi in "DeepCTR" Elaborare vocabolari di grandi dimensioni, creando vettori di incorporamento nel livello di incorporamento e stacking della nostra rete DCN.

Il "Dataset Criteo Display Ads" prevede il tasso di click-through degli annunci. Dispone di 13 caratteristiche intere e 26 caratteristiche categoriche in cui ogni categoria ha un'elevata cardinalità. Per questo set di dati, un miglioramento di 0.001 nella perdita di log è praticamente significativo a causa delle grandi dimensioni dell'input. Un piccolo miglioramento della precisione di previsione per una base di utenti di grandi dimensioni

può potenzialmente portare a un aumento significativo dei ricavi di un'azienda. Il set di dati contiene 11 GB di log utente da un periodo di 7 giorni, che equivale a circa 41 milioni di record. Abbiamo utilizzato Spark `dataFrame.randomSplit()function` suddividere casualmente i dati per il training (80%), la convalida incrociata (10%) e il restante 10% per il test.

DCN è stato implementato su TensorFlow con keras. L'implementazione del processo di training del modello con DCN comprende quattro componenti principali:

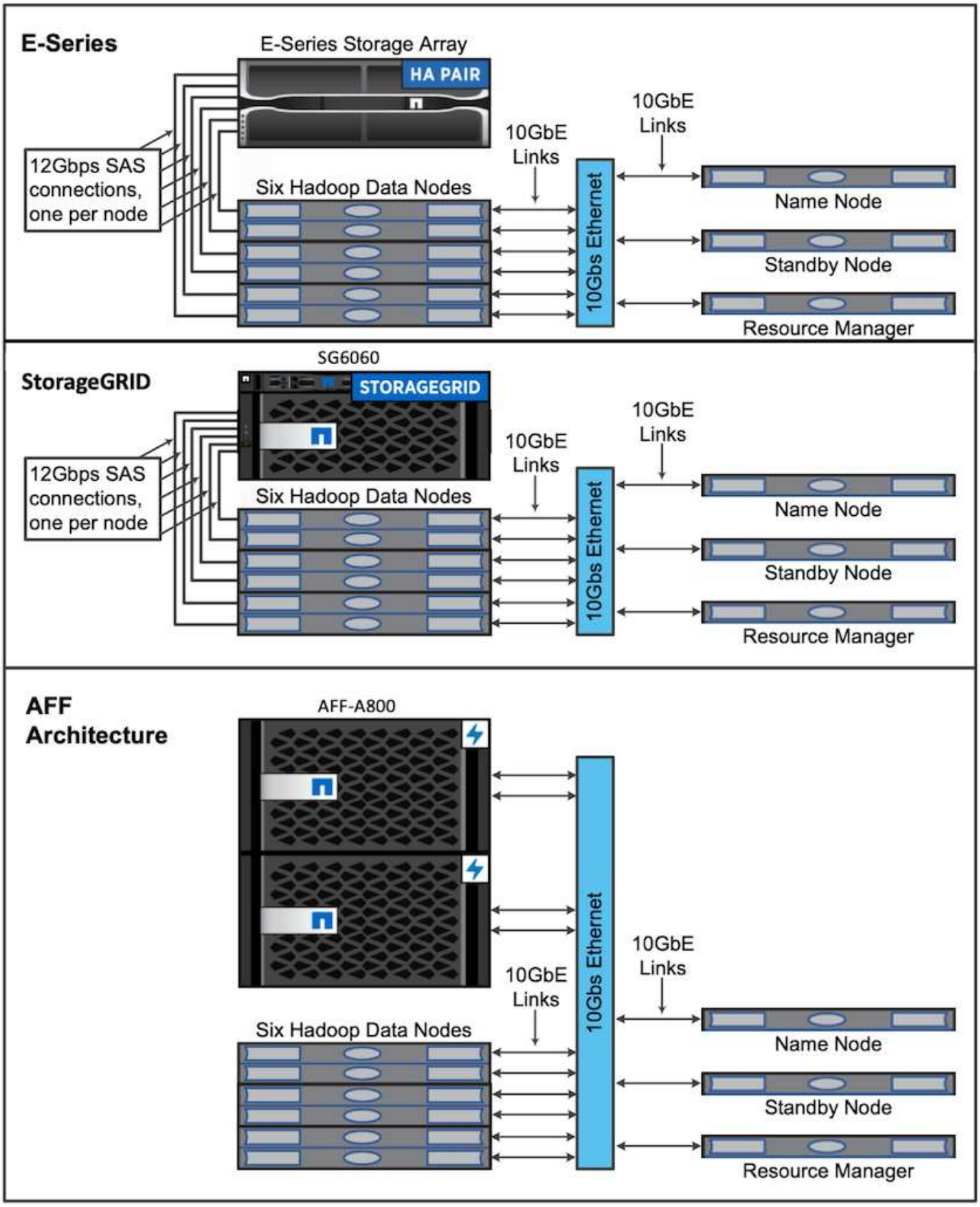
- **Elaborazione e incorporamento dei dati.** le funzionalità a valore reale vengono normalizzate applicando una trasformazione del log. Per le funzionalità categoriche, le funzionalità sono incorporate in vettori densi di dimensione 6×(categoria cardinalità)1/4. Concatenando tutte le incorporazioni si ottiene un vettore di dimensione 1026.
- **Optimization.** abbiamo applicato l'ottimizzazione stocastica mini-batch con Adam Optimizer. La dimensione del batch è stata impostata su 512. La normalizzazione batch è stata applicata alla rete profonda e la norma del gradiente clip è stata impostata su 100.
- **Regolarizzazione.** abbiamo utilizzato la sospensione anticipata, in quanto la regolarizzazione L2 o il dropout non sono stati trovati efficaci.
- **Hyperparameters.** i risultati vengono riportati in base a una ricerca in griglia sul numero di livelli nascosti, la dimensione del livello nascosto, la velocità di apprendimento iniziale e il numero di livelli incrociati. Il numero di livelli nascosti variava da 2 a 5, con dimensioni dei livelli nascosti comprese tra 32 e 1024. Per DCN, il numero di strati incrociati era da 1 a 6. Il tasso di apprendimento iniziale è stato ottimizzato da 0.0001 a 0.001 con incrementi di 0.0001. Tutti gli esperimenti hanno subito interrotto la fase di training 150,000, oltre la quale ha iniziato a verificarsi un overfitting.

Oltre a DCN, abbiamo anche testato altri modelli di deep-learning molto diffusi per la previsione CTR, tra cui "DeepFM", "XDeepFM", "Int. Auto", e. "DCN v2".

## Architetture utilizzate per la convalida

Per questa convalida, abbiamo utilizzato quattro nodi di lavoro e un nodo master con una coppia ha AFF-A800. Tutti i membri del cluster erano connessi tramite switch di rete 10 GbE.

Per la convalida della soluzione NetApp Spark, abbiamo utilizzato tre diversi controller di storage: E5760, E5724 e AFF-A800. I controller di storage e-Series erano collegati a cinque nodi dati con connessioni SAS a 12 Gbps. Il controller di storage AFF ha-Pair offre volumi NFS esportati attraverso connessioni 10 GbE ai nodi di lavoro Hadoop. I membri del cluster Hadoop erano connessi tramite connessioni 10GbE nelle soluzioni e-Series, AFF e StorageGRID Hadoop.




## Risultati del test

Abbiamo utilizzato gli script TeraSort e TeraValidate nello strumento di benchmarking

TeraGen per misurare la convalida delle performance Spark con le configurazioni E5760, E5724 e AFF-A800. Inoltre, sono stati testati tre casi di utilizzo principali: Pipeline SPARK NLP e training distribuito TensorFlow, training distribuito Horovod e deep learning multi-worker con keras per la previsione CTR con DeepFM.

Per la convalida di e-Series e StorageGRID, abbiamo utilizzato il fattore di replica Hadoop 2. Per la convalida AFF, abbiamo utilizzato una sola fonte di dati.

La seguente tabella elenca la configurazione hardware per la convalida delle prestazioni di Spark.

| Tipo | Nodi di lavoro Hadoop | Tipo di disco | Dischi per nodo | Controller dello storage |
|---|---|---|---|---|
| SG6060 | 4 | SAS | 12 | Singola coppia ad alta disponibilità (ha) |
| E5760 | 4 | SAS | 60 | Coppia ha singola |
| E5724 | 4 | SAS | 24 | Coppia ha singola |
| AFF800 | 4 | SSD | 6 | Coppia ha singola |

La seguente tabella elenca i requisiti software.

| Software | Versione |
|---|---|
| RHEL | 7.9 |
| Ambiente di runtime OpenJDK | 1.8.0 |
| Server VM OpenJDK a 64 bit | 25.302 |
| Git | 2.24.1 |
| GCC/G++ | 11.2.1 |
| Scintilla | 3.2.1 |
| PySpark | 3.1.2 |
| SparkNLP | 3.4.2 |
| TensorFlow | 2.9.0 |
| Ere | 2.9.0 |
| Horovod | 0.24.3 |

## Analisi del sentimento finanziario

Abbiamo pubblicato "TR-4910: Analisi del sentimento da Customer Communications con NetApp ai", In cui è stata costruita una pipeline di ai conversazionale end-to-end utilizzando "NetApp DataOps Toolkit", Storage AFF e sistema NVIDIA DGX. La pipeline esegue l'elaborazione del segnale audio batch, il riconoscimento vocale automatico (ASR), l'apprendimento del trasferimento e l'analisi del sentimento utilizzando il DataOps Toolkit, "SDK NVIDIA Riva"e il "Framework di Tao". Espandendo il caso d'uso dell'analisi del sentimento nel settore dei servizi finanziari, abbiamo creato un workflow SparkNLP, caricato tre modelli BERT per varie attività NLP, come il riconoscimento delle entità nominate, e ottenuto un sentimento a livello di frase per le chiamate trimestrali sui guadagni delle prime 10 aziende NASDAQ.

Il seguente script `sentiment_analysis_spark. py` Utilizza il modello FinBERT per elaborare le trascrizioni in HDFS e produrre conteggi di sentimenti positivi, neutri e negativi, come mostrato nella seguente tabella:

```
-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
```

La seguente tabella elenca l'analisi del sentimento a livello di frase e di chiamata degli utili per le prime 10 aziende NASDAQ dal 2016 al 2020.

| I conteggi dei sentimenti e la percentuale | Tutte le 10 aziende | AAAPL | AMD | N. AMZN | CSCO | GOOGL | INTC | MSFT | NVDA |
|---|---|---|---|---|---|---|---|---|---|
| Conteggi positivi | 7447 | 1567 | 743 | 290 | 682 | 826 | 824 | 904 | 417 |
| Conteggi neutrali | 64067 | 6856 | 7596 | 5086 | 6650 | 5914 | 6099 | 5715 | 6189 |
| Conteggi negativi | 1787 | 253 | 213 | 84 | 189 | 97 | 282 | 202 | 89 |
| Conteggi senza categoria | 196 | 0 | 0 | 76 | 0 | 0 | 0 | 1 | 0 |
| (conteggi totali) | 73497 | 8676 | 8552 | 5536 | 7521 | 6837 | 7205 | 6822 | 6695 |

In termini di percentuali, la maggior parte delle frasi pronunciate dagli amministratori delegati e dai CFO è fattuale e quindi ha un sentimento neutrale. Durante una chiamata sui guadagni, gli analisti pongono domande che potrebbero trasmettere un sentimento positivo o negativo. Vale la pena di analizzare in maniera quantitativa il modo in cui il sentimento negativo o positivo influisce sui prezzi delle azioni nello stesso giorno o nel giorno successivo di negoziazione.

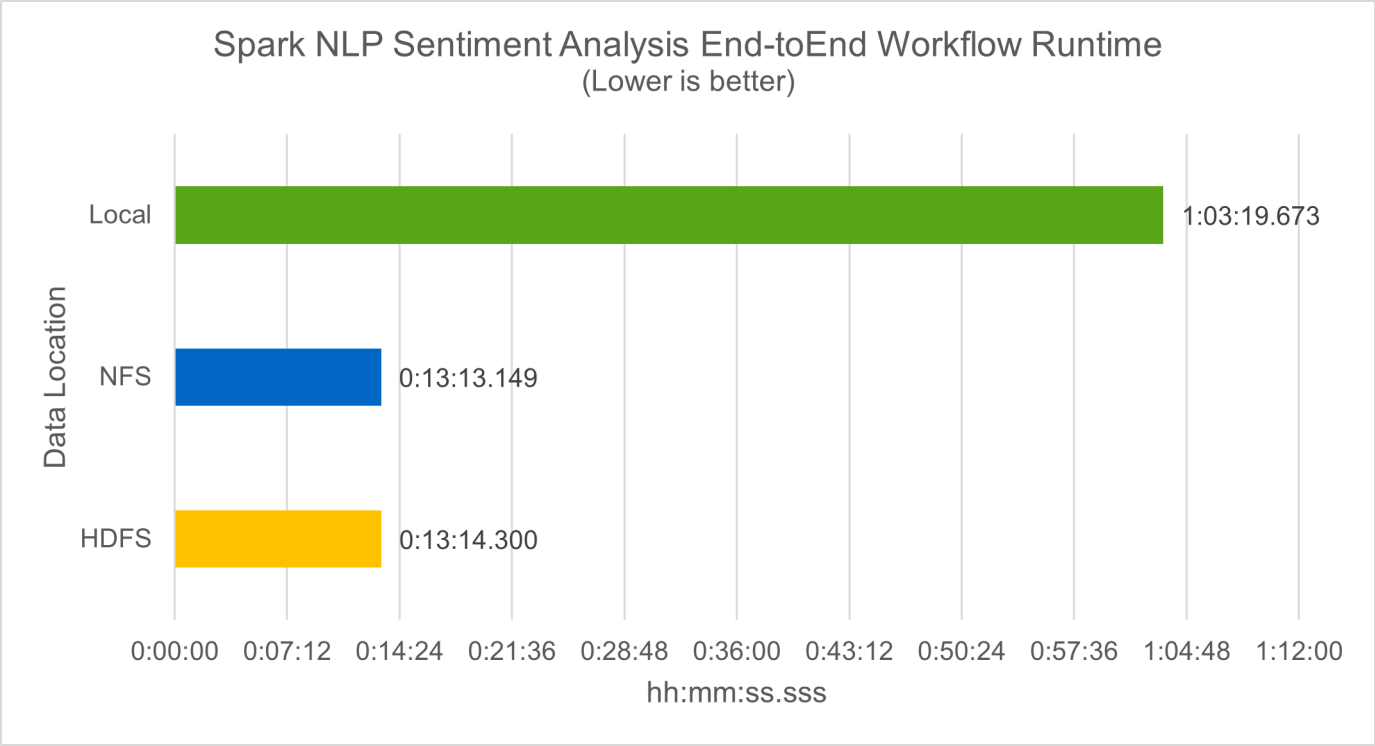
La seguente tabella elenca l'analisi del sentimento a livello di frase per le prime 10 aziende NASDAQ, espressa in percentuale.

| Percentuale di sentimento | Tutte le 10 aziende | AAAPL | AMD | N. AMZN | CSCO | GOOGL | INTC | MSFT | NVDA |
|---|---|---|---|---|---|---|---|---|---|
| Positivo | 10.13% | 18.06% | 8.69% | 5.24% | 9.07% | 12.08% | 11.44% | 13.25% | 6.23% |
| Neutro | 87.17% | 79.02% | 88.82% | 91.87% | 88.42% | 86.50% | 84.65% | 83.77% | 92.44% |
| Negativo | 2.43% | 2.92% | 2.49% | 1.52% | 2.51% | 1.42% | 3.91% | 2.96% | 1.33% |
| Senza categoria | 0.27% | 0% | 0% | 1.37% | 0% | 0% | 0% | 0.01% | 0% |

In termini di runtime del workflow, abbiamo riscontrato un significativo miglioramento di 4,78 volte `local` A un ambiente distribuito in HDFS e un ulteriore miglioramento del 0.14% grazie all'utilizzo di 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
```

Come mostrato nella figura seguente, il parallelismo dei dati e dei modelli ha migliorato l'elaborazione dei dati e la velocità di deduzione del modello TensorFlow distribuito. La posizione dei dati in NFS ha prodotto un runtime leggermente migliore perché il collo di bottiglia del workflow è il download di modelli preformati. Se aumentiamo le dimensioni del set di dati delle trascrizioni, il vantaggio di NFS è più evidente.




## Formazione distribuita con performance Horovod

Il seguente comando ha prodotto informazioni di runtime e un file di log nel cluster Spark utilizzando un singolo `master` nodo con 160 esecutori ciascuno con un core. La memoria dell'esecutore era limitata a 5 GB per evitare errori di memoria esaurita. Vedere la sezione ""Script Python per ogni caso di utilizzo principale"" per ulteriori dettagli sull'elaborazione dei dati, sul training del modello e sul calcolo della precisione del modello in `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
```

Il runtime risultante con dieci epoche di training è stato il seguente:

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

Ci sono voluti più di 43 minuti per elaborare i dati di input, formare un modello DNN, calcolare la precisione e produrre checkpoint TensorFlow e un file CSV per i risultati delle previsioni. Abbiamo limitato il numero di epoche di training a 10, che in pratica è spesso impostato a 100 per garantire una precisione del modello soddisfacente. Il tempo di training in genere è in grado di scalare in modo lineare con il numero di epoche.

Successivamente, abbiamo utilizzato i quattro nodi di lavoro disponibili nel cluster ed eseguito lo stesso script in `yarn` Modalità con dati in 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
```

Il runtime risultante è stato migliorato come segue:

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

Con il modello di Horovod e il parallelismo dei dati in Spark, abbiamo visto una velocità di runtime di 5,29x `yarn` contro `local` con dieci epoche di training. Questo è mostrato nella figura seguente con le legende `HDFS` e. `Local`. Il training sul modello DNN TensorFlow sottostante può essere ulteriormente accelerato con le GPU, se disponibili. Prevediamo di condurre questo test e di pubblicare i risultati in un report tecnico futuro.

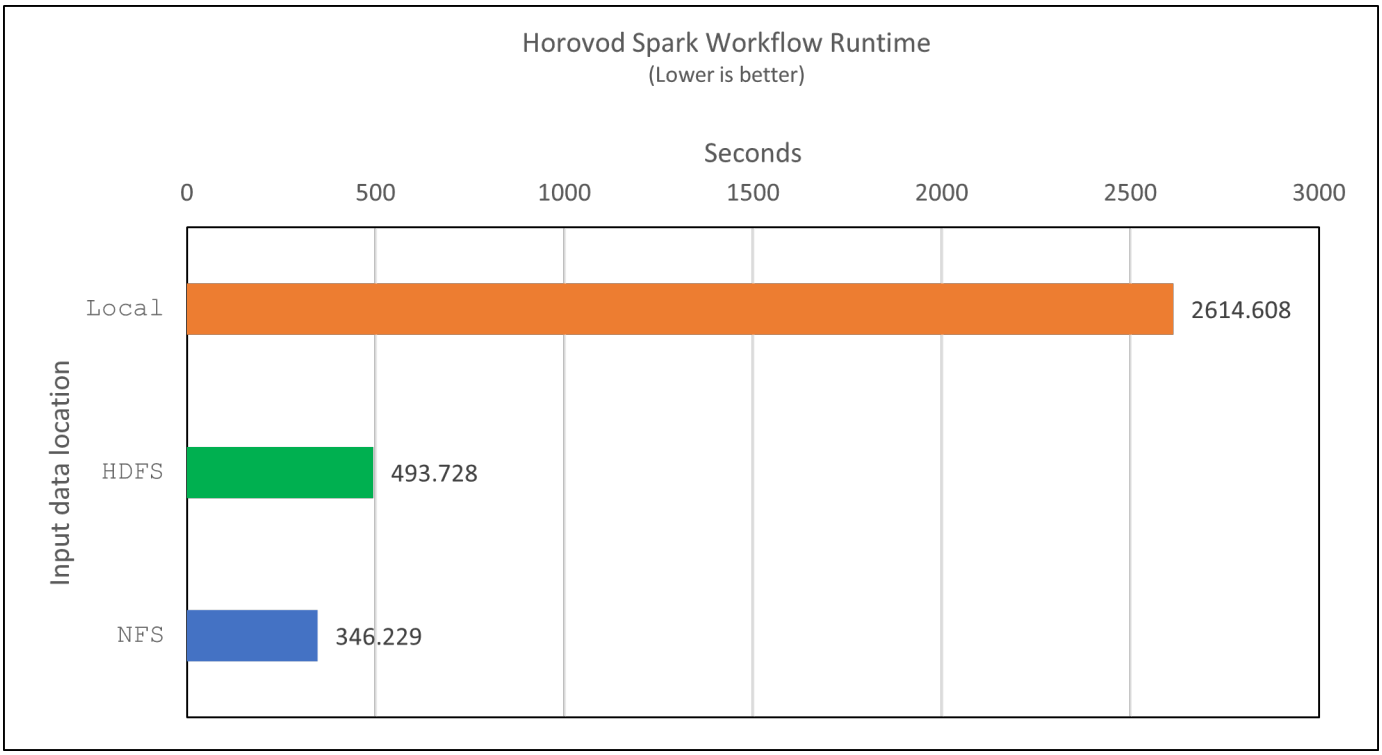
Il nostro test successivo ha confrontato i runtime con i dati di input che risiedono in NFS rispetto a HDFS. Il volume NFS su AFF A800 è stato montato `/sparkdemo/horovod` Tra i cinque nodi (un master, quattro dipendenti) nel cluster Spark. Abbiamo eseguito un comando simile a quello dei test precedenti, con `--data-dir` Parametro ora che punta al montaggio 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
```

Il runtime risultante con NFS è stato il seguente:

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

Si è verificato un ulteriore velocismo di 1,43 volte, come mostrato nella figura seguente. Pertanto, con uno storage all-flash NetApp collegato al cluster, i clienti possono usufruire dei vantaggi di un rapido trasferimento e distribuzione dei dati per i flussi di lavoro di Horovod Spark, ottenendo una velocità di 7,55 volte superiore rispetto all'esecuzione su un singolo nodo.



## Modelli di deep learning per performance di previsione CTR

Per i sistemi di raccomandazione progettati per massimizzare il CTR, è necessario imparare sofisticate interazioni di funzionalità dietro i comportamenti degli utenti che possono essere calcolati matematicamente da basso ordine a alto ordine. Le interazioni di funzionalità di basso e alto ordine devono essere ugualmente importanti per un buon modello di deep learning senza polarizzare l'uno o l'altro. DeepFM (Deep Factorization Machine), una rete neurale basata su macchine per la fattorizzazione, combina macchine per la fattorizzazione per consigli e un apprendimento approfondito per l'apprendimento delle funzionalità in una nuova architettura di rete neurale.

Anche se le macchine convenzionali di fattorizzazione modellano le interazioni a coppie come prodotto interno di vettori latenti tra le funzionalità e possono teoricamente acquisire informazioni di ordine elevato, in pratica, i professionisti dell'apprendimento automatico di solito utilizzano solo le interazioni di funzionalità di secondo ordine a causa dell'elevata complessità di calcolo e storage. Varianti di rete neurali profonde come quelle di Google " modelli profondi" d'altro canto, impara sofisticate interazioni di funzionalità in una struttura di rete ibrida combinando un modello ampio lineare e un modello profondo.

Ci sono due input per questo modello ampio e profondo, uno per il modello ampio sottostante e l'altro per il deep, l'ultima parte del quale richiede ancora un esperto di ingegneria delle funzionalità e quindi rende la tecnica meno generalizzabile per altri domini. A differenza di Wide & Deep Model, DeepFM può essere addestrato in modo efficiente con funzionalità raw senza alcuna progettazione delle funzioni, perché la sua parte ampia e profonda condividono lo stesso input e lo stesso vettore di inclusione.

Abbiamo elaborato per la prima volta il Criteo `train.txt` (11 GB) in un file CSV denominato `ctr_train.csv` Memorizzato in un montaggio NFS `/sparkdemo/tr-4570-data` utilizzo di `run_classification_criteo_spark.py` dalla sezione ""Script Python per ogni caso di utilizzo principale"." All'interno di questo script, la funzione `process_input_file` esegue diversi metodi di stringa per rimuovere le schede e inserire `','` come delimitatore e. `'\n'` come novità. Tenere presente che è necessario elaborare solo l'originale `train.txt` una volta, in modo che il blocco di codice sia visualizzato come commenti.

Per i seguenti test di diversi modelli DL, abbiamo utilizzato `ctr_train.csv` come file di input. Nelle successive esecuzioni dei test, il file CSV di input è stato letto in un Spark DataFrame con schema contenente un campo di `'label'`, caratteristiche ad alta densità di numeri interi `['I1', 'I2', 'I3', …, 'I13']`e funzioni sparse `['C1', 'C2', 'C3', …, 'C26']`. Quanto segue `spark-submit` Command acquisisce un input CSV, allena i modelli DeepFM con una suddivisione del 20% per la convalida incrociata e sceglie il modello migliore dopo dieci epoche di training per calcolare l'accuratezza della previsione sul set di test:

```
(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
```

Tenere presente che dal file di dati `ctr_train.csv` È superiore a 11 GB, è necessario impostare un valore sufficiente `spark.driver.maxResultSize` maggiore della dimensione del set di dati per evitare errori.

```
 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()
```

In quanto sopra `SparkSession.builder` anche la configurazione è stata abilitata "Freccia Apache", Che converte un DataFrame Spark in un DataFrame Pandas con `df.toPandas()` metodo.

```
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.
```

Dopo la suddivisione casuale, nel set di dati di training sono presenti più di 36 M di righe e 9 M di esempi nel set di test:

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

Poiché questo report tecnico è incentrato sul test della CPU senza utilizzare alcuna GPU, è fondamentale creare TensorFlow con i flag appropriati del compilatore. Questo passaggio evita di invocare librerie con accelerazione GPU e sfrutta al meglio le istruzioni AVX (Advanced Vector Extensions) e AVX2 di TensorFlow. Queste funzionalità sono progettate per calcoli algebrici lineari come addizione vettorizzata, moltiplicazioni di matrice all'interno di un training feed-forward o DNN back-propagation. L'istruzione FMA (Fused Multiply Add) disponibile con AVX2 che utilizza registri a virgola mobile (FP) a 256 bit è ideale per i tipi di dati e codice intero, con una velocità fino a 2 volte superiore. Per il codice FP e i tipi di dati, AVX2 raggiunge una velocità dell'8% su 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.
```

Per creare TensorFlow dall'origine, NetApp consiglia di utilizzare "Bazel". Per il nostro ambiente, abbiamo eseguito i seguenti comandi nel prompt della shell per l'installazione `dnf`, `dnf-plugins`E Bazel.

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

È necessario abilitare GCC 5 o versioni successive per utilizzare le funzionalità C++17 durante il processo di creazione, fornito da RHEL con Software Collections Library (SCL). I seguenti comandi vengono installati `devtoolset` E GCC 11.2.1 sul nostro 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
```

Si noti che gli ultimi due comandi sono disponibili `devtoolset-11`, che utilizza `/opt/rh/devtoolset-11/root/usr/bin/gcc` (GCC 11.2.1). Inoltre, assicurarsi di `git` La versione è superiore alla 1.8.3 (fornita con RHEL 7.9). Fare riferimento a questo "articolo" per l'aggiornamento `git` a 2.24.1.

Supponiamo che tu abbia già clonato l'ultimo repo master TensorFlow. Quindi, creare un `workspace` directory con un `WORKSPACE` File per la creazione di TensorFlow dall'origine con AVX, AVX2 e FMA. Eseguire `configure` E specificare la posizione binaria di Python corretta. "CUDA" È disattivato per i test perché non abbiamo utilizzato una GPU. R `.bazelrc` il file viene generato in base alle impostazioni. Inoltre, abbiamo modificato il file e il set `build --define=no_hdfs_support=false` Per attivare il supporto HDFS. Fare riferimento a. `.bazelrc` nella sezione ""Script Python per ogni caso di utilizzo principale"," per un elenco completo di impostazioni e flag.

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

Dopo aver creato TensorFlow con i flag corretti, eseguire il seguente script per elaborare il set di dati Criteo Display Ads, formare un modello DeepFM e calcolare l'area sotto la curva caratteristica operativa ricevitore (ROC AUC) in base ai punteggi di previsione.

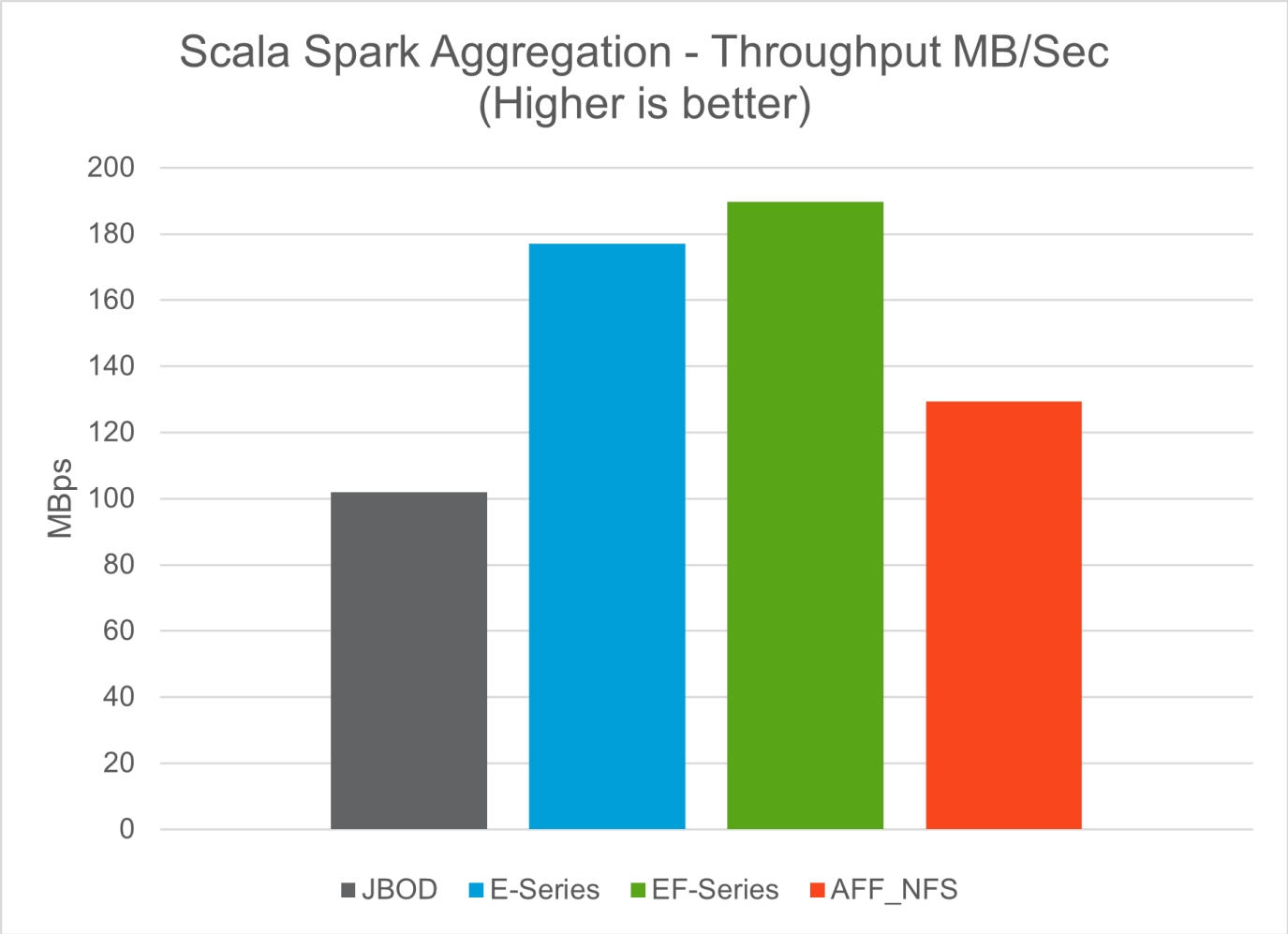
```
(base) [root@n138 examples]# ~/anaconda3/bin/spark-submit
--master yarn
--executor-memory 15g
--executor-cores 1
--num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py
--data-dir file:///sparkdemo/tr-4570-data
> . /run_classification_criteo_spark_nfs.log 2>&1
```

Dopo dieci epoche di training, abbiamo ottenuto il punteggio AUC nel set di dati di 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
```

In modo simile ai casi di utilizzo precedenti, abbiamo confrontato il runtime del workflow Spark con i dati che

risiedono in posizioni diverse. La figura seguente mostra un confronto della previsione CTR di apprendimento approfondito per un runtime di workflow Spark.

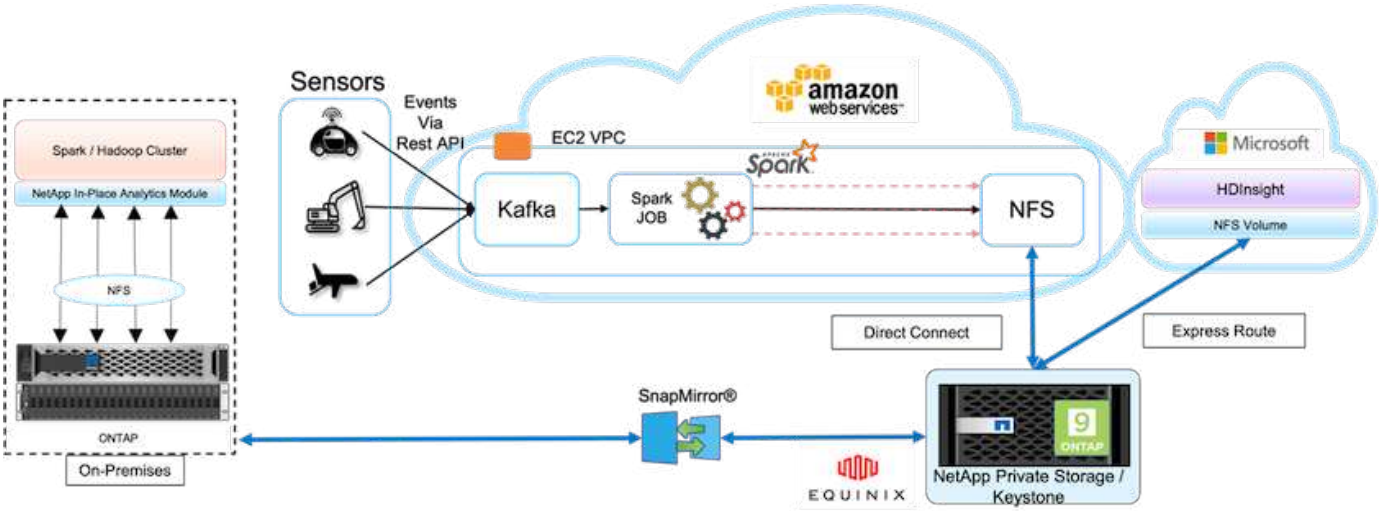



## Soluzione di cloud ibrido

Un moderno data center aziendale è un cloud ibrido che connette più ambienti di infrastruttura distribuita attraverso un piano di gestione continua dei dati con un modello operativo coerente, on-premise e/o in più cloud pubblici. Per ottenere il massimo da un cloud ibrido, devi essere in grado di spostare perfettamente i dati tra ambienti on-premise e multi-cloud senza la necessità di conversioni di dati o refactoring delle applicazioni.

I clienti hanno indicato di iniziare il loro percorso nel cloud ibrido spostando lo storage secondario nel cloud per casi di utilizzo come la protezione dei dati o spostando meno carichi di lavoro business-critical come lo sviluppo delle applicazioni e DevOps nel cloud. Passano quindi a carichi di lavoro più critici. Hosting di contenuti e web, sviluppo di applicazioni e DevOps, database, analytics e applicazioni containerizzate sono tra i carichi di lavoro di cloud ibrido più diffusi. La complessità, i costi e i rischi dei progetti ai aziendali hanno storicamente ostacolato l'adozione dell'ai dalla fase sperimentale alla produzione.

Con una soluzione di cloud ibrido NetApp, i clienti possono beneficiare di strumenti integrati per la sicurezza, la governance dei dati e la conformità con un unico pannello di controllo per la gestione dei dati e del workflow in ambienti distribuiti, ottimizzando al contempo il costo totale di proprietà in base al consumo. La figura seguente è una soluzione di esempio di un partner di servizi cloud che ha il compito di fornire connettività multi-cloud per i dati di analisi dei big data dei clienti.




In questo scenario, i dati IoT ricevuti in AWS da diverse origini vengono memorizzati in una posizione centrale in NetApp Private Storage (NPS). Lo storage NPS è connesso ai cluster Spark o Hadoop situati in AWS e Azure, consentendo l'esecuzione di applicazioni di big data analytics in più cloud che accedono agli stessi dati. I requisiti e le sfide principali per questo caso di utilizzo includono:

- I clienti vogliono eseguire lavori di analisi sugli stessi dati utilizzando più cloud.
- I dati devono essere ricevuti da fonti diverse, ad esempio ambienti on-premise e cloud, attraverso diversi sensori e hub.
- La soluzione deve essere efficiente e conveniente.
- La sfida principale è quella di creare una soluzione conveniente ed efficiente in grado di offrire servizi di analisi ibridi tra diversi ambienti on-premise e cloud.

La nostra soluzione per la protezione dei dati e la connettività multicloud risolve il problema di avere applicazioni di analisi del cloud su più hyperscaler. Come mostrato nella figura precedente, i dati provenienti dai sensori vengono trasmessi e acquisiti nel cluster AWS Spark tramite Kafka. I dati vengono memorizzati in una condivisione NFS residente in NPS, che si trova all'esterno del cloud provider all'interno di un data center Equinix.

Poiché NetApp NPS è connesso ad Amazon AWS e Microsoft Azure rispettivamente tramite Direct Connect e Express Route Connections, i clienti possono sfruttare il modulo di analisi in-place per accedere ai dati da entrambi i cluster di analisi Amazon e AWS. Di conseguenza, poiché sia lo storage on-premise che NPS esegue il software ONTAP, "SnapMirror" Può eseguire il mirroring dei dati NPS nel cluster on-premise, fornendo analisi del cloud ibrido su cloud on-premise e multipli.

Per ottenere le migliori performance, NetApp consiglia di utilizzare più interfacce di rete e connessioni dirette o percorsi espressi per accedere ai dati dalle istanze cloud. Abbiamo altre soluzioni di data mover, tra cui "XCP" e. "Copia e sincronizzazione di BlueXP" Per aiutare i clienti a creare cluster Spark di cloud ibrido basati su applicazioni, sicuri e convenienti.

# Script Python per ogni caso di utilizzo principale

I tre script Python riportati di seguito corrispondono ai tre principali casi di utilizzo testati. Il primo è `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}")
```

Il secondo script è `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()
```

Il terzo script è `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))
```

# Conclusione

In questo documento, discutiamo dell'architettura di Apache Spark, dei casi di utilizzo dei clienti e del portfolio di storage NetApp in relazione a big data, analytics moderni e ai, ML e DL. Nei nostri test di convalida delle performance basati su strumenti di benchmarking standard di settore e sulla domanda dei clienti, le soluzioni NetApp Spark hanno dimostrato performance superiori rispetto ai sistemi Hadoop nativi. Una combinazione dei casi di utilizzo dei clienti e dei risultati delle performance presentati in questo report può aiutarti a scegliere una soluzione Spark appropriata per la tua implementazione.

# Dove trovare ulteriori informazioni

In questo TR sono stati utilizzati i seguenti riferimenti:

- Architettura e componenti di Apache Spark

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

- Casi di utilizzo di Apache Spark

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

- Le sfide di Apache

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

- NLP. Scintilla

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

- BERT

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

- Deep and Cross Network for ad Click Predictions

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

- FlexGroup

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

- ETL streaming

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

- Soluzioni NetApp e-Series per Hadoop

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

- Analisi del sentimento da Customer Communications con NetApp ai

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

- Soluzioni NetApp per l’analisi dei dati moderna

  "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

- Copia e sincronizzazione di BlueXP

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

- Toolkit DataOps

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