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.qas\
  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): JT è un demone in esecuzione in un server dedicato nel cluster per gestire le risorse del cluster e i lavori MR. Le responsabilità principali di JT sono:

- gestione e monitoraggio delle risorse (CPU e memoria) nei TT.
- predisposizione del piano esecutivo e coordinamento delle fasi.
- pianificazione della mappatura/riduzione delle attività al TT.
- gestione del ciclo di vita dei lavori (dal momento della presentazione fino al completamento).
- mantenere la cronologia dei lavori (statistiche a livello di lavoro).
- fornire tolleranza agli errori.
- JT è inoltre in grado di riconoscere il rack durante la pianificazione delle attività di mappatura/ riduzione. Nella maggior parte dei casi, l'elaborazione avviene nei nodi in cui il blocco dati richiesto è fisicamente disponibile per ridurre il traffico di rete.

Task Tracker (TT). TT è un demone che viene eseguito in ogni nodo slave nel cluster Hadoop. Gestisce le risorse

- locali (CPU e memoria) come uno slot. Uno slot (chiamato anche contenitore) è un pacchetto
- logico fisso di una porzione di memoria e CPU per eseguire l'attività di mappatura/riduzione

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.

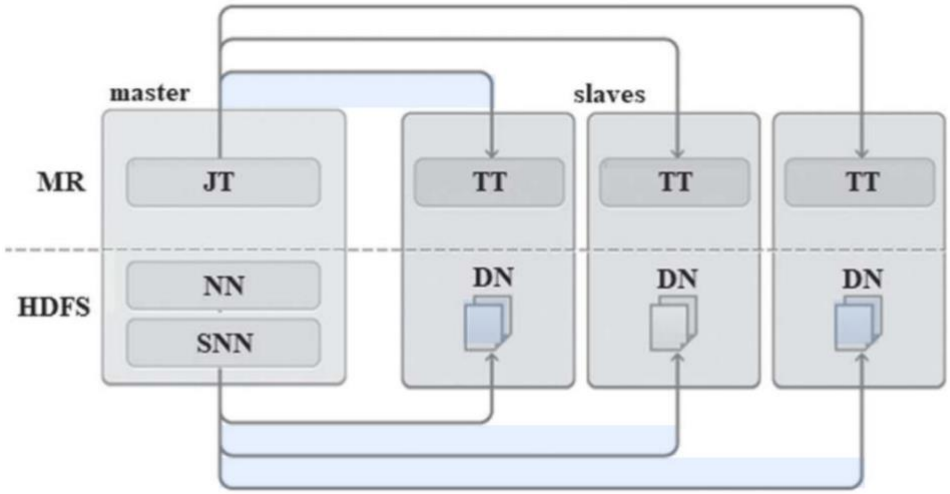
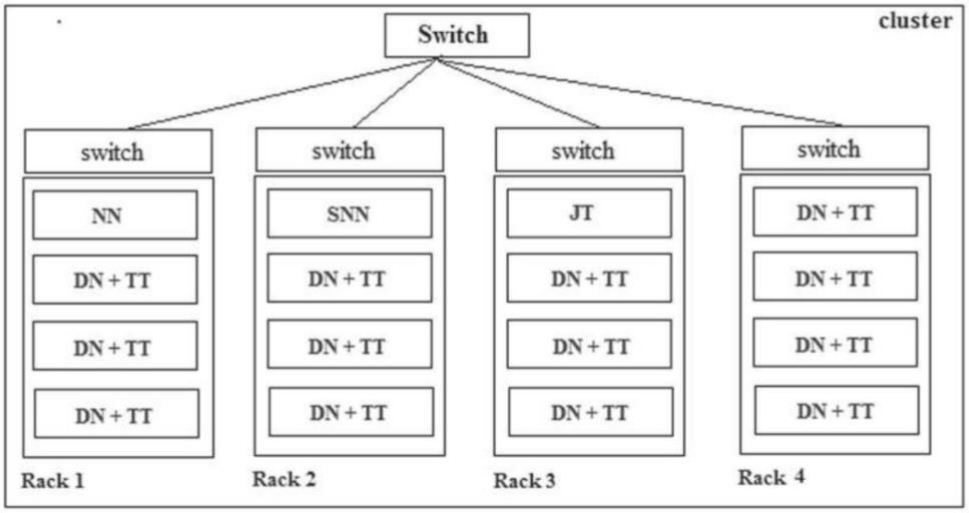


FIGURA 2.1 Componenti MR e HDFS.



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.

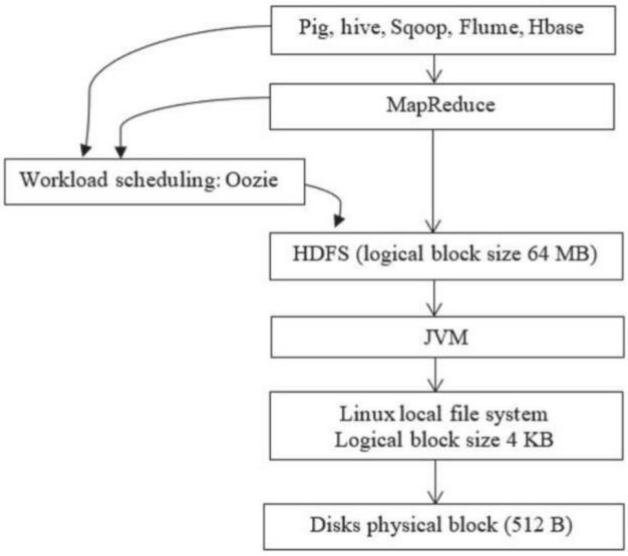


FIGURA 2.6 HDFS sul file system locale.

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.

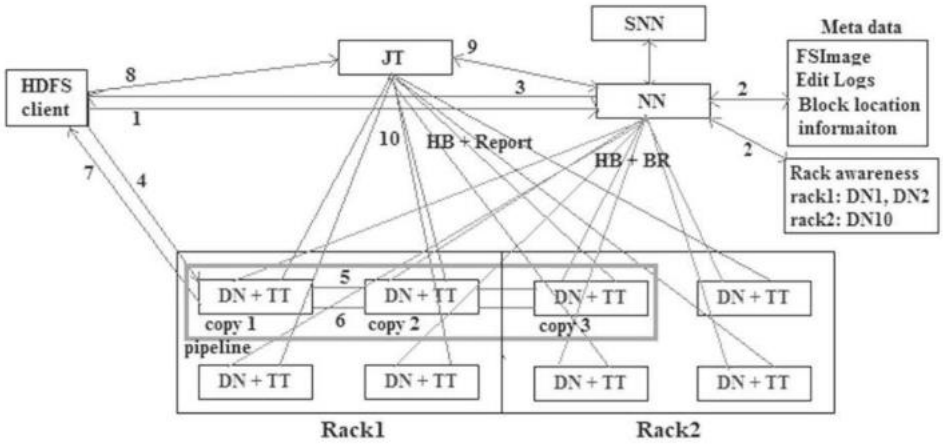


FIGURA 2.11 Blocco di lettura/scrittura in HDFS.

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.

****

Passaggio 8: presupponi che i dati siano già caricati su HDFS. Ora, un lavoro MR viene inviato a Job Client, che è un demone che fa parte di JT.

Passaggio 9: JT interrogherà NN sulle posizioni dei blocchi di un file di input specificato. NN risponde con un elenco di tre DN che hanno copie di ciascun blocco in base alla consapevolezza del rack.

Fase 10: JT prepara un piano di esecuzione ed esegue il coordinamento delle fasi. Similmente a NN, anche JT è altamente sensibile a rack/topologia. Sceglie per primo il DN più vicino avente il blocco desiderato. Se non è presente uno slot libero nel TT, viene scelto il DN successivo per ottenere l'esecuzione locale dei dati. Se non c'è nessuno slot libero nei tre slave dati che hanno la copia desiderata, una copia di quel blocco viene portata a qualsiasi altro TT che abbia uno slot libero per essere eseguita. Questa è chiamata esecuzione non locale e consuma più larghezza di banda della rete. L'esecuzione non locale è preferibile solo quando non è possibile ottenere l'esecuzione locale dei dati per un certo periodo di tempo. È sempre opportuno avviare il lavoro MR dopo che il file completo è stato caricato in HDFS poiché non è garantito che tutti i blocchi di un file siano stati scritti in HDFS.

****

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

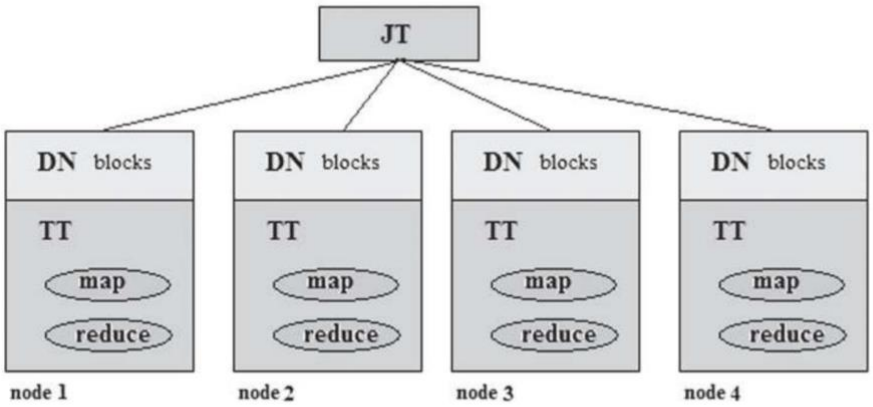


FIGURA 2.12 Componenti MapReduce.

FASI MAP REDUCE
In generale, ci sono due fasi per l'esecuzione di un lavoro MR, come mostrato nella Figura 2.14: fase di mappatura e fase di riduzione.

- La fase di mappa esegue una serie di attività di mappa (mapper) per leggere i dati dai dischi come coppie chiave-valore e produrre un numero arbitrario di coppie chiave-valore intermedie in base alla funzione di mappa definita dall'utente.
- La fase di riduzione esegue una serie di attività di riduzione (riduttori), che raccolgono l'output da tutte le attività della mappa, uniscono, ordinano in base alla chiave, raggruppano elenchi di valori che appartengono alla stessa chiave e producono l'output finale in base alla riduzione definita dall'utente funzione.

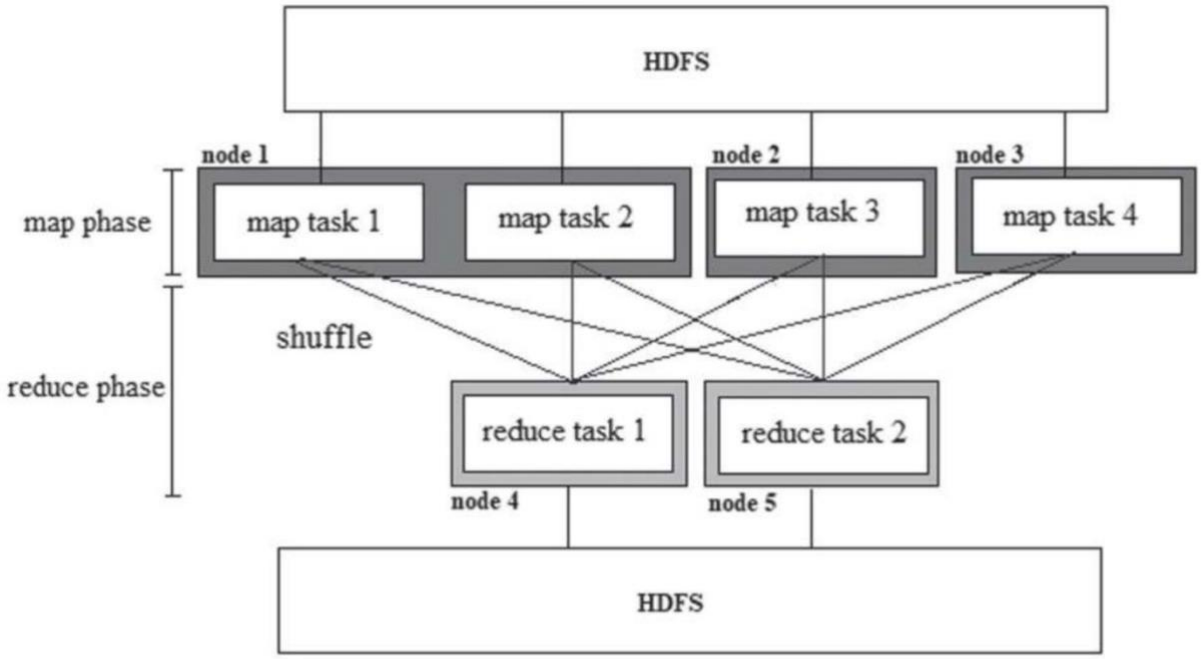
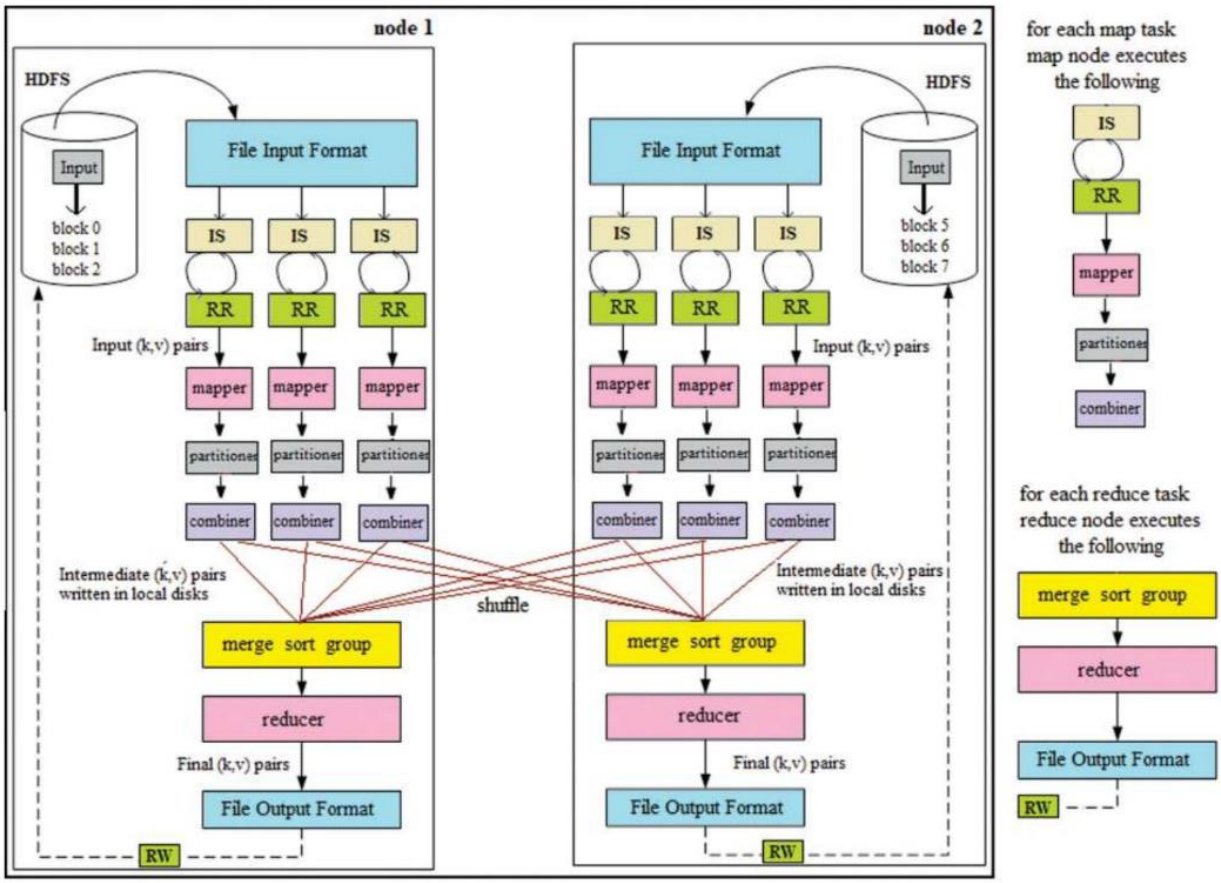


FIGURA 2.14 Fasi RM.



FASE MAPPA

HDFS divide fisicamente il file di input inviato in blocchi di uguali dimensioni e lo archivia in DN diversi in base alla consapevolezza del rack. I blocchi di dati richiesti vengono portati in memoria per alimentare l'attività della mappa per l'esecuzione per iniziare la fase della mappa. La fase della mappa inizia da FileInputFormat e termina quando tutte le attività della mappa sono state completate.

Il nodo della mappa per un'attività della mappa esegue la seguente sequenza di funzioni da eseguire insieme alla funzione della mappa:

Divisione input -->Lettore di record --> Mapper --> Partizionatore --> Combinator

Passaggio 1: FileInputFormat e Input Split (IS) o Dividi

Un IS rappresenta un insieme di blocchi che devono essere elaborati da una singola attività della mappa.

Block è il concetto HDFS mentre IS è il concetto MR. IS è il raggruppamento logico di uno o più blocchi fisici. IS farà riferimento ad almeno un blocco. Per impostazione predefinita, la dimensione IS è uguale alla dimensione del blocco predefinita (64 MB). Come mostrato in figura 2.16, il file di input da 256 MB è diviso in quattro blocchi fisici da 64 MB e archiviati in DN diversi.
Se la dimensione dell'IS è 128 MB, ogni IS raggruppa logicamente due blocchi fisici. Ogni IS viene elaborato da un'attività di mappa. Tieni presente che IS non contiene una copia dei blocchi fisici. Contiene solo la posizione dei blocchi fisici e i relativi metadati.

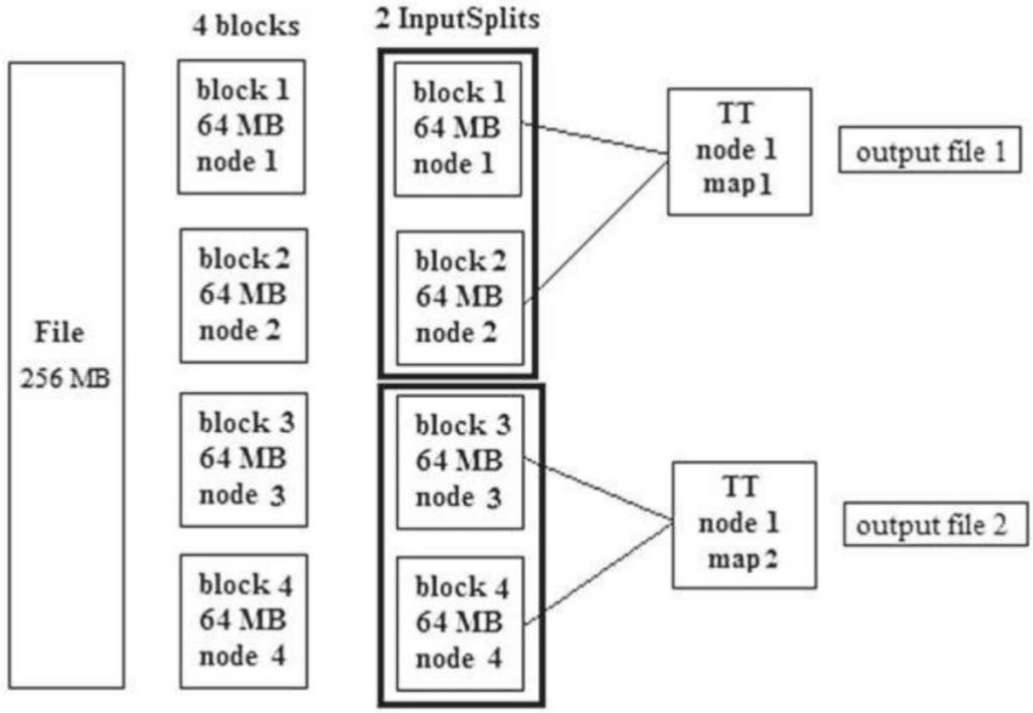
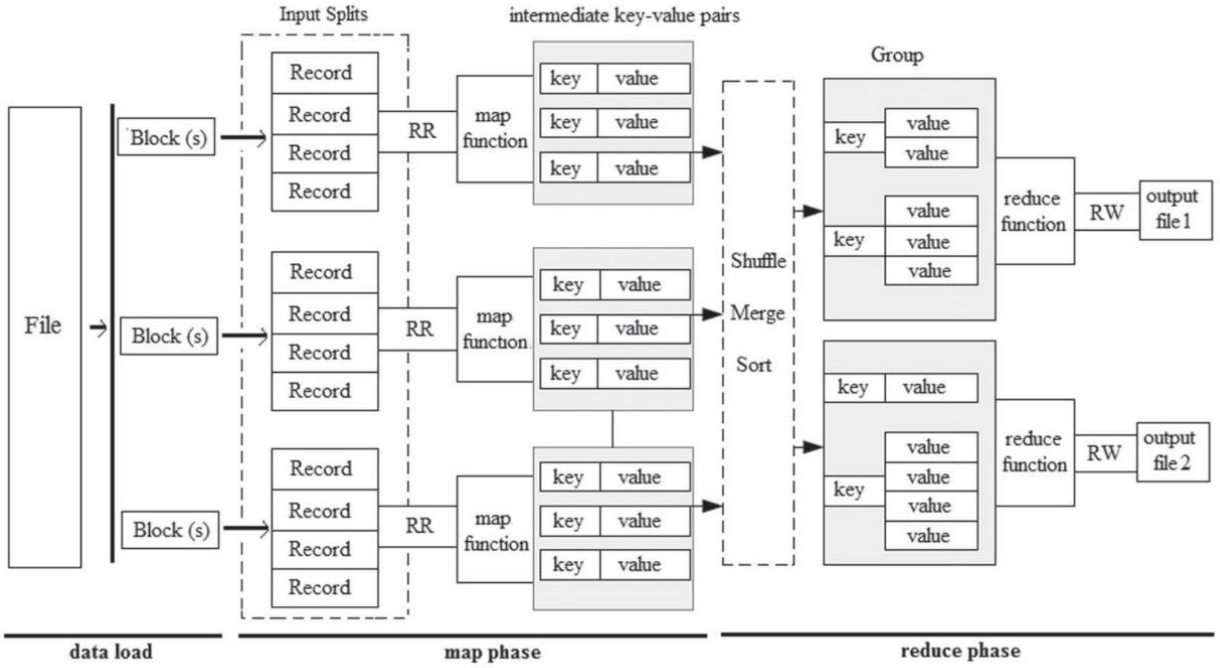


FIGURA 2.16 Blocco vs InputSplit.

coppia chiave-valore
Nell'ambiente di programmazione MR, nessun dato può reggere da solo. Ad ogni dato è associata una chiave. Una coppia chiave-valore è chiamata record. IS e i record sono entità logiche utilizzate al momento dell'esecuzione del lavoro. Non influenzano i blocchi fisici. Ogni passaggio nella sequenza di esecuzione MR accetta una coppia chiave-valore come input e restituisce coppie chiave-valore, come mostrato nella Figura 2.18.



Passaggio 3: attività di mappatura

Il mapping o l'attività di mappa è una funzione di mappa definita dall'utente, utilizzata principalmente per la pre-elaborazione dei record. Pertanto, l'output dell'attività di mappatura viene chiamato come coppie chiave-valore intermedie, ma non il risultato del lavoro MR. RR alimenta un record per mappare la funzione. Quando una funzione riceve input e produce output, viene chiamata task. Le attività di mappatura e riduzione vengono richiamate solo una volta al momento del lancio. Tuttavia, le funzioni map e reduce vengono richiamate per ogni record di input.
Ogni attività della mappa ha il suo RR, come mostrato nella Figura 2.15. Se sono presenti "n" record di input in IS, viene richiamata la funzione di mappa "n" volte e produce zero o più record di output. La funzione mappa consente agli utenti di scrivere la propria logica per decidere cosa fare con i record. È possibile analizzare il valore ed estrarre solo i campi rilevanti (proiezione) o filtrare i record indesiderati/errati o trasformare i record in entrata. L'output dell'attività della mappa viene archiviato in un buffer in memoria, come mostrato nella Figura 2.23. Dalla Figura 5.12 si può osservare che la fase della mappa termina solo dopo che tutte le attività di mappa di un lavoro sono state completate. La latenza di ciascuna attività della mappa può variare a causa di altre attività simultanee nel sistema

Passaggio 4: partizionamento (bilanciamento e riduzione dell'input dell'attività)
Se viene avviata più di un'attività di riduzione, il partizionatore decide a quale attività di riduzione deve essere indirizzato un record di output della mappa. Il partizionatore è una funzione che bilancia la riduzione delle dimensioni dell'input delle attività da tutte le attività della mappa. Il partizionatore ha senso solo quando lanciamo più di un'attività di riduzione. Il partizionatore suddivide l'output di un'attività di mappa in più partizioni. Una partizione è una porzione dell'output della mappa che va a un particolare riduttore. Il numero di partizioni è uguale al numero di attività di riduzione.
L'obiettivo del partizionamento è portare la stessa chiave da diverse attività della mappa un unico riduttore. Ad esempio, considera 10 attività sulla mappa per un lavoro di conteggio delle parole. Considera che le attività della mappa 1, 4, 6 e 10 producono la parola "Hadoop" come output. Per trovare il conteggio totale della parola "Hadoop", è necessario ridurre un'attività. Pertanto, la parola “Hadoop” può essere contata come 4.
Il partizionatore predefinito è HashPartitioner, che calcola il valore hash per decidere a
quale riduttore deve essere inviato il record corrente. Funziona bene con qualsiasi numero di partizioni e garantisce che ciascuna partizione abbia il giusto mix di chiavi, portando a
partizioni di dimensioni più uniformi.

Passaggio 5: Combinatore (ottimizzazione IO di rete e disco)

L'attività di combinazione riduce al minimo il traffico di rete, il trasferimento di I/O su disco e il numero di record elaborati dall'attività di riduzione.
la funzione combinatore viene utilizzata per ridurre al minimo la dimensione dell'output della mappa localmente dopo la preparazione delle partizioni.

Fase Reduce
La funzione di riduzione in ciascuna attività di riduzione viene eseguita solo dopo che tutte le attività della mappa sono state completate. In generale, ridurre l'output degli spostamenti di fase (partizioni) di mapper/combinatore dai nodi della mappa per ridurre i nodi, unisce tutte le partizioni, ordina in base alla chiave e raggruppa tutti i valori che appartengono alla stessa chiave per eliminare la ridondanza della chiave

Shuffle

Shuffle è il processo mediante il quale l'output partizionato del mapper/combiner viene
trasferito sulla rete HTTP a uno o più TT dove verranno ridotte le attività essere eseguito. Questa è anche chiamata fase di copia. Ciascun nodo di riduzione riceve una o più partizioni da tutte le attività della mappa. Se si decide di eseguire l'attività di riduzione
nello stesso nodo in cui viene completata l'attività di mappa, lo shuffle non ha alcun ruolo.
Ma come fa un nodomdi riduzione a sapere quale nodo della mappa interrogare per le partizioni? Questo viene fatto con l'aiuto di JT. Al completamento di ogni attività della mappa, notifica al JT le partizioni. Ogni riduttore interroga periodicamente JT per conoscere il nodo che esegue le attività della mappa.
Considera un file di input da 2 TB. Cosa succede se la dimensione dell'output del mapper è la stessa dell'input? Lo spostamento di 2 TB per ridurre i nodi sulla rete richiede un'enorme larghezza di banda. Questo è il motivo per cui il combinatore viene eseguito per ridurre al minimo la dimensione dell'output della mappa da spostare attraverso una rete per raggiungere il nodo ridotto. Per ridurre ulteriormente le dimensioni dell'output della mappa, è possibile comprimerla. Snappy è il compressore più comunemente utilizzato in MR.

Passaggio 7: unisci e ordina

Le rispettive partizioni di output della mappa vengono copiate nell'attività di riduzione Java Virtual Machine (JVM). Non appena arrivano le partizioni da tutte le attività della mappa per ridurre i nodi, le partizioni dovrebbero essere unite in un unico file per l'ulteriore elaborazione. Ogni 10 file distribuiti vengono uniti per impostazione predefinita. Il file unito deve essere ordinato in base alla chiave.

Passaggio 8: gruppo

Un insieme di valori che appartengono alla stessa chiave viene raggruppato per eliminare la ridondanza delle chiavi. Il numero di volte in cui viene eseguita la funzione di riduzione è uguale al numero di coppie chiave:elenco(valori) dopo il raggruppamento. Se esistono chiavi duplicate, la funzione di riduzione viene ripetutamente richiamata per ogni chiave duplicata. Se la stessa chiave è presente in più file distribuiti, per una singola chiave è necessario inserire molti file memoria (richiede più passaggi) per alimentare la funzione di riduzione. Ciò comporta molto IO. Questo è il motivo per cui ordiniamo e raggruppiamo i record prima di chiamare la funzione di riduzione. Pertanto, è richiesto un solo passaggio per ciascuna chiave e viene richiamata una sola volta la funzione di riduzione per ciascuna chiave univoca.

Passaggio 9: ridurre l'attività

La funzione Riduci elabora un elenco di valori per ciascuna chiave e produce zero o più record di output. Il numero di volte in cui viene eseguita la funzione di riduzione è uguale al numero di record (chiave: elenco di valori) dopo il raggruppamento. La funzione mappa viene utilizzata principalmente per la pre-elaborazione dei record. Tuttavia, l'algoritmo principale è implementato nella funzione di riduzione. Le operazioni di aggregazione e unione vengono eseguite qui. L'output della mappa viene eliminato dopo il completamento con successo di tutti i riduttori. Se un nodo che esegue l'attività di mappa fallisce prima che l'output della mappa venga acquisito dalle attività di riduzione, il framework eseguirà automaticamente nuovamente l'attività di mappa per creare nuovamente l'output della mappa. JT decide in quale nodo devono essere eseguite le attività di riduzione. Potrebbe essere lo stesso nodo in cui vengono eseguite le attività della mappa o qualche altro nodo nello stesso rack o nodo in qualche altro rack. Dipende dalla disponibilità degli slot e dal carico della rete locale. Gli utenti possono specificare il numero di attività di riduzione in base ai requisiti di parallelismo. Se non specificata, viene avviata un'attività di riduzione predefinita, che utilizza l'output della mappa e i risultati così come sono, ma in modo ordinato mentre passa attraverso le funzioni di shule, sort e group. Se si imposta il numero di attività di riduzione su zero, l'output della mappa stesso viene considerato output del lavoro e verrà archiviato in HDFS. In questo caso non saranno previsti processi di shuffle, ordinamento, unione e raggruppamento.

L'output dell'attività di riduzione viene generalmente archiviato in HDFS per impostazione predefinita con la replica, a differenza dell'output della mappa archiviato in memoria o distribuito nel file system locale. La prima copia viene archiviata localmente dove è in esecuzione l'attività di riduzione, la seconda copia viene archiviata in qualsiasi nodo dello stesso rack e la terza copia viene archiviata in qualsiasi nodo di qualche altro rack nel cluster. Pertanto, la scrittura e la riduzione dell'output consumano la larghezza di banda della rete tanto quanto consuma la normale pipeline di scrittura HDFS.

Passaggio 10: scrittura di record (RW)

La funzione di riduzione fornisce la chiave di output e il valore di output a RW, che a sua volta scrive su HDFS con separazione di tabulazione per impostazione predefinita in base a TextOut-putFormat. Ogni riduttore scrive un file di output su HDFS con la RF

desiderata. RW apre un file di output e scrive i record di output ridotti. L'output delle attività di mappa viene riversato nel file system locale perché la scrittura di risultati intermedi su HDFS porterà a repliche non necessarie e richiederà lavoro aggiuntivo per eliminarli in seguito. Tuttavia, l'output dell'attività di riduzione viene scritto su HDFS poiché è il risultato finale e dovrebbe essere con tolleranza agli errori.

DEMONI CHE ESEGUONO MR STEPS

La Figura 2.21 mostra i demoni che controllano i passaggi nella sequenza di esecuzione di MR. NN gestisce il caricamento e la suddivisione dei big data in blocchi di uguali dimensioni. JT si occupa della gestione del ciclo di vita del lavoro e della formazione dell'IS. La funzione da RR a combinatore viene eseguita nella JVM dell'attività di mappatura. I passaggi (rimescolamento, unione, ordinamento, gruppo) sono gestiti dallo stesso framework di esecuzione MR. I passaggi rimanenti vengono eseguiti riducendo l'attività per terminare l'esecuzione del lavoro. Gli utenti possono definire quasi tutte le funzioni nella sequenza di esecuzione, ma solo le funzioni di mappatura, combinazione, partizionamento e riduzione sono abbastanza comunemente Personalizzate

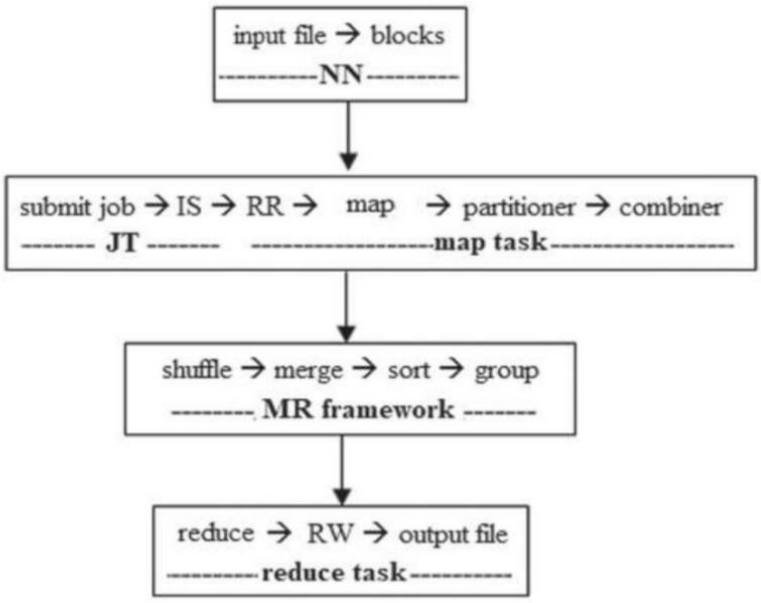
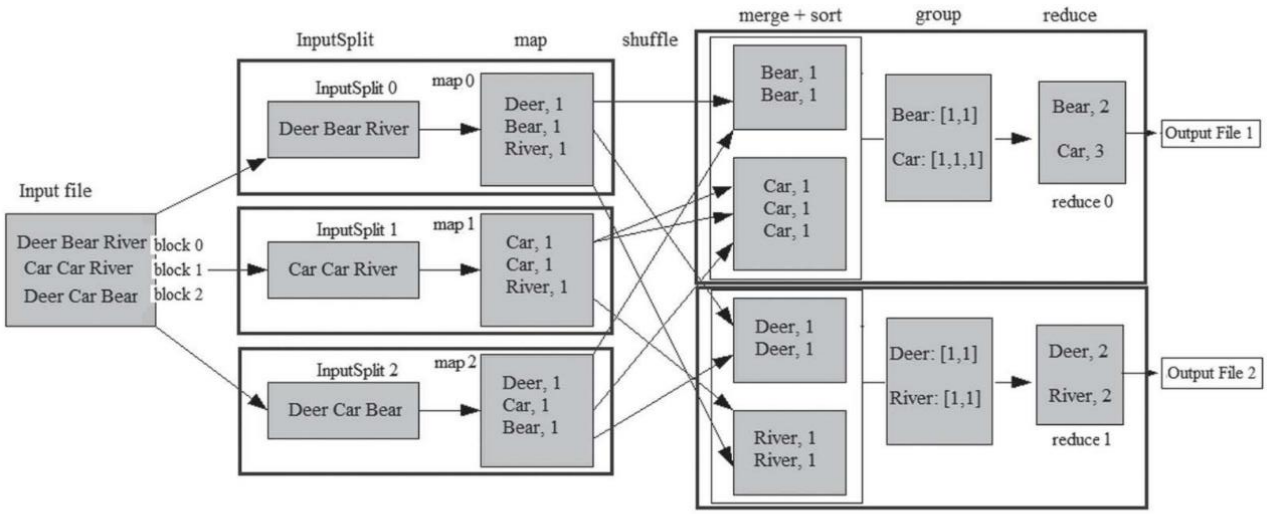


FIGURA 2.21 Demoni che gestiscono le fasi MR.

ESEMPIO

Sulla base della revisione completa della MR, la Figura 2.22 illustra l'esempio del conteggio delle parole. Il nostro obiettivo è ottenere la frequenza totale di ciascuna parola il file di input specificato. Considera un file di input con tre blocchi. Ogni IS punta a un blocco. La funzione mappa tokenizza i record di input e assegna il valore "1" per ogni parola. Con l'aiuto partizionatore, le parole che iniziano con A fino a C vengono inviate al riduttore 0 e il resto delle parole viene inviato al riduttore 1 in fase di shuffle. Le parole raccolte da diversi file vengono unite in un unico file. Quindi, i record vengono ordinati e raggruppati in base alla chiave. Pertanto, per ogni chiave, il valore 1 viene bastonato. Infine, la funzione Riduci riassume l'elenco di "1" assegnato alla singola parola. Ogni attività di riduzione genererà un file di output su HDFS con replica.



FIGURA

REVISIONE DELLA TERMINOLOGIA CORRELATA ALLA RM

NN: gestisce i metadati del file system, controlla e coordina i DN.

DN: gestisce i dischi locali, archivia e recupera blocchi di dati su istruzioni NN.

SNN: conserva una copia dei metadati da NN per evitare SPOF.
JT: prepara il piano di esecuzione, avvia le attività di mappatura/riduzione ed esegue il coordinamento delle fasi.

TT: gestisce CPU e memoria come slot, esegue attività di mappatura/riduzione, tiene traccia

Client HDFS: un demone che riceve la richiesta di caricamento dei dati e interagisce con NN per inserire blocchi di dati. Può essere eseguito in qualsiasi nodo del cluster Hadoop.

Job Client: un demone che riceve il lavoro MR e interagisce con JT. Può essere eseguito in qualsiasi nodo del cluster Hadoop.

Attività di mappatura: esegue una funzione di mappa definita dall'utente, che legge i dati da HDFS come coppie chiave/valore e produce un numero arbitrario di coppie chiave-valore intermedie.

Attività di riduzione: esegue una funzione di riduzione definita dall'utente, che raccoglie l'output da tutte le attività della mappa, unisce, ordina, raggruppa i valori che appartengono alla stessa chiave e infine produce un numero arbitrario di coppie chiave-valore.

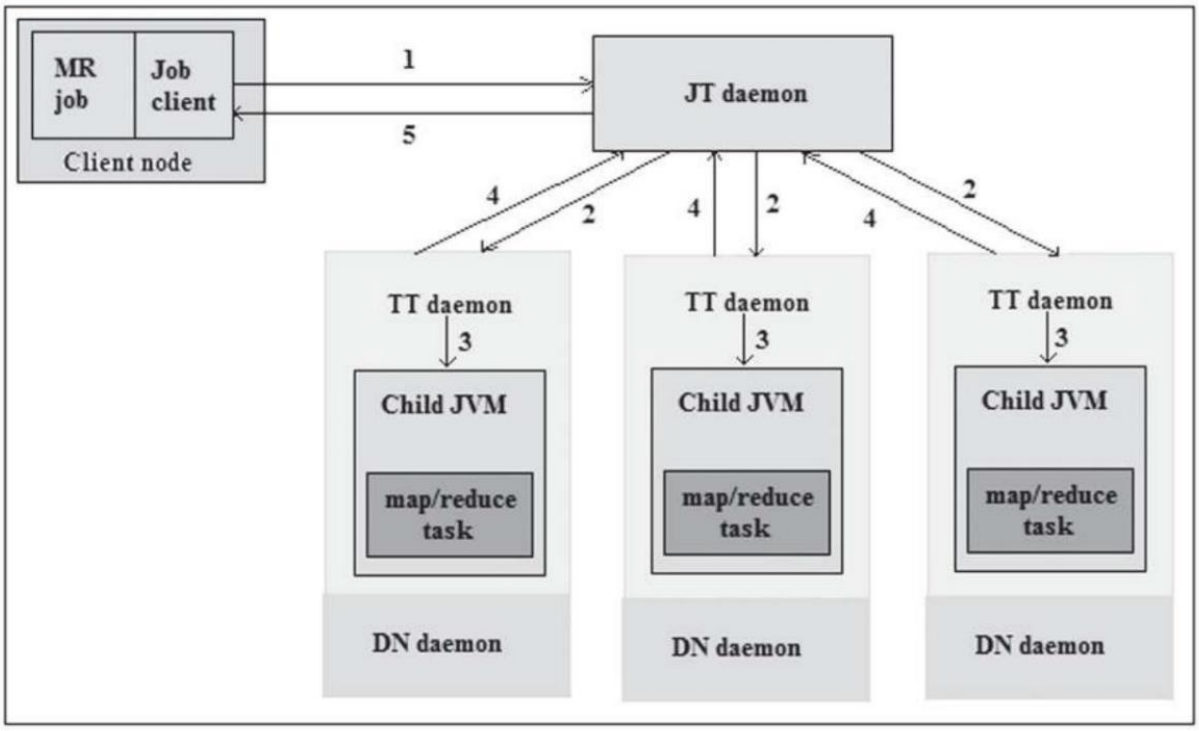


FIGURA 2.24 Flusso di esecuzione del lavoro MR.

1. Un utente invia un lavoro al client Lavoro, che a sua volta invia il lavoro a JT
2. JT prepara il piano di esecuzione, distribuisce i compiti e coordina
3. TT avvia le attività di mappatura/riduzione.
4. TT invia l'avanzamento delle attività di mappatura/riduzione e HB a JT.
5. JT invia al cliente lo stato di avanzamento del lavoro e il messaggio di completamento.

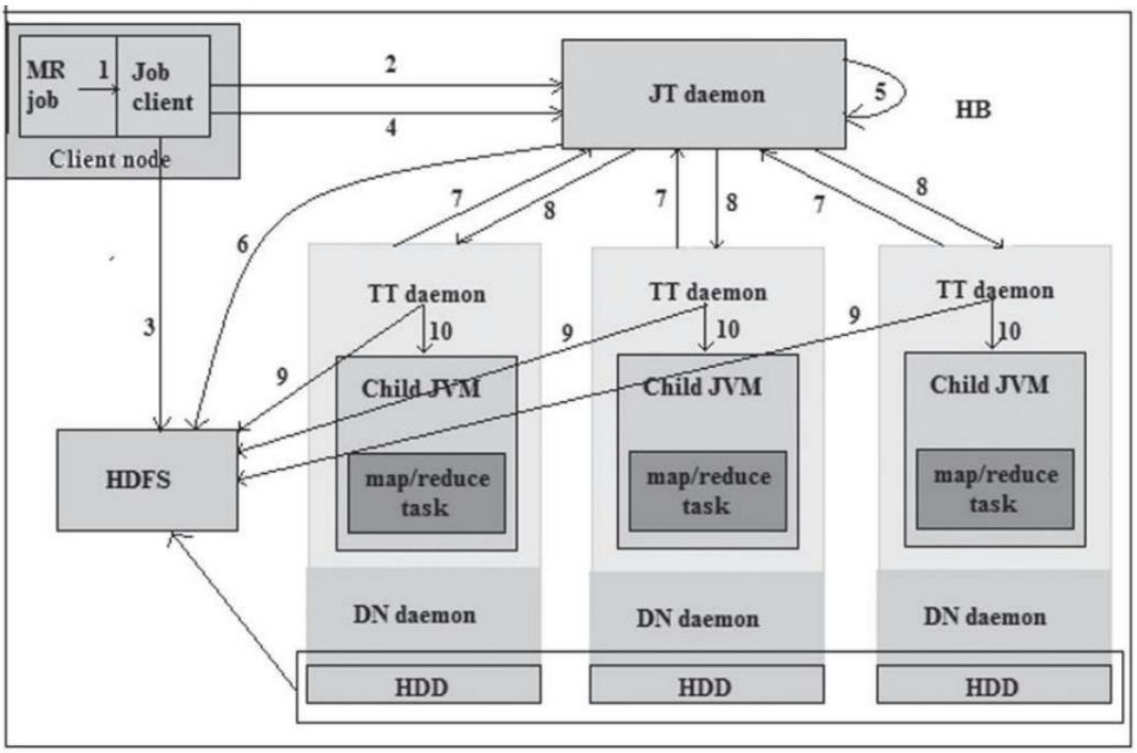


FIGURA 2.25 Flusso dettagliato di esecuzione del lavoro MR.

1 Caricare i dati su HDFS prima di avviare un lavoro MR. Quindi, l'utente invia un lavoro al client lavoro.
2. Il client del lavoro esegue vari controlli, se il file di input e la directory di output in HDFS esistono interagendo con NN. Altrimenti, il client lavoro interagisce con JT per ottenere jobID e calcola IS.
3. Tutte le informazioni relative al lavoro come job.xml, file di lavoro e informazioni IS vengono copiate su 10 DN per impostazione predefinita per essere altamente disponibili durante l'esecuzione.
4. Una volta spostate tutte le informazioni su HDFS, il client del lavoro invia un lavoro a JT per iniziare il ciclo di vita del lavoro e viene creata la directory di output. Tuttavia, i file di output delle attività di riduzione vengono creati solo alla fine del ciclo di vita del lavoro.

Inizializzazione del lavoro

5 Il calcolo della IS per client di lavoro potrebbe essere più lento. Quindi, puoi impostare JT per determinare IS comunicando NN. Una volta fatto ciò, il lavoro viene inserito nella coda dei lavori. Quindi, lo scheduler del lavoro MR preleva il lavoro dalla coda dei lavori e crea un oggetto per incapsulare le sue attività di mappatura/riduzione.
6 JT recupera le informazioni IS da HDFS (se calcolate dal client del lavoro) e prepara un piano di esecuzione basato sulla località dei blocchi di dati.
Ogni blocco ha tre copie per impostazione predefinita. Quindi, JT prepara un elenco (per il blocco 1, ad esempio DN2, DN1, DN3) per ciascun blocco. DN2 è il primo nell'elenco poiché è molto vicino a JT per via della consapevolezza del rack

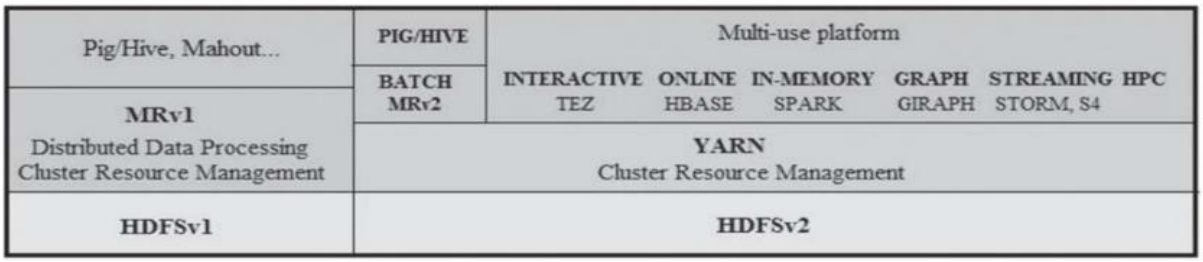
Assegnazione del compito
7. TT esegue un ciclo infinito che invia periodicamente informazioni sull'HB e sullo stato dell'attività a JT.

8. Lo scheduler tenta di avviare un'attività di mappa sul blocco1 in TT2. Se non c'è uno slot libero in TT2, viene tentato TT1. Se non ci sono slot liberi nei tre TT menzionati affinché il blocco 1 raggiunga la località dei dati, il blocco 1 viene copiato in qualsiasi altro TT che abbia una mappa/riduzione degli slot liberi per eseguire l'esecuzione non locale

Esecuzione dell'attività
9. TT localizza le informazioni relative al lavoro come il file jar e il file job.xml da HDFS
10. TT avvia una JVM per eseguire attività di mappatura/riduzione. NN ordina a DN di portare il blocco richiesto da HDFS e caricarlo in JVM.
Una volta completate tutte le attività di mappatura e riduzione, JT aggiorna lo stato del lavoro come riuscito e invia una notifica al client del lavoro. Alla fine, WebUI presenterà tutte le informazioni statistiche e di runtime. Infine, JT ordina a TT di rimuovere JVM e recuperare le risorse in modo sicuro

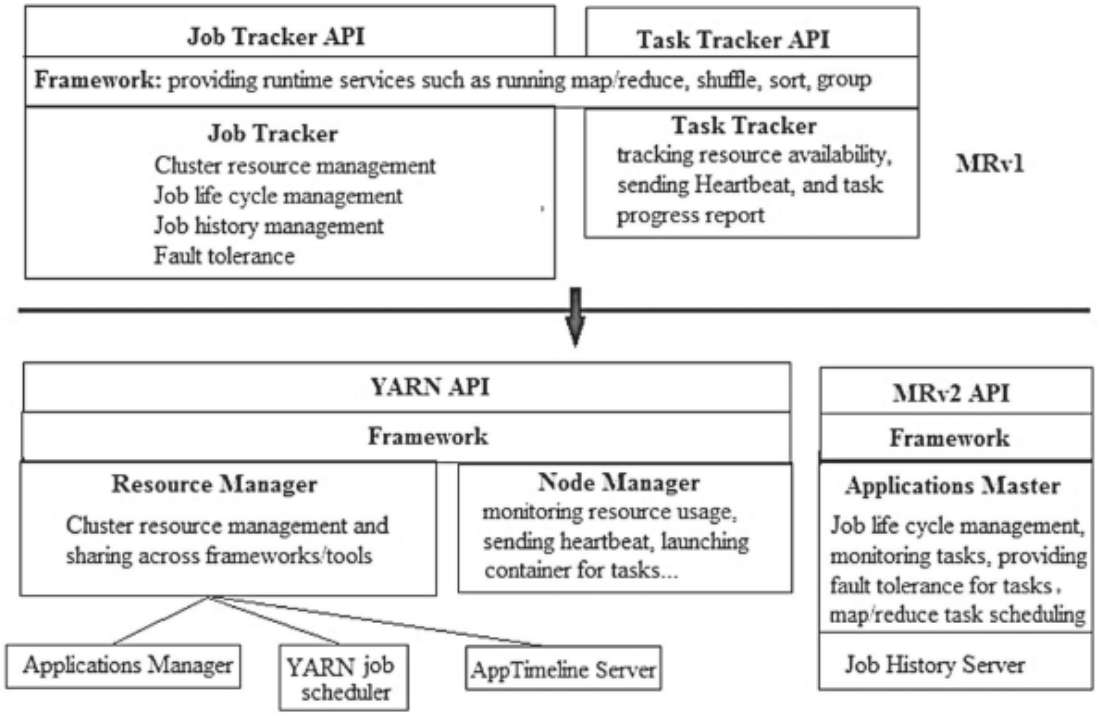
YARN

Se MRv1 è installato in un cluster, non è possibile distribuire nessun altro framework di elaborazione dati. Poiché non condivide le risorse del cluster con altri framework di elaborazione . Pertanto, il framework Hadoop 1.x è sufficiente solo per MapReduce (MR). Hadoop 2.x divide le funzionalità MRv1 in due componenti software (MRv1 e YARN) per sfruttare maggiore scalabilità e condivisione delle risorse tra framework



CARATTERISTICHE DI HDFS V2
HDFSv2 presenta molti vantaggi rispetto a HDFSv1:
NN HA per evitare SPOF, federazione HDFS per bilanciare il carico NN e migliore sicurezza dei dati.
Hadoop 2.x ha introdotto la federazione HDFS, che è un gruppo (federazione) di più NN indipendenti per bilanciare il carico di una NN.
NN HA
• fornire tolleranza agli errori per le attività.
La NN HA include anche più di una NN per superare SPOF. In HDFSv1, per ripristinare NN da un errore, sono necessari più di 30 minuti in un ambiente di produzione poiché comporta un intervento manuale. NN HA fornisce il failover automatico che evita l'intervento manuale per attivare il nuovo funzionamento NN. Questa funzionalità elimina i tempi di inattività (solo pochi minuti) e fa sì che il cluster HDFS rimanga Funzionale.

COMPONENTI IN HADOOP 1.X CONTRO HADOOP 2.X


Job Tracker API
Task Tracker API
Framework: providing runtime services such as running map/reduce, shuffle, sort, group
Job Tracker
Cluster resource management
Job life cycle management
Job history management
Fault tolerance
Task Tracker
tracking resource availability, sending Heartbeat, and task progress report
MRv1
YARN API
Framework
Resource Manager
Cluster resource management and sharing across frameworks/tools
Node Manager
monitoring resource usage, sending heartbeat, launching container for tasks...
Applications Manager
YARN job scheduler
AppTimeline Server
MRv2 API
Framework
Applications Master
Job life cycle management, monitoring tasks, providing fault tolerance for tasks, map/reduce task scheduling
Job History Server