Big Data Storage and Processing

MSc in Data Analytics

CCT College Dublin

Hadoop Distributed File System (HDFS)

Week 2

Data Storage & Processing
Data Analysis
Data Warehouse
Big Data Tools
No sequel Database
Realtime Processing
Messaging System

Lecturer: Dr. Muhammad Iqbal*

Email: miqbal@cct.ie

Agenda

cct | College Dublin
Computing • IT • Business

- Introduction
- Hadoop Design Goals
- Hadoop Concepts
- Hadoop Dataflow
- Hadoop Proximity
- Hadoop Replication

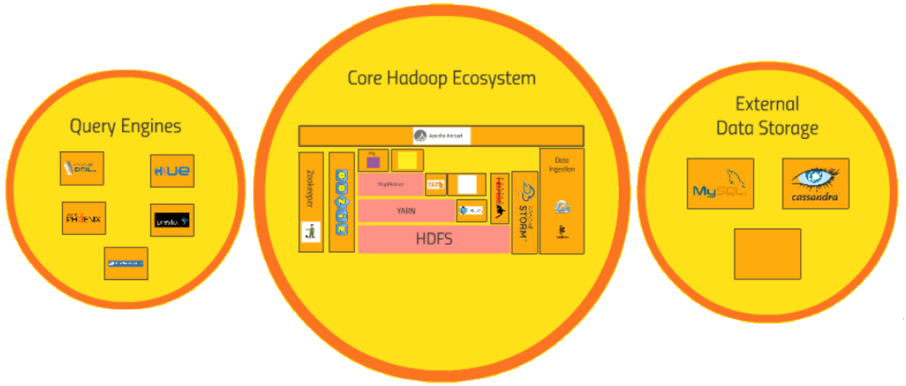
Introduction

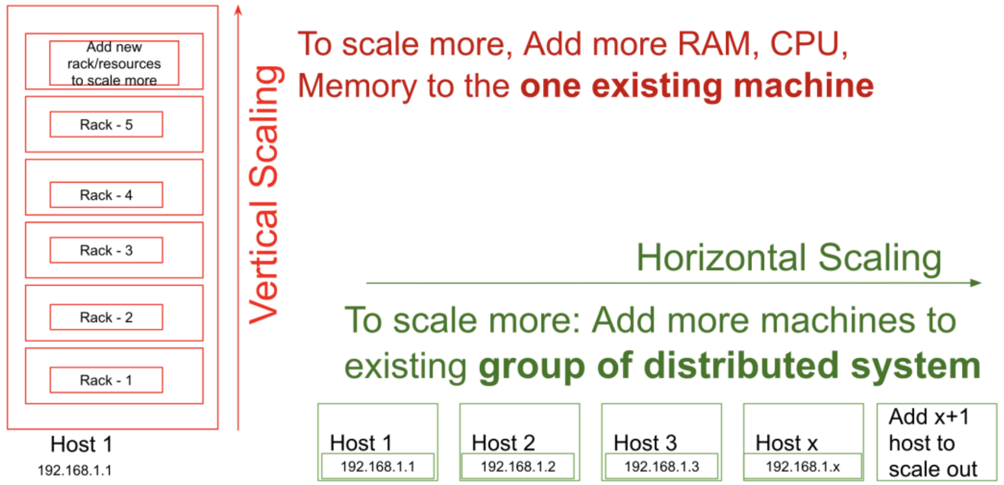
- **What is Hadoop?**
- An open-source software platform for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. *Hortonworks*
- **Hadoop Distributed File System (HDFS)**
    - When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines.
    - Filesystems that manage the storage across a network of machines are called **distributed filesystems**.
    - Storage requirements necessitate partitioning datasets across a number of machines.

Can we ensure filesystem tolerance to a node failure?

Why Hadoop?

- Data's too darn big - Terabytes per day
- **Vertical scaling** doesn't cut it
  - Disk seek times
  - Hardware failures
  - Processing times
- **Horizontal scaling** is linear.
- **Hadoop:** It's not just for batch processing anymore.
- **HDFS** is horizontally scalable.




Add new rack/resources to scale more
Rack - 5
Rack - 4
Rack - 3
Rack - 2
Rack - 1
Host 1
192.168.1.1
Vertical Scaling
To scale more, Add more RAM, CPU, Memory to the **one existing machine**
Horizontal Scaling
To scale more: Add more machines to existing **group of distributed system**
Host 1
192.168.1.1
Host 2
192.168.1.2
Host 3
192.168.1.3
Host x
192.168.1.x
Add x+1 host to scale out

Hadoop Design Goals

- **Hadoop distributed file system (HDFS)** is designed
  - to handle very **large datasets**
  - be deployed on **commodity hardware**
  - exhibit a high level of **fault tolerance**
  - enable streaming access to filesystem data

Megabytes – Terabytes - Petabytes

High chance of node failure

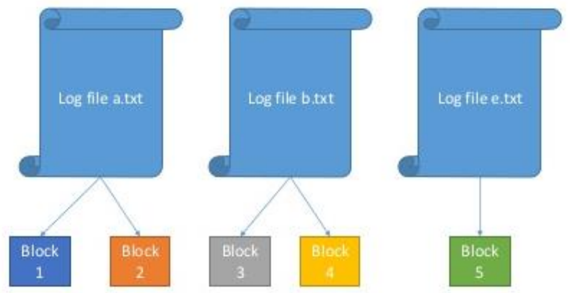
In case of node failure

Write-once, read-many pattern – time to read whole file more important than latency in reading the first record

Hadoop Concepts

Blocks

Files are broken up in 'blocks'



- We are familiar with block I/O at the disk level
- **Filesystems** implement a block size which is typically an integral multiple of the disk block size, e.g.

Disk Block Size	Filesystem Block Size
512 Bytes	4 Kilobytes

- **HDFS** also implements a block size
    - 128MB by default
- **HDFS** files are broken up into block sized chunks which can then be stored as independent units.
- **Note:** A **HDFS** file that is smaller than the **HDFS** block will not take up a full block of space – all blocks in a file are the same size except possibly for the last one.

Why such large HDFS block sizes?

Hadoop Concepts
Blocks

- HDFS blocks are large to reduce the cost of seek operations.

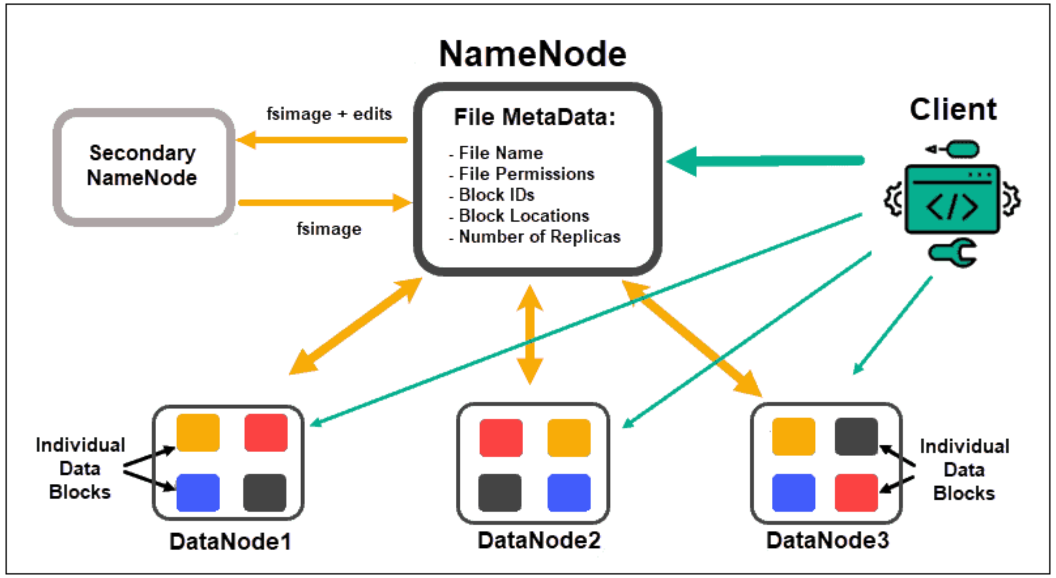
- We would like the time taken to transfer data from the disk to be as close as possible to the disk transfer rate.

  - **Reduce the percentage time spent doing seek operations**

If the seek time is approximately 10ms for a disk with a transfer rate of 100MB/s then to make the seek time 1% of the total transfer time, the block should be configured to approximately 100MB.

Hadoop Concepts

- **Namenodes and Datanodes**
  - **HDFS** has a master/ slave (or master/ worker) architecture
  - A ***namenode*** is the master
  - A set of ***datanodes*** act as the slaves or workers



Source: https://phoenixnap.com/kb/apache-hadoop-architecture-explained

Hadoop Concepts
The Namenode

- Maintains the filesystem tree and the metadata for the files and directories in the tree
  - Stored persistently on disk in the form of two files:

SPOF?

1. **The *namespace image (FSImage file)*** – filesystem namespace, file-block mappings, filesystem properties – 4GB RAM should be plenty for large filesystem
2. **The *edit log (EditLog file)*** – a transaction log

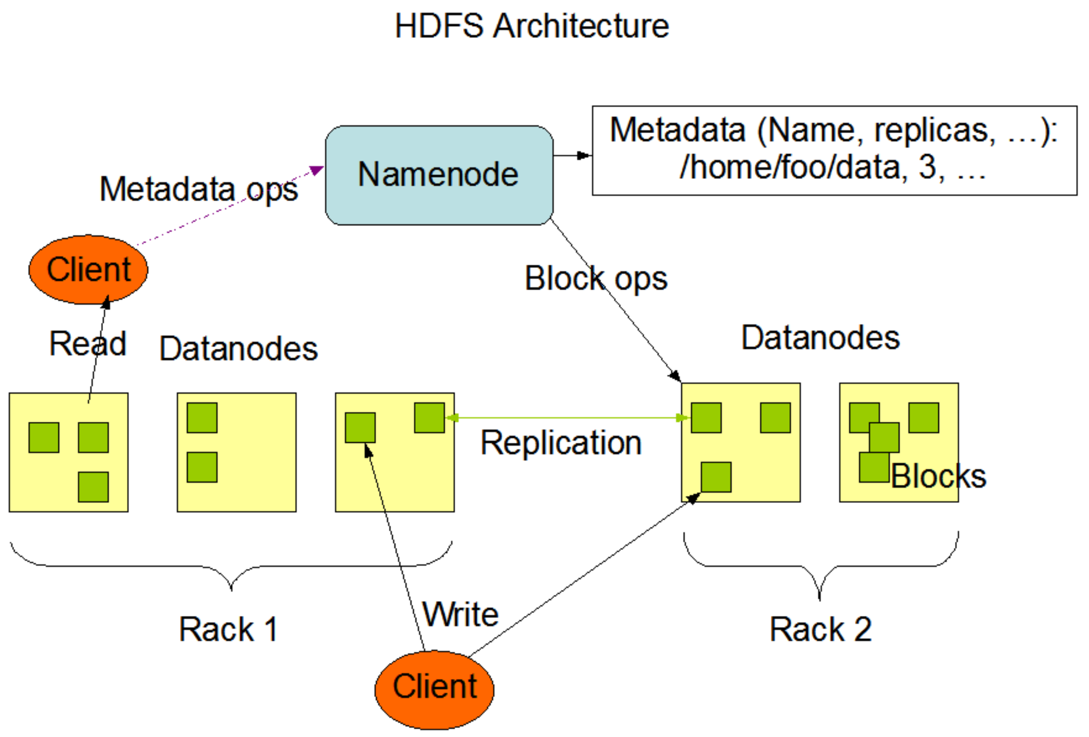
- The **namenode** also knows the **datanodes** on which all the blocks for a given file are located
  - This information is reconstructed from the datanodes when the system starts

Hadoop Concepts

The datanode

- For every node (Commodity hardware/ System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.
- Handle ***read/ write requests*** from clients (or the namenode).
- They also perform operations such as, block creation, deletion, and replication according to the instructions of the namenode.
- Usually, one datanode per node in the cluster.
- **Note:** User data does not flow through the namenode.

Namenodes and Datanodes



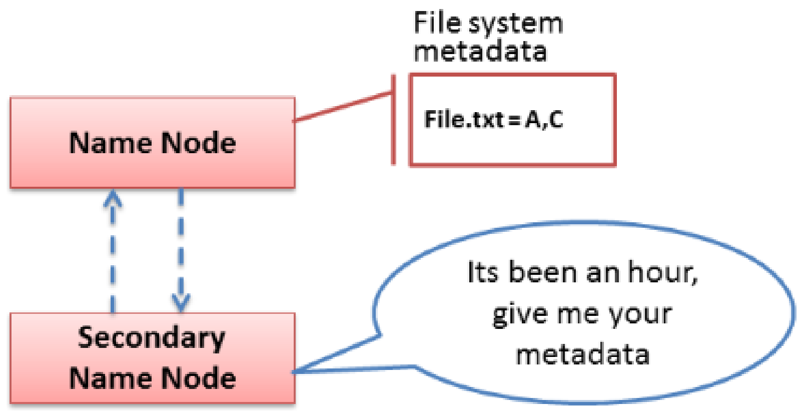
Hadoop Concepts

HDFS High Availability

- **NameNode Failure**
  - Replication of persistent state to remote Network File System (NFS) mount
    - Secondary NameNode (*deprecated)*
    - Checkpoint Node
    - Backup Node
  - Hadoop provides HDFS HA via active-passive standby configuration

SPOF?

- **Active / Standby** – should be machines with equivalent specifications
- **DataNodes** send **heartbeat** and blockreports to both Active and Standby nodes

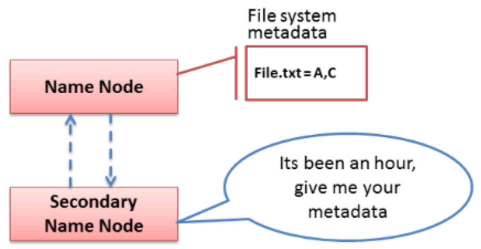


Hadoop Concepts

HDFS High Availability

- **Secondary NameNode**

  - Used to periodically merge the namespace image with the edit log file
  - Should run on a separate machine with same memory requirements as the NameNode
  - Keeps copy of the merged namespace image that can be used in the case of failure



Does not provide HA

- Secondary namenode will lag the primary – high probability of data loss!

Hadoop Concepts

HDFS High Availability

- **Checkpoint Node**

Does not provide HA

  - Replaces the role of the Secondary NameNode.
  - Multiple Checkpoint Nodes can be configured as long as there is no Backup Node operational.
  - It performs periodic checkpoints of the namespace and helps to minimize the size of the log stored at the NameNode containing changes to the HDFS.
  - The NameNode allows multiple Checkpoint nodes simultaneously, as long as there are no Backup nodes registered with the system.

Hadoop Concepts

HDFS High Availability

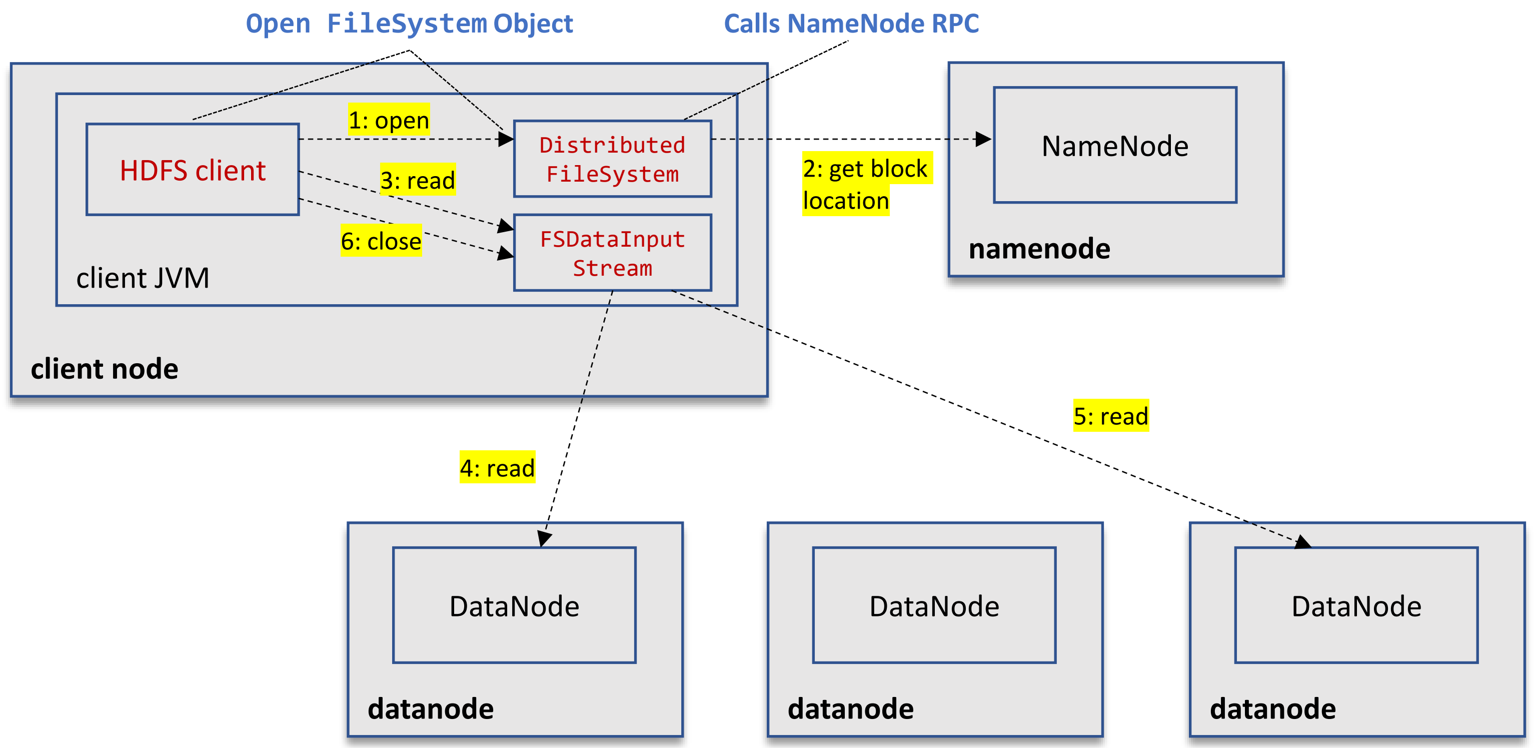
- **Backup Node**

facilitates warm standby possibility

  - Performs checkpointing
  - Also receives a stream of edits from the NameNode and maintains its own in-memory copy of the namespace, which is always in sync with the active NameNode namespace state.
  - Only one Backup node may be registered with the NameNode at once.

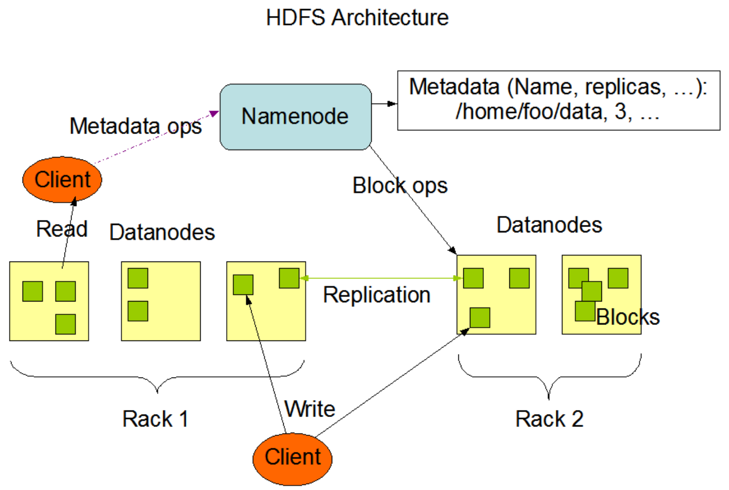
Hadoop Dataflow

File Read



Hadoop Dataflow

File Read



- **Client** **NameNode** **DataNodes**

- Client calls open() on a **`FileSystem` object** (which is an instance of **`DistributedFileSystem`**)

`DistributedFileSystem Object = initialization`

- **`DistributedFileSystem`** calls the namenode via **RPC** to determine locations of first few blocks in the file

- For each block, the namenode returns the addresses of the datanodes that have a copy of that block

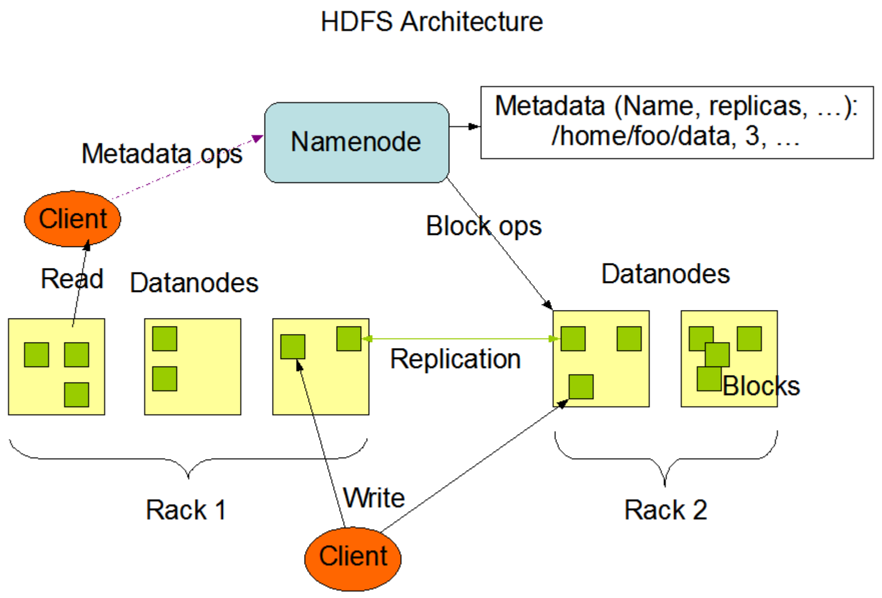
- Datanodes are sorted according to proximity of client process

- **`DistributedFileSystem`** returns a **`FSDataInputStream`** to the client

- **`FSDataInputStream`** wraps a **`DFSInputStream`** which manages the namenode and datanode I/O

Hadoop Dataflow

File Read

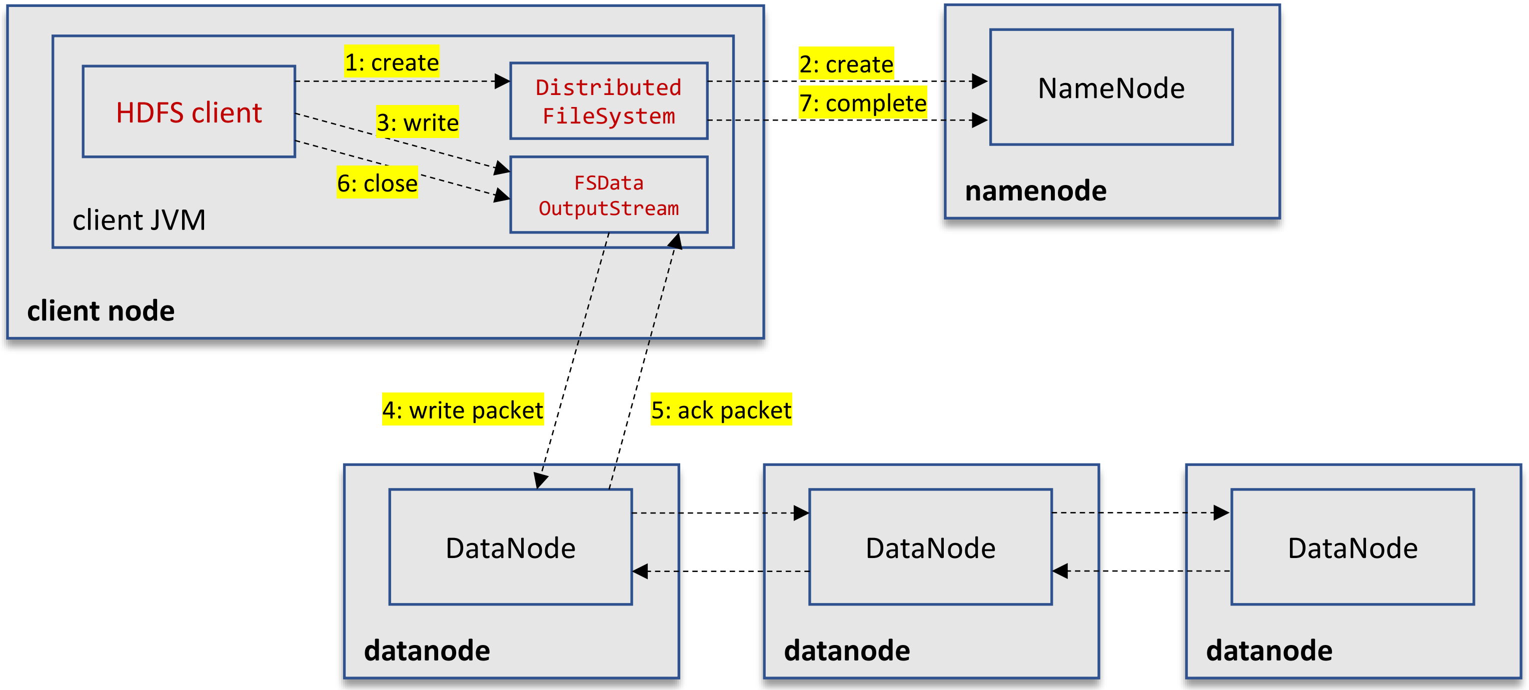


- **Client** **NameNode** **DataNodes**

- The client then calls **read()** on the stream
- **`DFSInputStream`** connects to the first datanode for the first block in the file
- The client calls **read()** repeatedly on the stream
- When the end of the block is reached, **`DFSInputStream`** closes the connection to the datanode and then finds the next datanode for the next block
- **`DFSInputStream`** will make calls when necessary to get the datanode locations for subsequent batches of blocks
- When the client has finished reading it calls **`close()`** on the **`FSDataInputStream`**

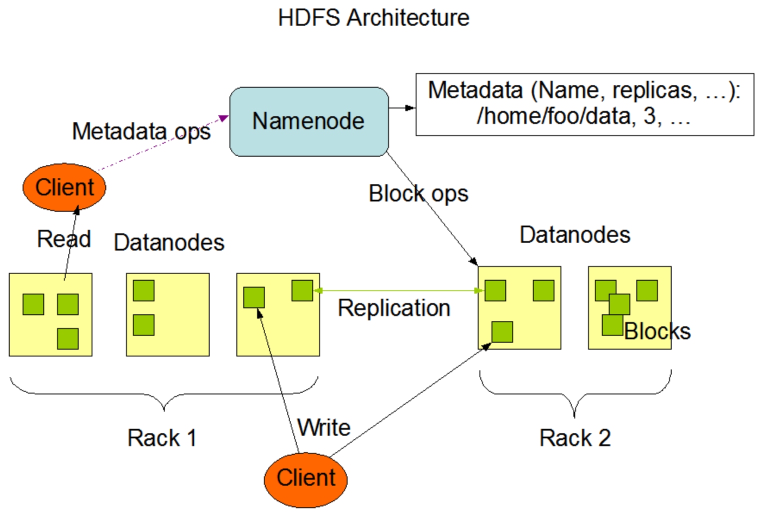
Hadoop Dataflow

File Write



Hadoop Dataflow

File Write

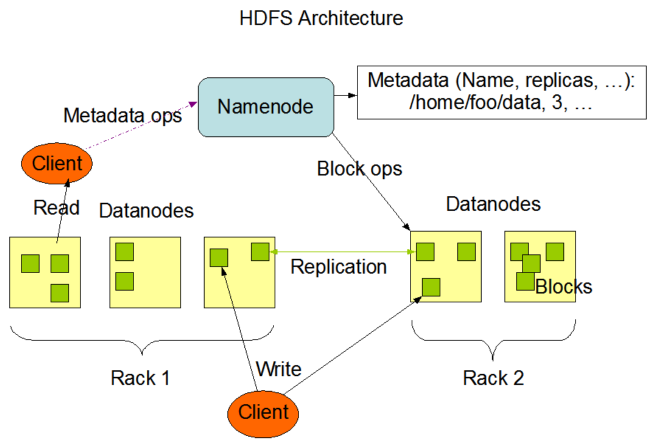


- **Client** **NameNode** **DataNodes**

- Client creates a file by calling **`create()`** on **`DistributedFileSystem`**
- **`DistributedFileSystem`** calls the **namenode** via **RPC** to create a new file in the filesystem's **namespace** (no blocks associated yet)
- **Namenode** performs a series of checks to ensure file does not already exist and that client has required permissions
- If checks pass **namenode** makes a record of the new file
- **`DistributedFileSystem`** returns a **`FSDataOutputStream`** to the client
- **`FSDataInputStream`** wraps a **`DFSOutputStream`** which manages the namenode and datanode I/O

Hadoop Dataflow

File Write

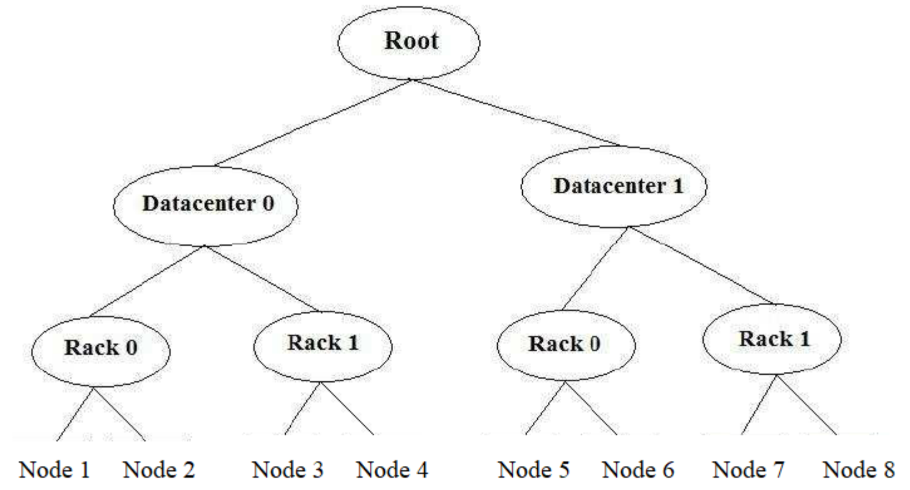


- **Client** **NameNode** **DataNodes**
- As the client writes data **`DFSOutputStream`** splits it into packets which are written to an internal ***data queue***
- The **data queue** is consumed by a **`DataStreamer`** which is responsible for requesting the namenode to allocate new blocks by picking a suitable set of datanodes to store the replicas
- The set of datanodes form a **pipeline (assume 3 datanodes)**
- The **`DataStreamer`** streams the packet to the <u>**first datanode**</u> in the pipeline which stores the packet and forwards it to the <u>**second datanode**</u>
- The <u>**second datanode**</u> stores the packet and forwards it to the <u>**third datanode**</u>
- The **`DFSOutputStream`** also maintains an internal queue of packets that are waiting to be acknowledged by the datanodes – the ***ack queue***
- **A packet is removed from the ack queue only when it has been acknowledged by all the datanodes in the pipeline**

Hadoop Proximity

Network Topology

- Measure of node 'closeness'.
- Use **bandwidth** between nodes as a measure of closeness.
- Network represented as a tree.
- Distance between nodes is the sum of their distances to their closest common ancestor.
- Commonly, levels in the tree correspond to the **data-centre**, the **rack**, and the **node** that the processing is running on.



- **Example:**

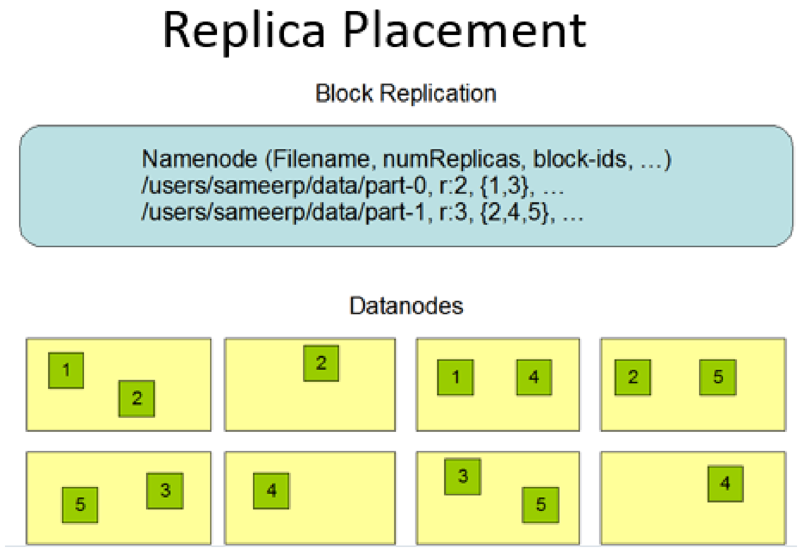
  Consider a node ***n1*** on rack ***r1*** in data-centre ***d1*** being represented by `/d1/r1/n1,` Then

  - *distance(/d1/r1/n1, /d1/r1/n1) = 0*
  - *distance(/d1/r1/n1, /d1/r1/n2) = 2*
  - *distance(/d1/r1/n1, /d1/r2/n3) = 4*
  - *distance(/d1/r1/n1, /d2/r3/n4) = 6*

Hadoop Replication

Replica Placement

- **Blocks** replicated for fault-tolerance.
- Application can specify the number of replicas required. **Replication factor** can be specified at create time and changed later.
- **Namenode** responsible for decision making with respect to replicas.
- **Namenode** will periodically receive a heartbeat from **datanodes** indicating that the **datanode** is functioning correctly.
- **Replica placement** is critical to **HDFS** performance and reliability.
- Optimising replica placement is a distinguishing factor of **HDFS** as opposed to other distributed file systems.
- Possible research area.


Replica Placement
Block Replication
Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Hadoop Replication

Replica Placement

- How does the **namenode** choose **datanodes** on which to store replicas?

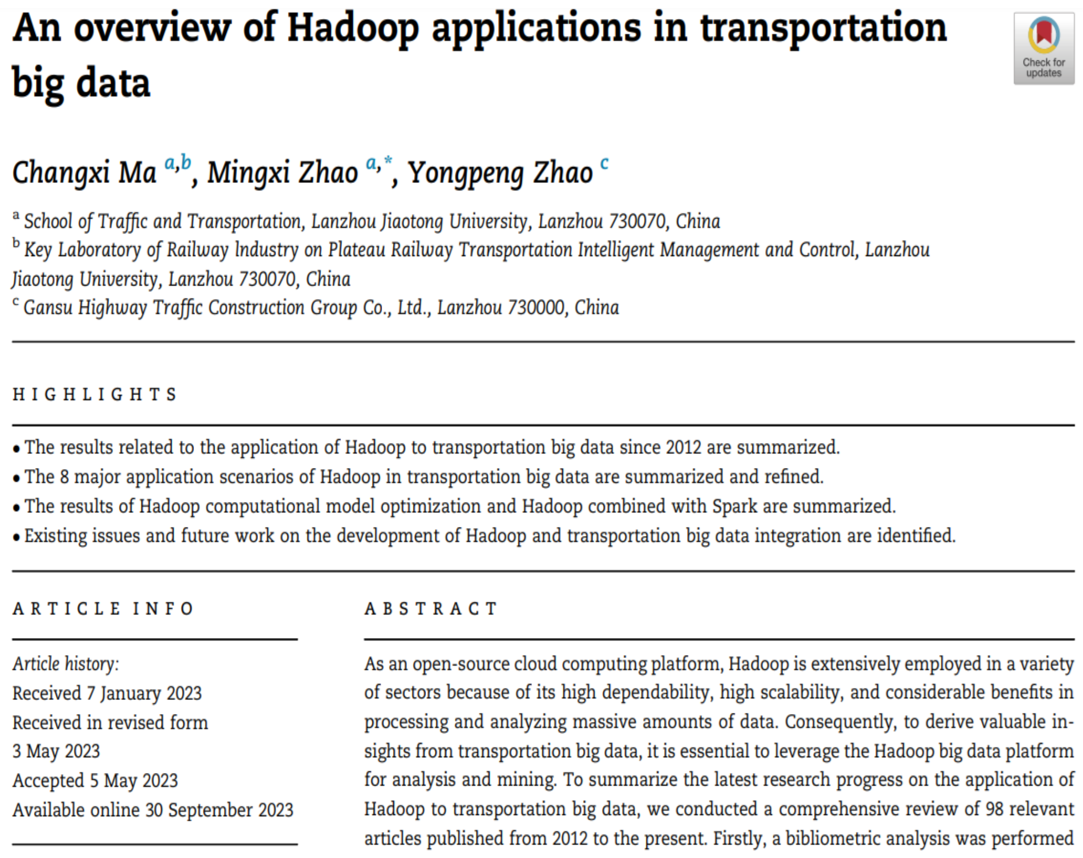
RELIABILITY
BANDWIDTH

- **All replicas on a single node**
  - Lowest write bandwidth penalty since replication pipeline runs on a single node
  - No real redundancy though
  - Read bandwidth high for off-rack reads
- **All replicas in different data-centres**
  - Maximise redundancy
  - High bandwidth cost

Hadoop Replication

Replica Placement

- Default behaviour is to place the first replica on the same node as the client.
- If the client is running from outside the cluster then a datanode is randomly chosen (taking account of which datanodes are busiest and the capacity available on datanodes).
- The second replica is placed ***off-rack*** randomly.
- The third replica is placed on the same rack as the second replica but on a different node chosen at random.
- Further replicas are placed on random nodes in the cluster with an effort not to place too many replicas on the same rack.



An overview of Hadoop applications in transportation big data

Changxi Ma [a,b], Mingxi Zhao [a,*], Yongpeng Zhao [c]

[a] School of Traffic and Transportation, Lanzhou Jiaotong University, Lanzhou 730070, China
[b] Key Laboratory of Railway Industry on Plateau Railway Transportation Intelligent Management and Control, Lanzhou Jiaotong University, Lanzhou 730070, China
[c] Gansu Highway Traffic Construction Group Co., Ltd., Lanzhou 730000, China

HIGHLIGHTS

- The results related to the application of Hadoop to transportation big data since 2012 are summarized.
- The 8 major application scenarios of Hadoop in transportation big data are summarized and refined.
- The results of Hadoop computational model optimization and Hadoop combined with Spark are summarized.
- Existing issues and future work on the development of Hadoop and transportation big data integration are identified.

ARTICLE INFO

Article history:
Received 7 January 2023
Received in revised form
3 May 2023
Accepted 5 May 2023
Available online 30 September 2023

ABSTRACT

As an open-source cloud computing platform, Hadoop is extensively employed in a variety of sectors because of its high dependability, high scalability, and considerable benefits in processing and analyzing massive amounts of data. Consequently, to derive valuable insights from transportation big data, it is essential to leverage the Hadoop big data platform for analysis and mining. To summarize the latest research progress on the application of Hadoop to transportation big data, we conducted a comprehensive review of 98 relevant articles published from 2012 to the present. Firstly, a bibliometric analysis was performed

Resources/ References

- Mastering Hadoop 3, Chanchal Singh, Manish Kumar, Packt Publishing, February 2019, 544 pages.
- Hadoop with Python, Zach Radtka; Donald Miner, O'Reilly Media, Inc., 2015.
- Data Analysis with Python and PySpark, By Jonathan Rioux, Manning Publications, March 2022, 456 pages.
- Lublinsky B., Smith K. T. and Yakubovich A 2013, Professional Hadoop Solutions, Wrox [ISBN: 13:978-11186]
- Holmes A 2012, Hadoop in Practice, Manning Publications [ISBN: 13:978-16172]
- McKinney W. 2012, Python for Data Analysis, O'Reilly Media [ISBN: 13: 978-14493]
- https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- Some images are used from Google search repository (https://www.google.ie/search) to enhance the level of learning.