**Hadoop**: una **piattaforma** **scalabile e affidabile** per lo storage e l’analisi condivisa, che

è Open Source e gira su h**ardware commodity**, quindi è **costo-efficace.** Se è necessario avere una maggiore

capacità di risorse, basterà aggiungerne qualcuna, per rendere più efficace l’intero sistema (**scalabilità orizzontale**).

Esistono due s**trumenti principali in Hadoop**, sui quali sono installati tutti gli altri strumenti.

* **File system distribuito Hadoop**: parte di archiviazione (**HDFS**)
* **MapReduce** – parte di elaborazione (**MR**)

HDFS **non supporta l'operazione di aggiornamento**, pertanto i dati in HDFS vengono scritti una volta e letti molte volte (**WORM**).

HDFS ha tre **sottocomponenti**:

* **Name Node (NN):**NN è un servizio/daemon centralizzato che funge da gestore di archiviazione del cluster.

Le sue responsabilità principali sono:

• mantenimento dei metadati di file e directory.

• controllo e coordinamento dei **DN** per le operazioni del sistema file come creare, leggere, scrivere, ecc.

I client comunicano con NN per eseguire le operazioni quotidiane. A sua volta, NN fornisce ai client la posizione dei DN nel cluster (dove è disponibile il blocco dati) per eseguire operazioni.

* **Secondary Name Node (SNN)**
* **Data Node (DN) :** Il demone DN è responsabile della gestione dei dischi di archiviazione locali. Gestisce le

operazioni del file system come la creazione, la lettura, l'apertura, la chiusura e l'eliminazione dei blocchi. Il DN non sa nulla dei blocchi dati. Memorizza ogni blocco come file nel file system locale. I blocchi vengono caricati/eliminati nei DN in base alle istruzioni di NN, che convalida ed elabora le richieste dei client. NN non esegue alcuna operazione di lettura/scrittura per i client. I clienti comunicano con NN per conoscere la posizione dei blocchi e reindirizzati ai DN per eseguire operazioni di lettura/scrittura.

**MapReduce (MR)** – MR è un modello di programmazione **distribuito, scalabile, tollerante ai guasti e parallelo** per l'elaborazione di big data su un **cluster di macchine di base a basso costo e inaffidabili**.

**MR consente di scrivere lavori distribuiti e scalabili con poco sforzo**

MR ha due **sottocomponenti**:

* **Job Tracker (JT)**
* **Task Tracker (TT).**

La MR è comunemente preferita per le seguenti applicazioni:

• Ricerca, ordinamento, raggruppamento.

• Statistiche semplici: conteggio, classifica.

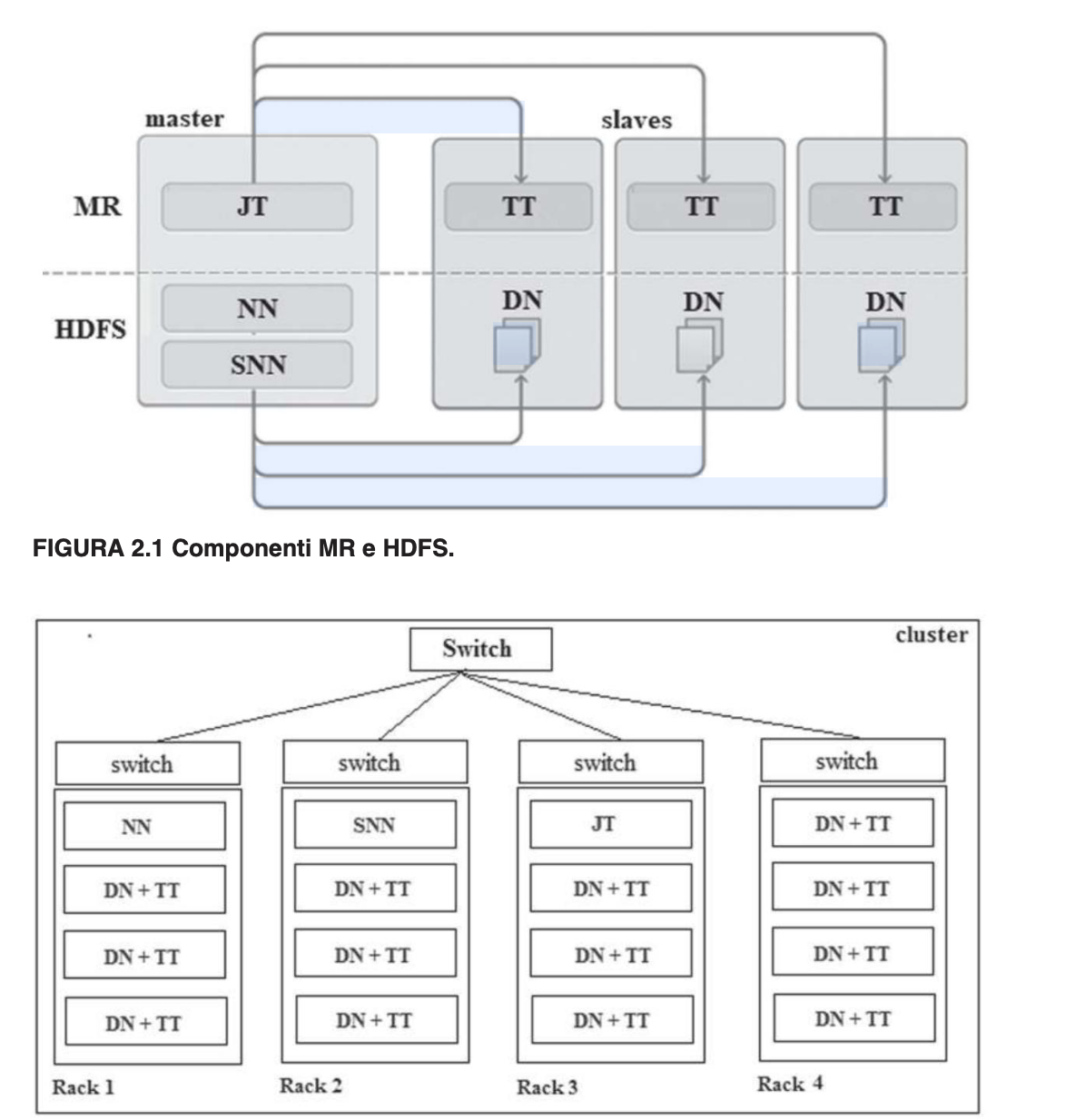
• Statistiche complesse: PCA, covarianza.

• Pre-elaborazione di enormi quantità di dati per applicare algoritmi di apprendimento automatico.

• Classificazione: bayes naïve, foresta casuale, regressione.

• Clustering: k significa, gerarchico, densità, bi-clustering.

• Elaborazione del testo, creazione di indici, creazione e analisi di grafici, riconoscimento di modelli, filtraggio collaborativo, analisi del sentiment.



**HDFS** gestisce i big data su un cluster di macchine con modello di accesso ai dati in **streaming**. Utilizza l'archiviazione distribuita per fornire una visualizzazione del disco singolo e per fornire **uno spazio dei nomi globale univoco sull'archiviazione distribuita**. Si tratta di un file system appositamente progettato con f**unzionalità quali distribuzione, tolleranza agli errori, replica e distribuito su macchine a basso costo e inaffidabili**. . Dal punto di vista dell'utente, sembra essere un grande spazio di archiviazione centralizzato, ma dal punto di vista del sistema è il singolo server che contribuisce al suo spazio di archiviazione.

**HDFS** fornisce l'astrazione del file, il che significa che un file oltre la dimensione del disco di archiviazione viene partizionato in blocchi e archiviato in un cluster di nodi.

Per un utente normale, **il file enorme viene logicamente mostrato come un singolo file**, ma in realtà parti di questo file sono archiviate in diversi nodi nel cluster. HDFS è immutabile, il che significa che è possibile solo il caricamento iniziale,non esiste alcuna funzionalità di modifica dei file in HDFS utilizzando vi, gedit, ecc. poiché comporta un enorme sovraccarico di I/O.

Usa HDFS quando vuoi

• archiviare dati di grandi dimensioni (oltre TB) su server di base.

• elaborare un numero limitato di file grandi rispetto a un numero elevato di file piccoli files.

• letture batch invece di letture/scritture casuali.

Non utilizzare HDFS

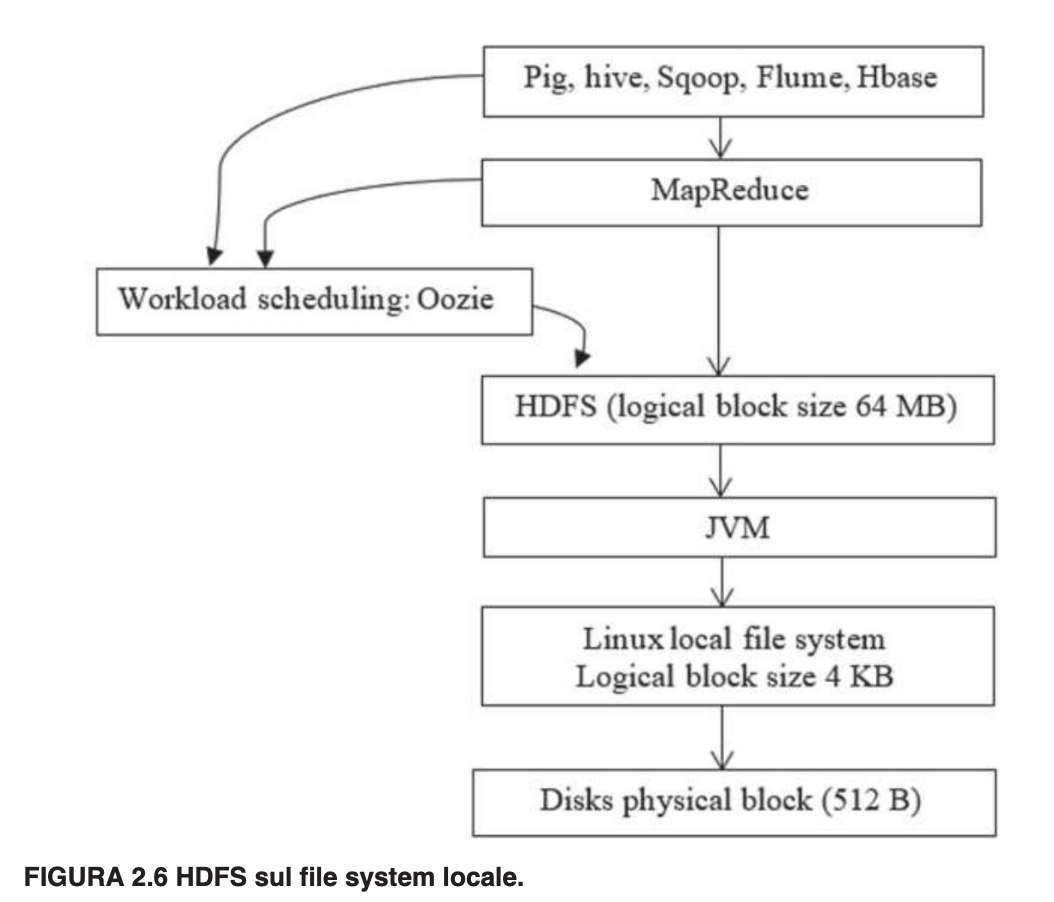
• per il processo di transazione e accesso a bassa latenza (in ms).

• elaborare molti file di piccole dimensioni (HDFS richiede più metadati e risorse).

• per la scrittura parallela (HDFS supporta solo la modalità di aggiunta).

• alla lettura arbitraria (HDFS fornisce solo la lettura batch).

L'unità di archiviazione e accesso ai dati in HDFS è un blocco, che denota la quantità minima di dati che possono essere archiviati e recuperati da HDFS. I dati che volevamo caricare su HDFS vengono divisi in blocchi di uguali dimensioni (chiamati blocchi) e archiviati nei DN nel cluster. I DN memorizzano ciascun blocco di dati come file nel relativo file system locale. Il file system Linux utilizza una dimensione di blocco logico di 4 KB (un gruppo di più blocchi fisici) . I blocchi piccoli sono adatti per i database transazionali. I blocchi di grandi dimensioni sono utili per i database analitici. FS di Linux mantiene i metadati a livello file. HDFS utilizza la dimensione del blocco logico di 64 MB. Pertanto, mantiene i metadati a livello di blocco. **HDFS è un file system logico sopra file system locale**, come mostrato nella Figura 2.6. Non è un vero e proprio sistema file come ext3, ext4 che funziona a livello di disco fisico. Pertanto, HDFS viene eseguito su sistemi file locali (ext3/ext4) e non interagisce direttamente con i dispositivi di archiviazione.



***POSIZIONAMENTO DELLA REPLICAZIONE USANDO RACK AWARENESS***

Poiché la distribuzione di Hadoop viene eseguita principalmente con server di base inaffidabili, è più probabile il

su RAID poiché risolve solo i guasti dell'HDD e il mirroring è costoso per i big data.

Il numero di repliche dei blocchi di dati è indicato come RF, che è tre per impostazione predefinita. La replica aiuta a ottenere tolleranza agli errori, prestazioni di rete e una migliore elaborazione parallela dei dati.

Se memorizzi tre copie di un blocco nella stessa macchina, se la macchina è inattiva, nessuna delle copie sarà accessibile. Quindi, quel particolare blocco è perso. Pertanto n**on ha senso** fare più di una **copia** di un **blocco sulla stessa macchina**. Spreca memoria e non fornisce alcuna tolleranza agli errori. Pertanto, ogni copia viene distribuita su più di un nodo nel cluster. La regola pratica è che RF deve essere inferiore o uguale al numero di nodi nel cluster.

**NN decide dove posizionare i blocchi replicati in base alla topologia del cluster.**  **Questa è chiamata consapevolezza del rack**.

La **larghezza di banda** è una risorsa **scarsa** nei data center. La larghezza di banda

**intra-rack è maggiore** rispetto alla comunicazione **inter-rack (tra rack)**. Quindi, per ridurre al minimo il

trasferimento di blocchi tra cluster frequentemente, la distanza viene calcolata in base

alla posizione del server e utilizzata da NN durante il posizionamento del blocco. **Lo**

**scheduler MR utilizza anche la distanza per determinare dove è disponibile la replica**

**più vicina per avviare le attività di map/reduce**. Di seguito è riportata la distanza tipica dei server

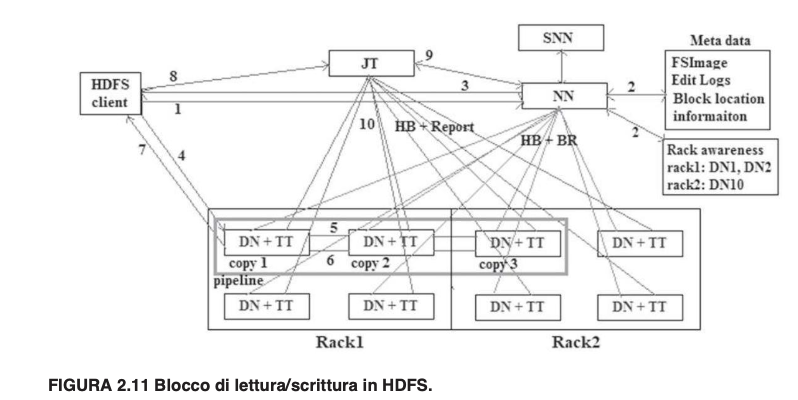
D = 0 stesso nodo

D = 2 distanza tra i nodi nello stesso rack

D = 4 distanza tra i nodi in rack diversi

D = 6 distanza tra i nodi di diversi data center

Due rack ciascuno con quattro nodi che eseguono DN+TT e quattro nodi separati per eseguire il client HDFS, JT, NN, SNN nel cluster. Un utente si connette in remoto al client HDFS per caricare big data su HDFS. La Figura 2.11 illustra il flusso di lettura/scrittura del blocco dati. Il client HDFS è responsabile della divisione di un file di input in blocchi e inoltra la richiesta dell'utente a NN. Questo demone può essere eseguito in un nodo separato nel cluster o co-localizzare con altri servizi HDFS (NN/DN/SNN). Un nodo che esegue il client HDFS è anche chiamato nodo gateway.



***SCRITTURA DI FILE SU HDFS***

Passaggio 1: una volta che il client HDFS riceve la richiesta di caricamento del file, divide fisicamente il file in blocchi e richiede NN dove archiviare tutti questi blocchi.Poiché il client HDFS non sa quale DN dispone di spazio libero e dove archiviarlo in base al riconoscimento del rack.

Passaggio 2: NN si iferisce ai metadati (file FSImage) e determina tre posizioni in base alla consapevolezza

del rack per ciascun blocco. Quindi, NN invia un elenco di tre posizioni DN per ciascun

blocco. Supponiamo che le posizioni per il blocco 1 siano DN1, DN2 e DN3. Il suggerimento del numero di DN dipende dalla RF. Per impostazione predefinita, vengono suggeriti tre DN per ciascun blocco poiché RF predefinito è 3. Se il nodo client si trova nel cluster Hadoop, è il primo nodo a copiare il blocco dati.

Passaggio 3: dopo aver ricevuto tre posizioni per ciascun blocco, il client HDFS è pronto per scrivere i blocchi rispettivi DN in parallelo aprendo il flusso di output HDFS. Non vi è alcun vincolo che il blocco 2 debba copiato su HDFS solo dopo il blocco 1. Tutti i blocchi vengono copiati nel cluster in base al numero di thread che un client HDFS può servire.

Passaggio 4: il client HDFS divide ulteriormente logicamente il blocco 1 in una coda di pacchetti (ciascuno da 64 KB), denominata coda dati. Per block1, il client HDFS forma una pipeline con DN1, DN2 e DN3.

Passaggio 5: il primo pacchetto dalla coda dati viene copiato su DN1, che memorizza e inoltra il pacchetto a DN2 lungo la pipeline.

Passaggio 7: il client HDFS riceve una conferma dai DN che indica che il blocco è stato copiato correttamente. Il client HDFS invia un messaggio di successo all'utente. Tutti questi processi sono altamente trasparenti per l'utente. L'utente non sa dove sono archiviati i blocchi di dati. Il lavoro MR può essere avviato solo

dopo che tutti i blocchi sono stati caricati correttamente in HDFS. L'utente ha anche il privilegio di specificare la dimensione del blocco, RF durante il caricamento di big data.

***2.4.3.3 LETTURA DI FILE DA HDFS***

Passaggio 1: l'utente avvia un comando di lettura al client HDFS, che inoltra la richiesta a NN per trovare la posizione dei blocchi per il file richiesto.

Passaggio 2: NN fa riferimento ai suoi metadati e trova un elenco di DN in cui è stato archiviato ciascun blocco. Questo elenco viene preparato in base alla posizione (riconoscimento nel rack) delle copie in blocco.

Passaggio 3: il client HDFS è pronto per leggere blocchi da diversi DN creando un flusso di input HDFS.

Passaggio 4: il client HDFS preferisce i DN vicini al client (per ridurre il traffico di rete). Se il primo DN dell'elenco non è raggiungibile, viene scelto il secondo DN dell'elenco.

Passo 5: I blocchi di un file vengono letti in sequenza uno per uno secondo l'ordine di costruzione del file originale. Non è utile leggere più blocchi contemporaneamente. Infine, i DN trasmettono i blocchi di dati al client in ordine.

**MAPREDUCE (MR)**

MR è uno strumento di elaborazione batch in parallelo dei dati altamente distribuito,

scalabile orizzontalmente, con tolleranza agli errori nel framework Hadoop che viene

eseguito su un cluster di server di base inaffidabili per elaborare big data. I dati vengono

raccolti e a**rchiviati su HDFS prima di avviare i lavori MR**. La programmazione con MPI per

l'elaborazione parallela distribuita ha una scalabilità limitata, inoltre, le responsabilità del

programmatore sono enormi, come accennato in precedenza. Al contrario, è facile

sviluppare algoritmi scalabili utilizzando la MR senza ulteriori sforzi per gestire un sistema

distribuito. **Un programma sviluppato per un nodo può essere utilizzato per migliaia di nodi**

**senza modificare nuovamente il codice.** MR stesso parallelizza l'esecuzione, quindi gli utenti

non devono investire sforzi per l'esecuzione parallela.

L**a programmazione MR si basa su LISP (LISt Programming), un tipo di paradigma di**

**programmazione funzionale** (come Haskell, Scala, Clojure, Smalltalk, Ruby). Tutto nella

programmazione funzionale ruota attorno alla funzione. Una funzione può essere passata/

ricevuta come argomenti, distribuita tra i nodi. Una funzione che accetta un'altra funzione

come argomento è detta funzione di ordine superiore. **La programmazione funzionale non**

**mantiene lo stato e non supporta il blocco e la sincronizzazione**. **Quindi, è scalabile**. Le

**attività MR vengono eseguite in modo indipendente**, **quindi è facile gestire i guasti parziali.**

Le prestazioni della MR sono inferiori rispetto ai linguaggi di programmazione generali durante l'elaborazione di set di dati di poche 100 MB, poiché la MR consiste in una sequenza di passaggi da eseguire. Tuttavia, per sperimentare la vera potenza della MR, si dovrebbe lavorare c**on set di dati in TB**, perché è qui che RDBMS

impiega ore e fallisce, mentre Hadoop fa lo stesso in pochi minuti. Come già discusso, MR ha due sottocomponenti (vedi Figura 2.12): JT e TT. Un tipico lavoro MR esegue due attività:

mappare e ridurre

