



UNIVERSITÀ
DEGLI STUDI
DI PALERMO



Introduzione a HDFS

CORSO DI BIG DATA
a.a. 2019/2020

Prof. Roberto Pirrone

Sommario

- Caratteristiche di HDFS
- Architettura
- Operazioni di read/write
- Caratteristiche avanzate

Caratteristiche di HDFS

- *Hadoop Distributed File System* (HDFS) è un filesystem distribuito che si basa sulla filosofia di conservare «pochi grandi frammenti» di un file nell'infrastruttura, piuttosto che molti frammenti piccoli
 - Fault tolerance
 - Elaborazione ad elevato throughput
 - Grandi quantità di dati
 - Write-once-read-many
 - Computazione vicino ai dati
 - Portabilità

Caratteristiche di HDFS

- *Hadoop Distributed File System* (HDFS)
 - Fault tolerance
 - In genere un'istanza HDFS gira su migliaia di nodi e ci sono meccanismi interni di anomaly detection
 - Elaborazione ad elevato throughput
 - Pensato per una elaborazione batch, compensa la latenza non bassa con l'elevato throughput per elaborazione di tipo streaming

Caratteristiche di HDFS

- *Hadoop Distributed File System* (HDFS)
 - Grandi quantità di dati
 - La scalabilità dei nodi, la dimensione elevata dei blocchi e l'elevato throughput consentono la gestione di data set nell'ordine dei terabytes
 - Write-once-read-many
 - Tendenzialmente un file viene creato e scritto e poi non modificato
 - Si adatta al modello MapReduce: il *mapper* scrive e il *reducer* legge

Caratteristiche di HDFS

- *Hadoop Distributed File System* (HDFS)

- Computazione vicino ai dati

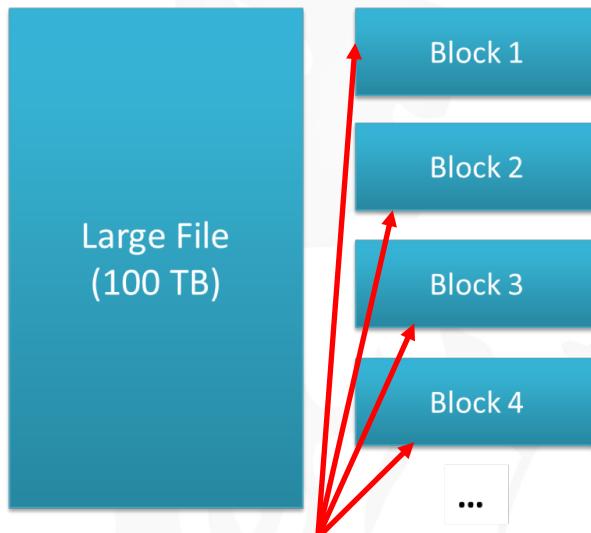
- È più semplice muovere la computazione vicino ai dati e non viceversa
- La banda utilizzata si riduce

- Portabilità

- Scritto appositamente con questo scopo
- Basato su Java

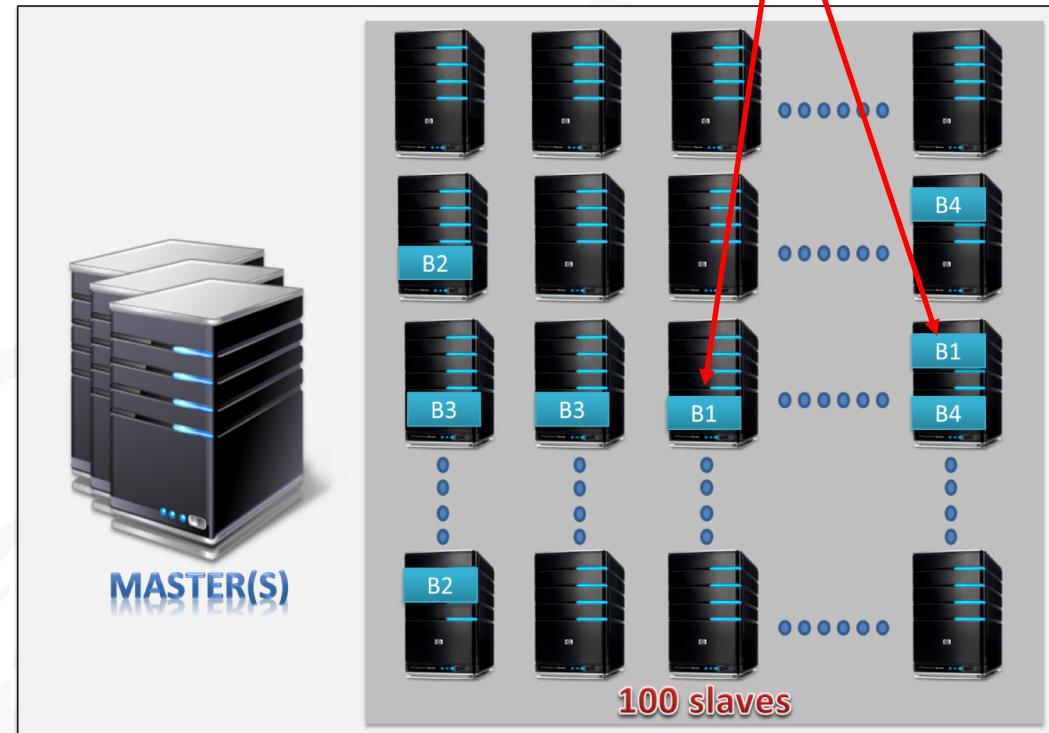
Architettura

- Master → Namenode
- Slave → Datanode



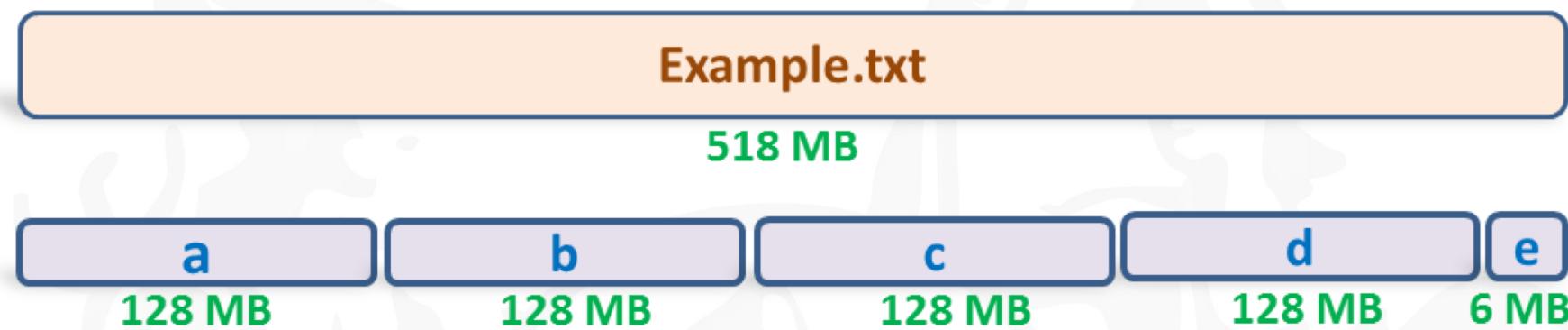
Dimensione minima blocchi: 128MB
Allocazione intelligente dei dati residui

Replication factor:
tre repliche per blocco di default

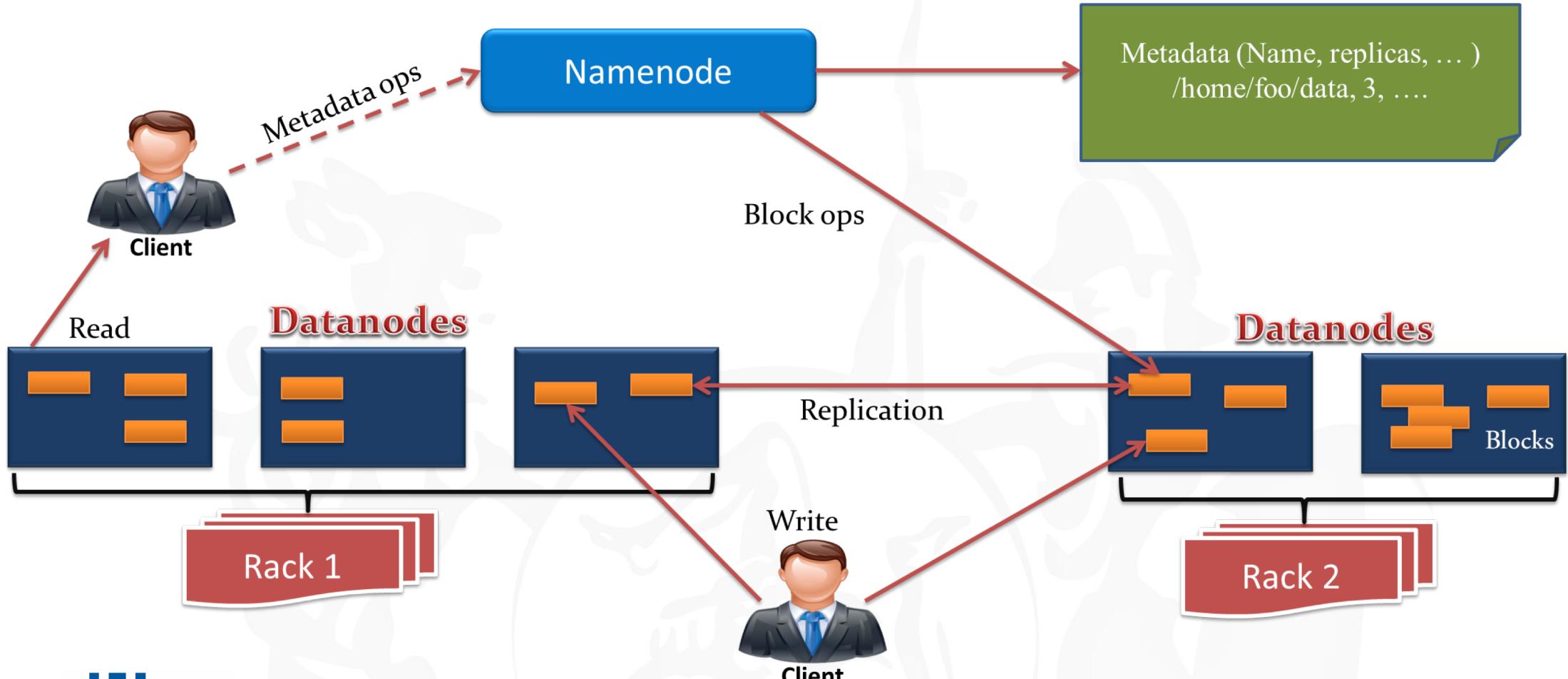


Architettura

- Gestione dei blocchi dei file



Architettura



Architettura

- Namenode
 - Gestisce il namespace del file system
 - Regola l'accesso ai file da parte dei client
 - Esegue le operazioni di file system: apertura, chiusura, rinominazione file e cartelle ...
 - Gestisce le segnalazioni da parte dei datanode:
 - *Heartbeat* cioè la segnalazione di datanode attivo
 - *Block report* che contiene la lista dei datanode attivi
 - Definisce il *replication factor* per tutti i blocchi

Architettura

- Namenode
 - Metadati
 - *FsImage* – l'intero spazio dei nomi del file system è conservato come una sorta di «file immagine» nel file system locale del Namenode
 - Contiene una rappresentazione serializzata di tutte le cartelle e gli inode del file system dove stanno effettivamente i metadati
 - *EditLogs* – contiene tutte le modifiche recenti del file system sulla versione più recente di *FsImage*
 - In questo file sono registrate tutte le richieste di create/update/delete che vengono dai Datanode

Architettura

- Namenode secondario: si utilizza per accelerare i tempi di restart del Namenode, in caso di fallimento
- A startup, il Namenode legge lo stato di HDFS da FsImage e poi applica gli edit contenuti in EditLogs
 - Esegue un merge dei due file che può essere molto lungo per effetto delle loro dimensioni
- Il Namenode secondario esegue il download di FsImage e EditLogs dal Namenode ed effettua il merge per suo conto, conservandolo su storage persistente
- Si sincronizza regolarmente con il Namenode

Architettura

- DataNode

- Lo slave dell'architettura
- Conserva i dati fisici
- Esegue le operazioni di read/write, creazione delle repliche dei blocchi e loro cancellazione
- Si occupa esplicitamente dello storage dei dati
- Ogni 3 sec invia il segnale di heartbeat al Namenode

Architettura

- Checkpoint node
 - Crea checkpoint periodici del namespace
 - Esegue il download di FsImage e EditLogs dal Namenode e ne fa il merge localmente
 - Aggiorna il Namenode facendo l'upload della nuova immagine
 - Conserva il checkpoint in una struttura di cartelle identica a quella del Namenode per renderla sempre immediatamente disponibile a quest'ultimo se necessario

Architettura

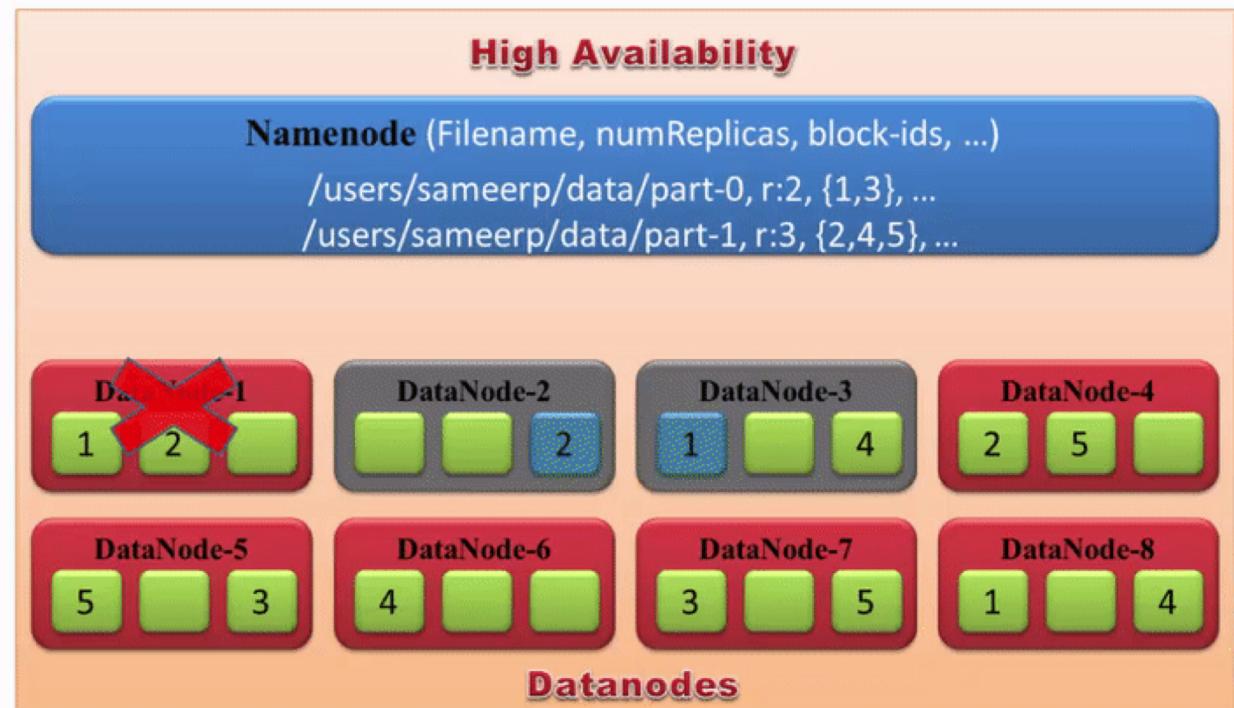
- Backup node
 - Mantiene una copia aggiornata del namespace direttamente in memoria centrale
 - È sempre sincronizzato con il Namenode
 - Non ha bisogno di checkpoint esplicativi: semplicemente scarica FsImage e EditLogs e ne fa il merge
 - C'è un solo backup node alla volta per ogni Namenode

Architettura

- Rack awareness
 - Un *rack* è un cluster di macchine fisiche su cui è fatto il deploy di un insieme di datanode
 - L'algoritmo di rack awareness è un criterio di scelta del datanode più vicino da parte del Namenode per creare un blocco
 - Si basa sulla conoscenza del rack id di ogni datanode
 - Serve a limitare l'impegno di banda passante sulla rete
 - Riduce i costi delle operazioni di read/write
 - La prima replica va sul nodo locale che l'ha creata, la seconda su un altro datanode del rack e la terza su un datanode di un altro rack

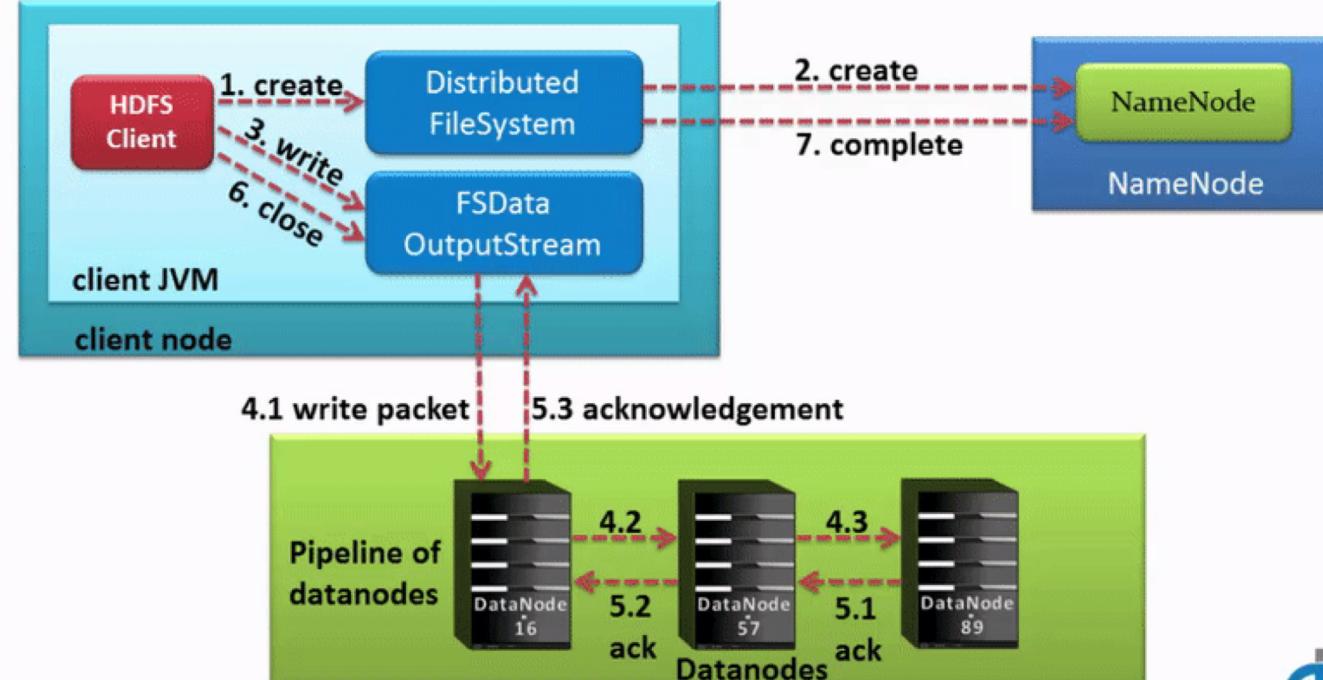
Architettura

- Data availability



Operazioni di read/write

- Flusso delle operazioni di write



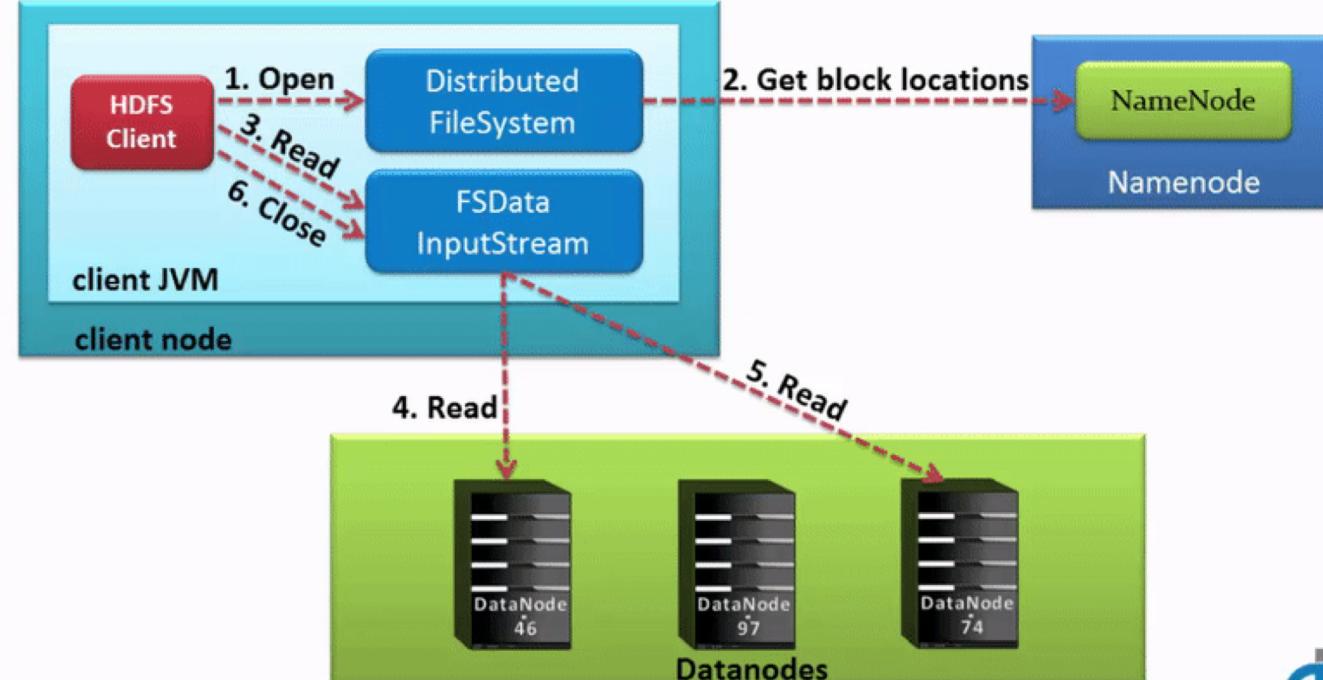
Operazioni di read/write

- Flusso delle operazioni di write

```
1.  FileSystem fileSystem = FileSystem.get(conf);
2.  // Check if the file already exists
3.  Path path = new Path("/path/to/file.ext");
4.  if (fileSystem.exists(path)) {
5.      System.out.println("File " + dest + " already exists");
6.      return;
7.  }
8.  // Create a new file and write data to it.
9.  FSDataOutputStream out = fileSystem.create(path);
10. InputStream in = new BufferedInputStream(new FileInputStream(
11.     new File(source)));
12. byte[] b = new byte[1024];
13. int numBytes = 0;
14. while ((numBytes = in.read(b)) > 0) {
15.     out.write(b, 0, numBytes);
16. }
17. // Close all the file descriptors
18. in.close();
19. out.close();
20. fileSystem.close();
```

Operazioni di read/write

- Flusso delle operazioni di read



Operazioni di read/write

- Flusso delle operazioni di read

```
1. FileSystem fileSystem = FileSystem.get(conf);
2. Path path = new Path("/path/to/file.ext");
3. if (!fileSystem.exists(path)) {
4.     System.out.println("File does not exists");
5.     return;
6. }
7. FSDataInputStream in = fileSystem.open(path);
8. int numBytes = 0;
9. while ((numBytes = in.read(b)) > 0) {
10.     System.out.println((char)numBytes); // code to manipulate the data which is read
11. }
12. in.close();
13. out.close();
14. fileSystem.close();
```

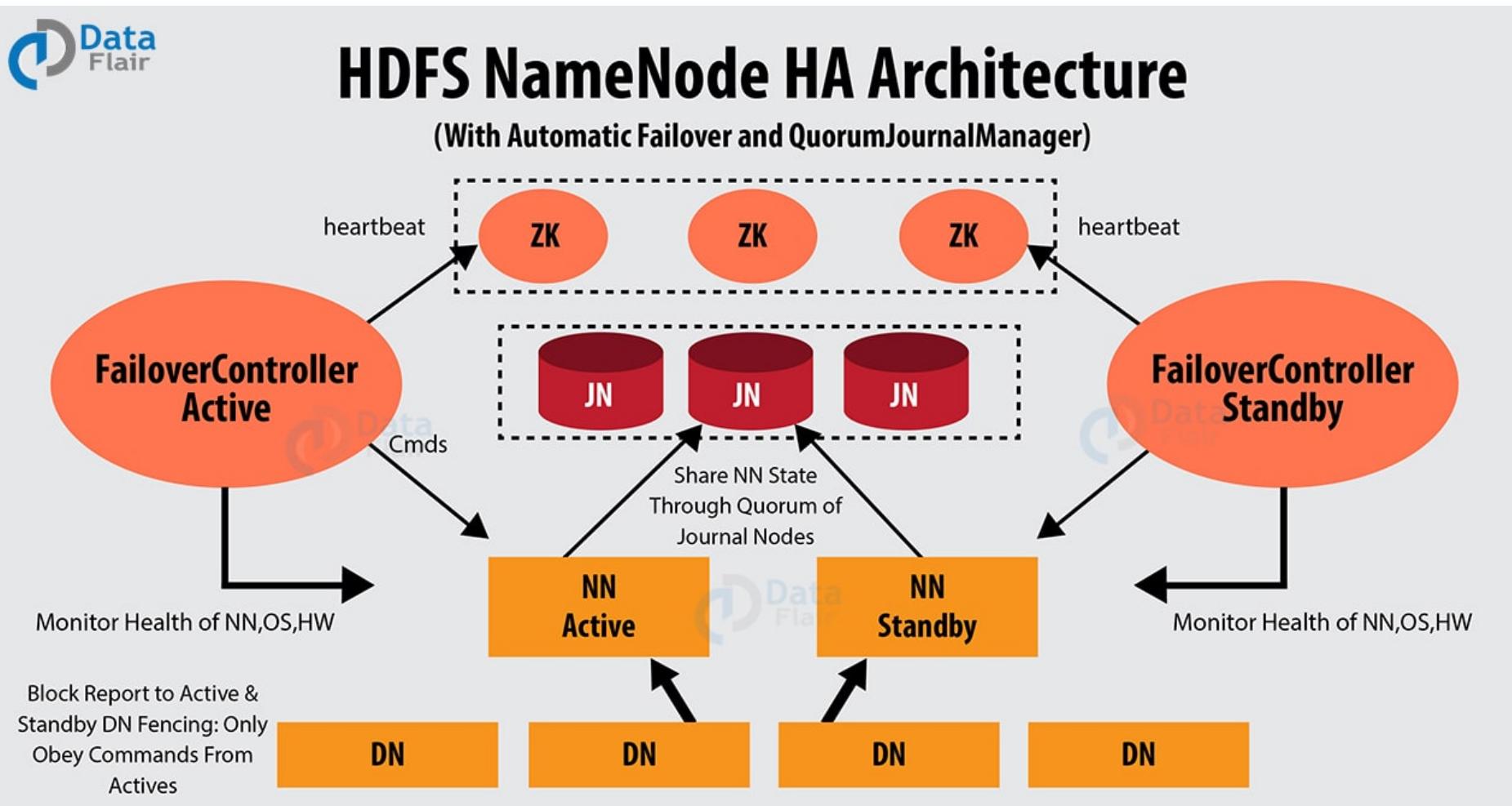
Caratteristiche avanzate

- Namenode high availability
 - Uso di Namenode ridondanti, uno attivo ed uno passivo che sono posti eventualmente in standby da dei controller esterni
 - Il NN passivo viene attivato in caso di fallimento dell'altro che viene a sua volta posto in standby
 - Sincronizzazione continua

Caratteristiche avanzate

- Namenode high availability
 - Quorum Journal Node
 - Nodi di journaling che generano effettivamente gli edit log
 - Sono un pool e condividono gli edit
 - Il NN passivo legge gli edit da tutti i JN in maniera continua e, in caso di fallimento, si propone come attivo solo dopo essere certo di aver aggiornato tutto il proprio namespace
 - N Journal Node $\rightarrow (N-1)/2$ fallimenti
 - I datanode mandano le informazioni sui blocchi contemporaneamente a tutti i NN sia attivi sia passivi

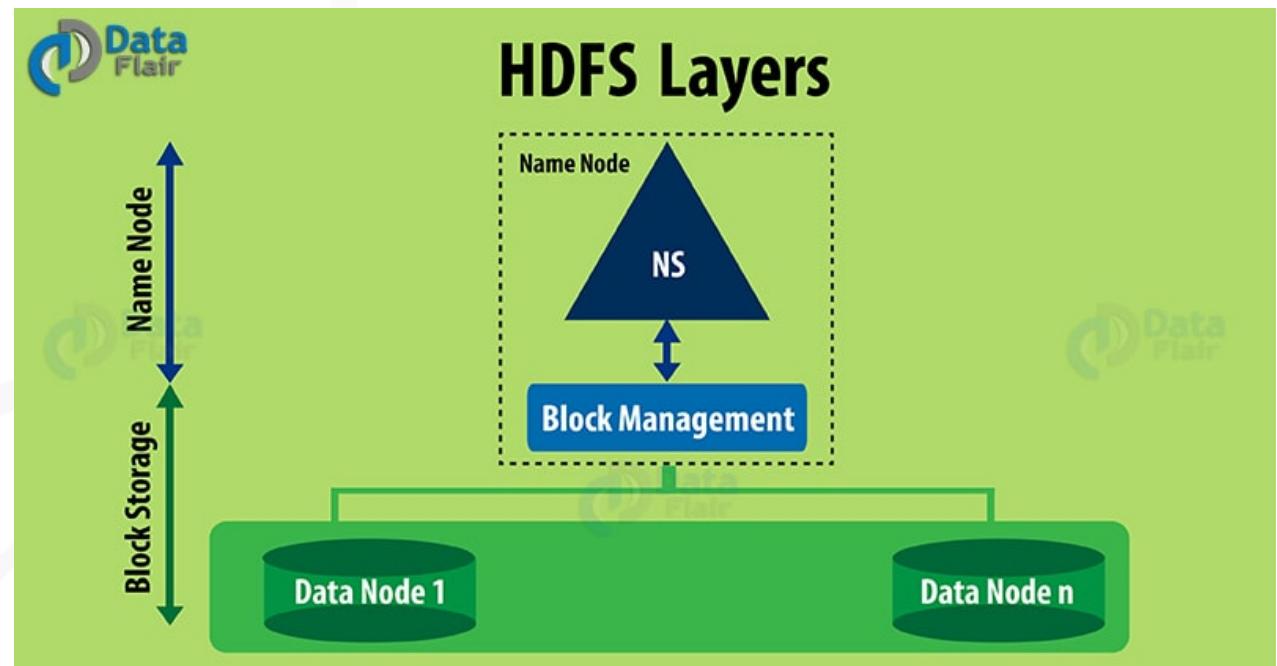
Caratteristiche avanzate



Caratteristiche avanzate

- Federazione

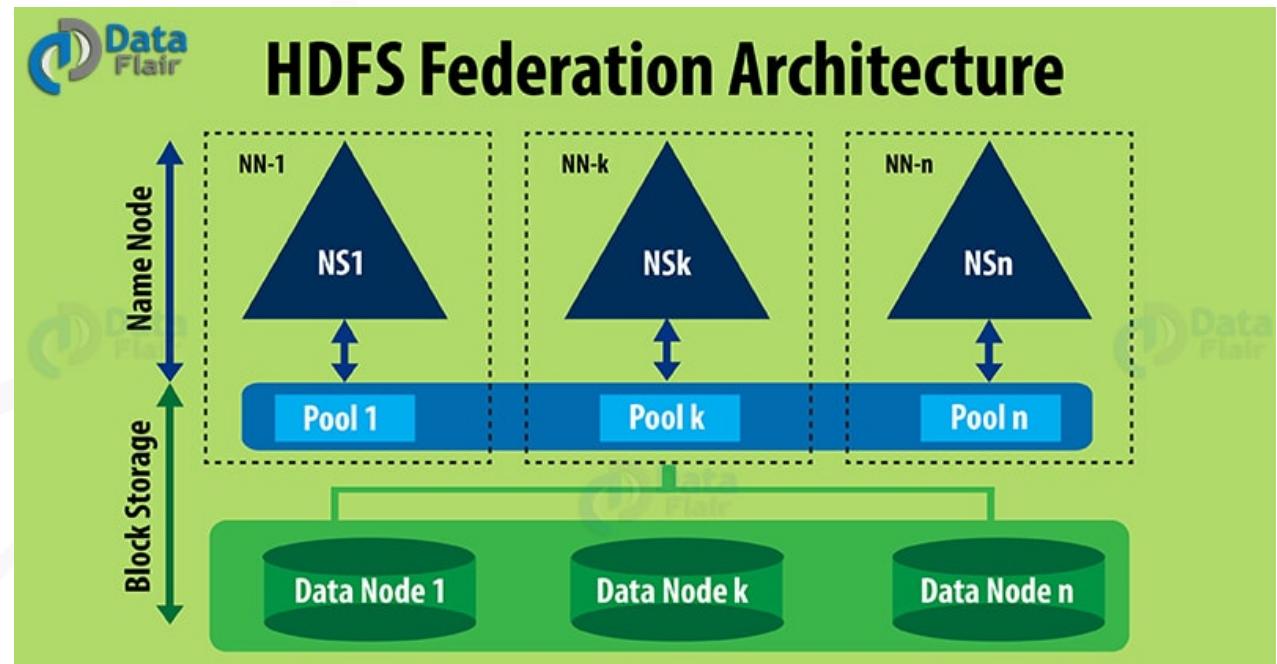
- Un'architettura HDFS soffre di:
 - Forte accoppiamento tra lo storage dei blocchi e il namespace
 - Scalabilità del namespace
 - Performance limitate dal throughput del NN
 - Non c'è isolamento tra le applicazioni e tra chi gestisce il cluster e la gestione del namespace



Caratteristiche avanzate

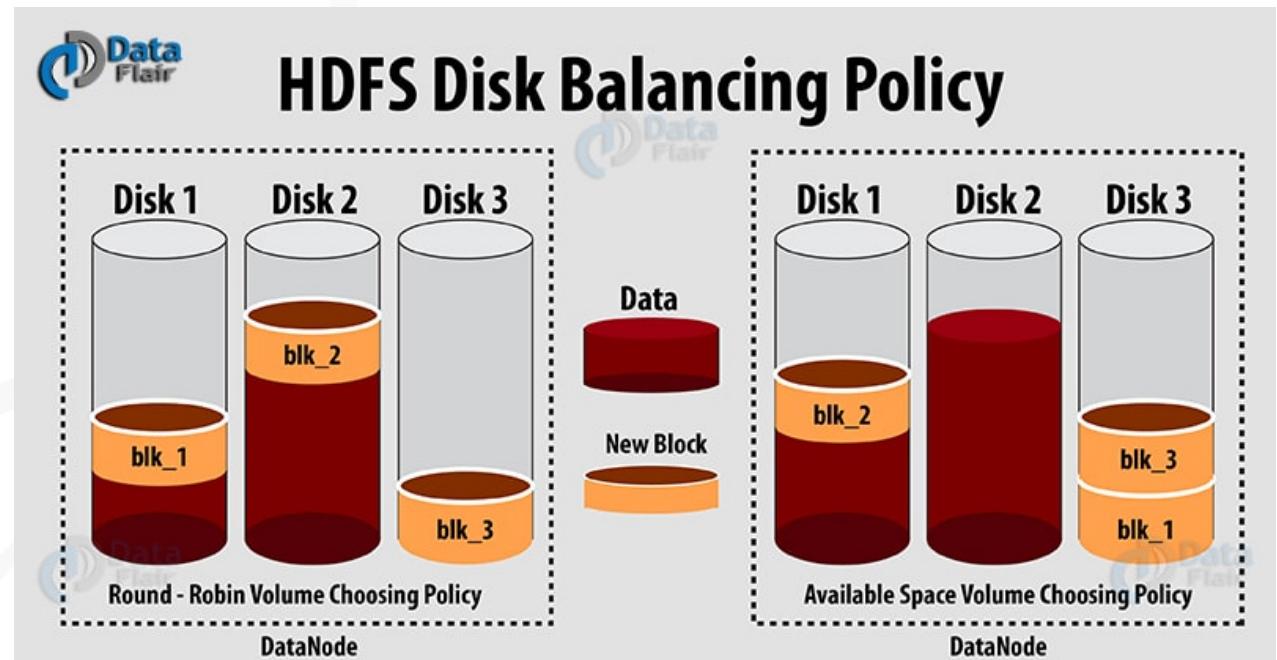
- Federazione

- Diversi NN ciascuno con un *block pool*: ciascun NN crea i propri blocchi
- Gestione separata dei Datanode
- Namespace volume: gestione separata dei namespace come diversi volumi virtuali
- Isolamento tra applicazioni
- Scalabilità dei namespace
- Aumento di performance



Caratteristiche avanzate

- Disk balancing
 - Gestione dello spazio sui dischi del Datanode secondo politiche sia round-robin sia sulla base dello spazio disponibile
 - Spostamento dei dati da un disco all'altro mentre il Datanode è attivo



Caratteristiche avanzate

- Erasure coding
 - Gestione efficiente dei dati ridondanti che supera le limitazioni imposte dalla struttura delle tre repliche di un blocco
 - Diminuzione significativa dello spazio su disco
 - Overhead: 200% → 50%
 - Usa RAID (striping con l'algoritmo Reed-Solomon)
 - Necessita di estensioni software per NameNode, DataNode e Client