Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica

Big Data

Tecniche e tool di analisi

Elaborato Finale

Laureando

Emanuele Gentiletti

Matricola: **090150**

Relatore

Prof. Diletta Romana Cacciagrano

Anno Accademico 2016/2017

Indice

Introduzione

Negli ultimi decenni, i Big Data hanno preso piede in modo impetuoso in una grande varietà di ambiti. Il fenomeno ha avuto un enorme impatto: settori come medicina, finanza, business analytics e marketing sfruttano i Big Data per guidare lo sviluppo in modi semplicemente non possibili prima.

L'innovazione che rende possibili questi risultati è guidata dal software molto più che dall'hardware. Ci sono stati dei grandi cambiamenti nel modo di pensare alla computazione e all'organizzazione dei suoi processi, che hanno portato a risultati notevoli nell'efficienza di elaborazione di grandi quantità di dati.

Uno dei più importanti fenomeni che hanno portato a questa spinta è stato lo sviluppo di Hadoop, un framework open source progettato per la computazione batch di dataset di grandi dimensioni. Utilizzando un'architettura ben congeniata, Hadoop ha permesso l'analisi in tempi molto rapidi di interi dataset di dimensioni nell'ordine dei terabyte, fornendo una capacità di sfruttamento di questi, e conseguentemente un valore molto più alti.

Una delle conseguenze più importanti di Hadoop è stata una democratizzazione delle capacità di analisi dei dati:

- Hadoop è sotto licenza Apache, permettendo a chiunque di utilizzarlo a scopi commerciali e non;
- Hadoop non richiede hardware costoso ad alta affidabilità, e incoraggia l'adozione di macchine più generiche e prone al fallimento per il suo uso, che possono essere ottenute a costi inferiori;
- Il design di Hadoop permette la sua esecuzione in cluster di macchine eterogenee nel software e nell'hardware che possono essere acquisite da diversi rivenditori, un altro fattore che permette l'abbattimento dei costi;
- I vari modelli di programmazione in Hadoop hanno in comune l'astrazione della computazione distribuita e dei problemi intricati che questa comporta, abbassando la barriere in entrata in termini di conoscenze e lavoro richiesti per creare programmi che necessitano di un altro grado di parallelismo.

Questi fattori hanno spinto a una vasta adozione di Hadoop e dell'ecosistema software che lo circonda, in ambito aziendale e scientifico. L'adozione di Hadoop, secondo un sondaggio fatto a maggio 2015[1], si aggira al 26% delle imprese negli Stati Uniti, e si prevede che il mercato attorno ad Hadoop sorpasserà i 16 miliardi di dollari nel 2020 [2].

Tutto questo accade in un'ottica in cui la produzione di informazioni aumenta ad una scala senza precedenti: secondo uno studio di IDC[3], la quantità di informazioni nell'"Universo Digitale" ammontava a 4.4 TB nel 2014, e la sua dimensione stimata nel 2020 è di 44 TB.

Data la presenza di questa vasta quantità di informazioni, lo sfruttamento efficace di queste è fonte di grandi opportunità. In questo documento si analizzano le varie tecniche che sono a disposizione per l'utilizzo effettivo dei Big Data, come queste differiscono tra di loro, e quali strumenti le mettono a disposizione. Si parlerà inoltre di come gli strumenti possano essere integrati in sistemi di produzione esistenti, le possibili architetture di un sistema di questo tipo e come …

La gestione di sistemi per l'elaborazione di Big Data richiede una configurazione accurata per ottenere affidabilità e fault-tolerance. Pur sottolineando che l'importanza di questi aspetti non è da sottovalutare, questa tesi si concentrerà più sul modello computazionale e di programmazione che gli strumenti offrono.

Parte 1

Big Data e Paradigmi di Elaborazione

Per Big Data si intendono collezioni di dati con caratteristiche tali da richiedere strumenti innovativi per poterli gestire e analizzare. Uno dei modelli tradizionali e più popolari per descrivere le caratteristiche dei Big Data si chiama **modello delle 3V**. Il modello identifica i Big Data come collezioni di informazione che presentano grande abbondanza in una più delle seguenti caratteristiche:

- Il **volume** delle informazioni, che può aggirarsi dalle decine di terabyte per arrivare all'ordine dei petabyte;
- La **varietà**, intesa come la varietà di *fonti* e di *possibili strutturazioni* delle informazioni di interesse;
- La **velocità** di produzione delle informazioni di interesse.

Ognuno dei punti di questo modello deriva da esigenze che vanno ad accentuarsi andando avanti nel tempo, in particolare:

- Il volume delle collezioni dei dati è aumentato esponenzialmente in tempi recenti, con l'avvento dei Social Media, dell'IOT, e degli smartphone muniti di molti sensori diversi. Generalizzando, i fattori che hanno portato a un grande incremento del volume dei data set sono un aumento della generazione automatica di dati da parte dei dispositivi e dei contenuti prodotti dagli utenti.
- L'aumento dei dispositivi e dei dati generati dagli utenti portano conseguentemente a un aumento delle fonti dei dati, ed essendo queste gestite da enti e persone diverse la struttura che le fonti presentano difficilmente sarà uniforme, l'una rispetto all'altra. Inoltre, l'utilizzo di dati non strutturati rigidamente è prevalente nelle tecnologie web (in particolare documenti JSON), che sono spesso un obiettivo desiderabile di analisi.
- Si possono fare le stesse considerazioni fatte per il volume dei dati per quanto riguarda la velocità. I flussi di dati vengono generati dai dispositivi e dagli utenti, che li producono a velocità molto maggiori rispetto agli operatori.

Per l'elaborazione di dataset con queste caratteristiche sono stati sviluppati molti strumenti, che usano diversi pattern di elaborazione a seconda delle esigenze dell'utente e del tipo di dati con cui si ha a che fare. I modelli di elaborazione più importanti e rappresentativi sono il *batch processing* e lo *stream processing*.

Batch e Streaming Processing

Il batch processing è il pattern di elaborazione generalmente più efficiente, e consiste nell'elaborazione di un intero dataset in un'unità di lavoro, per poi ottenere i risultati al termine di questa.

Questo approccio è ottimale quando non c'è una necessità impendente di avere risultati, ma in alcuni casi è necessario avere i risultati a disposizione mano a mano che la computazione procede, e il batch processing non è adatto a questo scopo:

- Le fasi del batch processing richiedono la schedulazione dei lavori da parte dell'utente, con un conseguente overhead dovuto alla schedulazione in sé o alla configurazione di strumenti automatizzati che se ne occupino;

- Non è possibile accedere ai risultati prima del termine del job, che può avere una durata eccessiva rispetto alle esigenze dell'applicazione o dell'utente.

Per use case in cui questi fattori sono rilevanti, lo **stream processing** si presta come più adatto. In questo paradigma, i dati da elaborare vengono ricevuti da stream (nella maggior parte dei casi da Internet) e vengono processati mano a mano con il loro arrivo. I job in streaming molto spesso non hanno un termine prestabilito, ma vengono terminati dall'utente, e i risultati dell'elaborazione possono essere disponibili mano a mano che l'elaborazione procede, permettendo quindi un feedback più rapido rispetto ai lavori batch.

Data at Rest e *Data in Motion*

I due paradigmi si differenziano anche per il modo in cui i dati sono disponibili. Il processing batch richiede che l'informazione sia *data at rest*, ovvero informazioni salvate interamente in un mezzo di memoria accessibile al programma. I dati di input in una computazione batch sono determinati all'inizio dell'elaborazione, e non possono cambiare nel corso di questa. Questo significa che se nuova informazione arriva nel corso di un job batch, questa non può essere tenuta in conto nell'elaborazione finale.

Lo **stream processing**, invece, è progettato per *data in motion*, dati in arrivo continuo la cui quantità non è fissa a priori. È possibile utilizzare strumenti di processing in streaming anche per *data at rest*, rappresentando il dataset come uno stream. Questa proprietà è desiderabile, perché permette di utilizzare le stesse applicazioni per l'elaborazione di dati in arrivo dalla rete e quelli salvati. Come si vedrà, la Kappa architecture, ovvero una possibile architettura software per l'elaborazione di Big Data, utilizza questa proprietà per sfruttare un solo paradigma sia per computazioni in real-time di dati in arrivo da stream, che per i dati salvati storicamente, permettendo di utilizzare gli stessi tool e interfacce di programmazione per entrambi i tipi di elaborazione e massimizzare il riutilizzo di codice.

Tabella 1.1: Differenze tra elaborazione batch e streaming

Caratteristiche	Batch	Streaming
Ottimizzazione	Alto throughput	Bassa latenza
Tipo di informazione	*Data at rest*	*Data in motion* e *Data at rest*
Accesso ai dati	Stabilito all'inizio	Dipendente dallo stream
Accesso ai risultati	Fine job	Continuo

Un esempio di *data at rest* sono i resoconti delle vendite di un'azienda, su cui si possono cercare pattern per identificare quali prodotti sono in trend nelle vendite. Per *data in motion* si può considerare l'invio di dati da parte di sensori IoT o le pubblicazioni degli utenti nei social media, che sono continui e senza una fine determinata.

Hadoop e modelli di elaborazione

Nella prossima sezione si discuterà di Hadoop, un progetto nato con l'intento di affrontare la computazione batch

Parte 2

Hadoop

Nell'ambito dei Big Data, Hadoop è il perno centrale su cui è basato un *ecosistema* di tool e tecnologie, tant'è che spesso il termine Hadoop viene utilizzato per riferirsi all'intero ecosistema di tool e tecnologie construiti attorno a questo.

La documentazione ufficiale[4] lo descrive come:

> …un framework che abilita l'elaborazione distribuita di grandi dataset in cluster di computer utilizzando semplici modelli di programmazione. Hadoop è progettato per essere scalato da server singoli a migliaia di macchine, dove ognuna di queste offre computazione e storage locale. Invece di affidarsi all'hardware per fornire un'alta affidabilità, Hadoop è progettato per rilevare e gestire i fallimenti [delle computazioni] a livello applicativo, mettendo a disposizione un servizio ad alta affidiabilità su cluster di computer proni al fallimento.

In questa definizione sono racchiusi dei punti molti importanti:

- **Semplici modelli di programmazione**

  Hadoop raggiunge molti dei suoi obiettivi fornendo un'interfaccia di livello molto alto al programmatore, in modo di potersi assumere la responsabilità di concetti complessi e necessari all'efficienza nella computazione distribuita, ma che hanno poco a che fare con il problema da risolvere in sé (ad esempio, la sincronizzazione di task paralleli e lo scambio dei dati tra nodi del sistema distribuito). Questo modello **pone dei limiti alla libertà del programmatore**, che deve adeguare la codifica della risoluzione del problema al modello di programmazione fornito.

- **Computazione e storage locale**

  L'ottimizzazione più importante che Hadoop fornisce rispetto all'elaborazione dei dati è il risultato dell'unione di due concetti: **distribuzione dello storage** e **distribuzione della computazione**.

  Entrambi sono importanti a prescindere dell'uso particolare che ne fa Hadoop: la distribuzione dello storage permette di combinare lo spazio fornito da più dispositivi e di farne uso tramite un'unica interfaccia logica, e di replicare i dati in modo da poter tollerare guasti nei dispositivi. La distribuzione della computazione permette di aumentare il grado di parallelizzazione nell'esecuzione dei programmi.

  Hadoop unisce i due concetti utilizzando cluster di macchine che hanno sia lo scopo di mantenere lo storage, che quello di elaborare i dati. Quando Hadoop esegue un lavoro, **quante più possibili delle computazioni richieste vengono eseguite nei nodi che contengono i dati da elaborare**. Questo permette di ridurre la latenza di rete, minimizzando la quantità di dati che devono essere scambiati tra i nodi del cluster. Il meccanismo è trasparente all'utente, a cui basta persitere i dati da elaborare nel cluster per usifruirne. Questo principio viene definito **data locality**.

- **Scalabilità**

  |||

- **Hardware non necessariamente affidabile**

I cluster di macchine che eseguono Hadoop non hanno particolari requisiti di affidabilità rispetto ad hardware consumer. Il framework è progettato per tenere in conto dell'alta probabilità di fallimento dell'hardware, e per attenuarne le conseguenze, sia dal punto di vista dello storage e della potenziale perdita di dati, che da quello della perdita di risultati intermedi e parziali nel corso dell'esecuzione di lavori computazionalmente costosi. In questo modo l'utente è sgravato dal compito generalmente difficile di gestire fallimenti parziali nel corso delle computazioni.

Hadoop è composto da diversi moduli:

- **HDFS**, un filesystem distribuito ad alta affidabilità, che fornisce replicazione automatica all'interno dei cluster e accesso ad alto throughput ai dati
- **YARN**, un framework per la schedulazione di lavori e per la gestione delle risorse all'interno del cluster
- **MapReduce**, un framework e un modello di programmazione fornito da Hadoop per la scrittura di programmi paralleli che processano grandi dataset.

HDFS

HDFS è un filesystem distribuito che permette l'accesso ad alto throughput ai dati. HDFS è scritto in Java, e viene eseguito nello userspace. Lo storage dei dati passa per il filesystem del sistema che lo esegue.

I dati contenuti in HDFS sono organizzati in unità logiche chiamate *blocchi*, come è comune nei filesystem. I blocchi di un singolo file possono essere distribuiti all'interno di più macchine all'interno del cluster, permettendo di avere file più grandi della capacità di storage di ogni singola macchina nel cluster. Rispetto ai filesystem comuni la dimensione di un blocco è molto più grande, 128 MB di default. La ragione per cui HDFS utilizza blocchi così grandi è minimizzare il costo delle operazioni di seek, dato il fatto che se i file sono composti da meno blocchi, si rende necessario trovare l'inizio di un blocco un minor numero di volte. Questo approccio riduce anche la frammentazione dei dati, rendendo più probabile che questi vengano scritti contiguamente all'interno della macchina[1].

Il blocco, inoltre, è un'astrazione che si presta bene alla replicazione dei dati nel filesystem all'interno del cluster: per replicare i dati, come si vedrà, si mette uno stesso blocco all'interno di più macchine nel cluster.

HDFS è basato sulla specifica POSIX, ma non la implementa in modo rigido: tralasciare alcuni requisiti di conformità alla specifica permette ad HDFS di ottenere prestazioni e affidabilità migliori, come verrà descritto in seguito.

Principi architetturali

La documentazione di Hadoop descrive i seguenti come i principi architetturali alla base della progettazione di HDFS:

- **Fallimento hardware come regola invece che come eccezione**

  Un sistema che esegue HDFS è composto da molti componenti, con probabilità di fallimento non triviale. Sulla base di questo principio, HDFS da' per scontato che **ci sia sempre un numero di componenti non funzionanti**, e si pone di rilevare errori e guasti e di fornire un recupero rapido e automatico da questi.

  Il meccanismo principale con cui HDFS raggiunge questo obiettivo è la replicazione: in un cluster, ogni blocco di cui un file è composto è replicato in più macchine (3 di default). Se un blocco non è disponibile in una macchina, o se non supera i controlli di integrità, una sua copia può essere letta da un'altra macchina in modo trasparente per il client.

  Il numero di repliche per ogni blocco è configurabile, e ci sono più criteri con cui viene deciso in quali macchine il blocco viene replicato, principalmente orientati al risparmio di banda di rete.

[1] Non è possibile essere certi della contiguità dei dati, perché HDFS non è un'astrazione diretta sulla scrittura del disco, ma sul filesystem del sistema operativo che lo esegue. Per cui la frammentazione effettiva dipende da come i dati vengono organizzati dal filesystem sottostante.

- **Modello di coerenza semplice**

  Per semplificare l'architettura generale, HDFS fa delle assunzioni specifiche sul tipo di dati che vengono salvati in HDFS e pone dei limiti su come l'utente possa lavorare sui file. In particolare, **non è possibile modificare arbitrariamente file già esistenti**, e le modifiche devono limitarsi a operazioni di troncamento e di aggiunta a fine file. Queste supposizioni permettono di semplificare il modello di coerenza, perché i blocchi di dati, una volta scritti, possono essere considerati immutabili, evitando una considerevole quantità di problemi in un ambiente dove i blocchi di dati sono replicati in più posti:

  – Per ogni modifica a un blocco di dati, bisognerebbe verificare quali altre macchine contengono il blocco, e rieseguire la modifica (o rireplicare il blocco modificato) in ognuna di queste.

  – Queste modifiche dovrebbero essere fatte in modo atomico, o richieste di lettura su una determinata replica di un blocco invece che in un'altra potrebbe portare a risultati inconsistenti o non aggiornati.

  Le limitazioni che Hadoop impone sono ragionevoli per lo use-case per cui HDFS è progettato, caratterizzato da grandi dataset che vengono copiati nel filesystem e letti in blocco.

- **Dataset di grandi dimensioni**

  I filesystem distribuiti sono generalmente necessari per aumentare la capacità di storage disponibile oltre quella di una singola macchina. La distribuzione di HDFS, assieme alla grande dimensione dei blocchi

- **Accesso in streaming**

  HDFS predilige l'accesso ai dati in streaming, per permettere ai lavori batch di essere eseguiti con grande efficienza. Questo approccio va a discapito del tempo di latenza della lettura dei file, ma permette di avere un throughput in lettura molto vicino ai tempi di lettura del disco.

- **Portabilità su piattaforme software e hardware eterogenee**

  HDFS è scritto in Java, ed è portabile in tutti i sistemi che ne supportano il runtime.

L'architettura di HDFS è di tipo master/slave, dove un nodo centrale, chiamato **NameNode**, gestisce i metadati e la struttura del filesystem, mentre i nodi slave, chiamati **DataNode**, contengono i blocchi di cui file sono composti. Tipicamente, viene eseguita un'istanza del software del DataNode per macchina del cluster, e una macchina dedicata esegue il NameNode.

I *client* del filesystem interagiscono sia con il NameNode che con i DataNode per l'accesso ai file. La comunicazione tra il client e i nodi avviene tramite socket TCP ed è coordinata dal NameNode, che fornisce ai client tutte le informazioni sul filesystem e su quali nodi contengono i DataBlock dei file richiesti.

Comunicare con HDFS

Hadoop fornisce tool e librerie che possono agire da client nei confronti di HDFS. Il tool più diretto è la CLI, accessibile nelle macchine in cui è installato Hadoop tramite il comando `hadoop fs`.

```
% hadoop fs -help
Usage: hadoop fs [generic options]
    [-appendToFile <localsrc> ... <dst>]
    [-cat [-ignoreCrc] <src> ...]
    [-checksum <src> ...]
    [-chgrp [-R] GROUP PATH...]
    [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
    [-chown [-R] [OWNER][:[GROUP]] PATH...]
    [-copyFromLocal [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
    [-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] <path> ...]
    [-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]
...
```

La CLI fornisce alcuni comandi comuni nei sistemi POSIX, come `cp`, `rm`, `mv`, `ls` e `chown`, e altri che riguardano specificamente HDFS, come `copyFromLocal` e `copyToLocal`, utili a trasferire dati tra la macchina su cui si opera e il filesystem.

I comandi richiedono l'URI che identifica l'entità su cui si vuole operare. Per riferirsi a una risorsa all'interno di un'istanza di HDFS, si usa l'URI del namenode, con schema `hdfs`[2], e con il path corrispondente al percorso della risorsa nel filesystem. Ad esempio, è possibile creare una cartella `foo` all'interno della radice del filesystem con il seguente comando:

```
hadoop fs -mkdir hdfs://localhost:8020/foo
```

Per diminuire la verbosità dei comandi, Hadoop può essere configurato per riferirsi a un filesystem di default quando riceve comandi che usano un URI relativo, accorciando l'esempio precedente a:

```
hadoop fs -mkdir foo
```

Ad esempio, data la seguente cartella:

```
[root@sandbox example_data]# ls
example1.txt  example2.txt  example3.txt
```

Si possono copiare i file dalla cartella locale della macchina al filesystem distribuito con il seguente comando:

```
[root@sandbox example_data]# hadoop fs -copyFromLocal example*.txt /example
```

Per verificare che l'operazione sia andata a buon fine, si può ottenere un listing della cartella in cui si sono trasferiti i file con il comando `ls`:

```
[root@sandbox example_data]# hadoop fs -ls /example
Found 3 items
-rw-r--r--   1 root hdfs         70 2017-06-30 03:58 /example/example1.txt
-rw-r--r--   1 root hdfs         39 2017-06-30 03:58 /example/example2.txt
-rw-r--r--   1 root hdfs         43 2017-06-30 03:58 /example/example3.txt
```

Il listing è molto simile a quello ottenibile su sistemi Unix. Una differenza importante è la seconda colonna, che non mostra il numero di hard link al file nel filesystem[3], ma il numero di repliche che HDFS ha a disposizione del file, in questo caso una per file. Il numero di repliche fatte da HDFS può essere impostato settando il fattore di replicazione di default, che per Hadoop in modalità distribuita è 3 di default. Si può anche cambiare il numero di repliche disponibili per determinati file, utilizzando il comando `hdfs dfs`:

```
[root@sandbox ~]# hdfs dfs -setrep 2 /example/example1.txt
Replication 2 set: /example/example1.txt
[root@sandbox ~]# hadoop fs -ls /example
Found 3 items
-rw-r--r--   2 root hdfs         70 2017-06-30 03:58 /example/example1.txt
-rw-r--r--   1 root hdfs         39 2017-06-30 03:58 /example/example2.txt
-rw-r--r--   1 root hdfs         43 2017-06-30 03:58 /example/example3.txt
```

Alcuni tool di amministrazione di cluster Hadoop offrono GUI web con cui è possibile interfacciarsi in HDFS. Alcuni esempi sono Cloudera Manager e Apache Ambari, che offrono un file manager lato web con cui è possibile interagire in modo più semplice, permettendo anche a utenti meno esperti nel campo di lavorare con il filesystem.

NameNode

Il NameNode è il riferimento centrale per i metadati del filesystem nel cluster, il che vuol dire che se il NameNode non è disponibile il filesystem non è accessibile. Questo rende il NameNode un *single point of failure* del sistema, e per questa ragione HDFS mette a disposizione dei meccanismi per attenuare l'indisponibilità del sistema in caso

[2] Hadoop è abbastanza generale da poter lavorare con diversi filesystem, con lo schema definisce il protocollo di comunicazione, che non deve essere necessariamente `hdfs`. Ad esempio, un URI con schema `file` si riferisce al filesystem locale, e le operazioni eseguite su URI che utilizzano questo schema vengono effettuate sulla macchina dove viene eseguito il comando. Questo approccio può essere adatto nella fase di testing dei programmi, ma nella maggior parte dei casi è comunque desiderabile lavorare su un filesystem distribuito adeguato alla gestione dei Big Data, e un'alternativa ad HDFS degna di nota è MapR-FS[5].

[3] HDFS correntemente non supporta link nel filesystem.

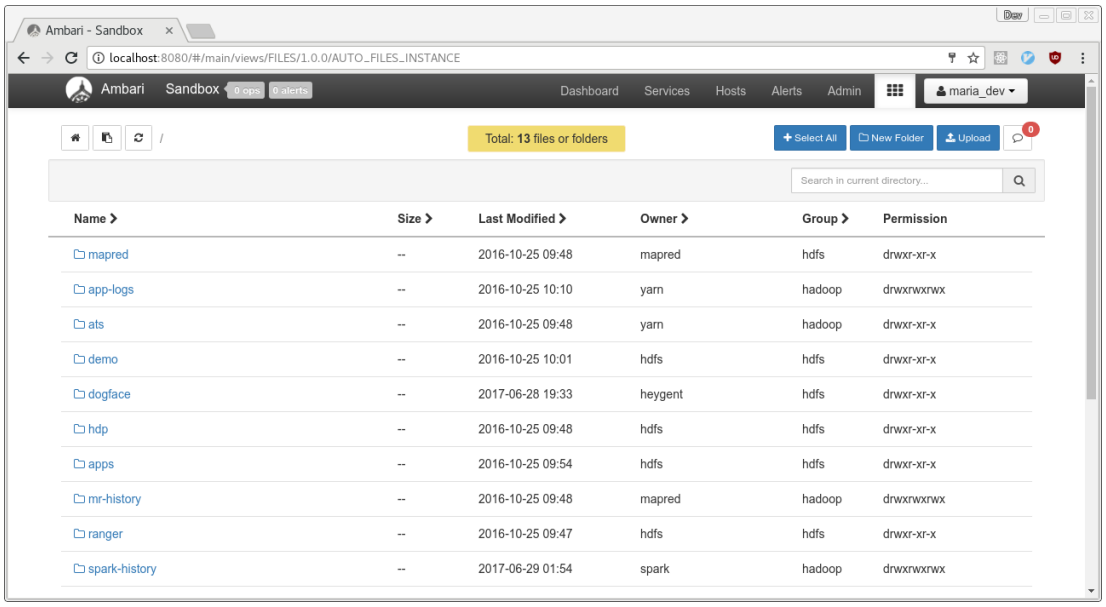

Figura 2.1: Screenshot del file manager HDFS incluso in Ambari

di non reperibilità del NameNode, e per assicurare che lo stato del filesystem possa essere recuperato a partire dal NameNode.

Il NameNode è anche il nodo a cui i client si connettono alla lettura del file. La connessione ha il solo scopo di fornire le informazioni sui DataNode che contengono i dati effettivi del file. I dati di un file non passano mai per il NameNode.

Tuttavia, il NameNode non salva persistentemente le informazioni sulle posizioni dei blocchi, che vengono invece mantenute dai DataNode. Perché il NameNode possa avere in memoria le informazioni sui file necessarie per essere operativo, questo deve ricevere le liste dei blocchi in possesso dei DataNode, in messaggi chiamati **block report**. Non è necessario che il DataNode conosca la posizione di tutti i blocchi sin dall'inizio, ma basta che per ogni blocco conosca la posizione di un numero minimo di repliche, determinato da un'opzione chiamata `dfs.replication.min.replicas`, di default 1.

Questa procedura avviene quando il NameNode si trova in uno stato chiamato **safe mode**.

Namespace image ed *edit log*

Le informazioni sui metadati del sistema vengono salvate nello storage del NameNode in due posti, la ***namespace image*** e l'***edit log***. La *namespace image* è uno snapshot dell'intera struttura del filesystem, mentre l'*edit log* è un elenco di operazioni eseguite nel filesystem a partire dalla *namespace image*. Partendo dalla *namespace image* e applicando le operazioni registrate nell'*edit log*, è possibile risalire allo stato attuale del filesystem. Il NameNode ha una rappresentazione dello stato del filesystem anche nella memoria centrale, che viene utilizzata per servire le richieste di lettura.

Quando HDFS riceve una richiesta che richiede la modifica dei metadati, il NameNode esegue le seguenti operazioni:

1. registra la transazione nell'*edit log*
2. aggiorna la rappresentazione del filesystem in memoria
3. passa all'operazione successiva.

La ragione per cui i cambiamenti dei metadati vengono registrati nell'*edit log* invece che nella *namespace image* è la velocità di scrittura: scrivere ogni cambiamento del filesystem mano a mano che avviene nell'immagine sarebbe lento, dato che questa può avere dimensioni nell'ordine dei gigabyte. Il NameNode esegue un *merge* dell'*edit log* e della *namespace image* a ogni suo avvio, portando lo stato attuale dell'immagine al pari di quello del filesystem.

Dato che la dimensione dell'*edit log* può diventare notevole, è utile eseguire l'operazione di *merge* al raggiungimento di una soglia di dimensione del log. Questa operazione è computazionalmente costosa, e se fosse eseguita dal NameNode potrebbe interferire con la sua operazione di routine.

Per evitare interruzioni nel NameNode, il compito di eseguire periodicamente il *merge* dell'*edit log* è affidato a un'altra entità, il **Secondary NameNode**. Il Secondary NameNode viene solitamente eseguito su una macchina differente, dato che richiede un'unità di elaborazione potente e almeno la stessa memoria del NameNode per eseguire l'operazione di merge.

Avvio del NameNode e *Safe Mode*

Prima di essere operativo, il NameNode deve eseguire alcune operazioni di startup, tra cui attendere di aver ricevuto i block report dai DataNode in modo da conoscere le posizioni dei blocchi. Durante queste operazioni, il NameNode si trova in uno stato chiamato *safe mode*, in cui sono permesse unicamente operazioni che accedono ai metadati del filesystem, e tentativi di lettura e scrittura di file falliscono. Prima di poter permettere l'accesso completo, il NameNode ha bisogno di ricevere le informazioni sui blocchi da parte dei DataNode.

Per ricapitolare, al suo avvio, il NameNode effettua il merge della *namespace image* con l'*edit log*. Al termine dell'operazione, il risultato del merge viene salvato come la nuova *namespace image*. Il Secondary NameNode non viene coinvolto in questo primo merge.

Prima di uscire dalla safe mode, il NameNode attende di avere abbastanza informazioni da poter accedere a un numero minimo di repliche di ogni blocco. A questo punto il NameNode esce dalla safe mode.

Si possono utilizzare dei comandi per verificare lo stato, attivare e disattivare la safe mode.

```
bash-4.1$ hdfs dfsadmin -safemode get
Safe mode is OFF
bash-4.1$ hdfs dfsadmin -safemode enter
Safe mode is ON
bash-4.1$ hdfs dfsadmin -safemode leave
Safe mode is OFF
```

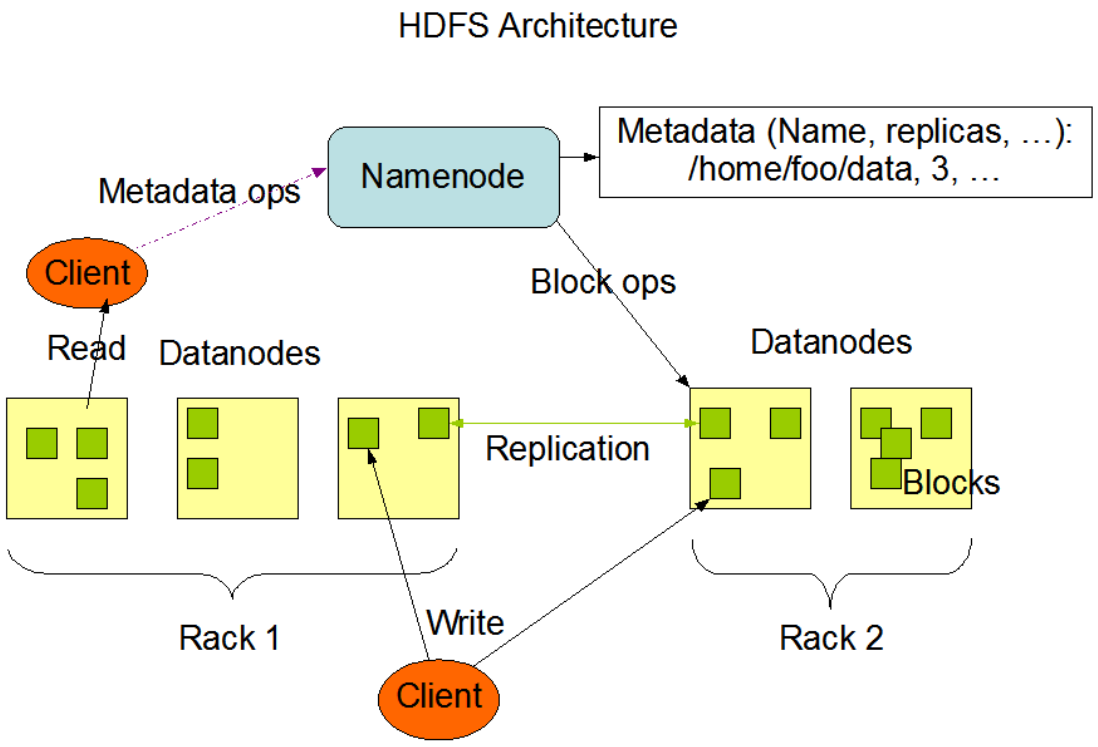

Figura 2.2: Schema di funzionamento dell'architettura di HDFS

MapReduce

Per dare un'idea concreta di come funziona la programmazione in un cluster Hadoop

MapReduce è il primo importante modello di programmazione a cui Hadoop fa riferimento per l'esecuzione di applicazioni distribuite. Hadoop è stato scritto e pensato per l'esecuzione di lavori MapReduce, e nelle prime versioni era il solo modello di programmazione disponibile.

I lavori MapReduce

Bibliografia

1. Gartner Survey Highlights Challenges to Hadoop Adoption, http://www.gartner.com/newsroom/id/3051717

2. Hadoop Market Forecast 2017-2022, https://www.marketanalysis.com/?p=279

3. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

4. Apache Hadoop Documentation, https://hadoop.apache.org/

5. MapR-FS Overview, https://mapr.com/products/mapr-fs/