# CS2510 – Computer Operating Systems

## HADOOP
## Distributed File System

**Dr. Taieb Znati**
**Computer Science Department**
**University of Pittsburgh**

## Outline

### HDF Design Issues

- HDFS Application Profile
- Block Abstraction
- Replication
- Namenode and Datanodes

# Outline

- Hadoop Data Flow
  - Read() and Write =() Operations
- Hadoop Replication Strategy
  - Hadoop Topology and Metric
- Hadoop Coherency Model
  - Semantics
  - Sync() Operation

3

# Hadoop Distributed Filesystem

## HDFS Design

## Apache Software Foundation Hadoop Project

- Hadoop is the top-level ASF project
  - A framework for the development of highly scalable distributed computing applications.
  - The framework handles the processing details, leaving developers free to focus on application logic
- Hadoop holds various subprojects

5

## Hadoop Project

- Hadoop Core, provides a distributed file system (HDFS) and support for the MapReduce
- Several other projects are built on Hadoop Core
  - HBase provides a scalable, distributed database.
  - Pig is a high-level data-flow language and execution framework for parallel computation.
  - Hive is a data warehouse infrastructure to support data summarization, ad-hoc querying and analysis of datasets.
  - ZooKeeper is a highly available and reliable coordination system

6

## The Design of HDFS

- HDFS is a file system designed for storing **very large** files with **streaming data access** patterns, running on clusters of **commodity** hardware.
  - HDFS supports files that are hundreds of megabytes, gigabytes, or terabytes in size.
  - HDFS's data processing pattern is a write-once, read many-times pattern.
  - Hadoop is designed to run on clusters of commodity hardware
    - HDFS is designed to tolerate failures without disruption or loss of data

7

## HDFS Streaming Data Access

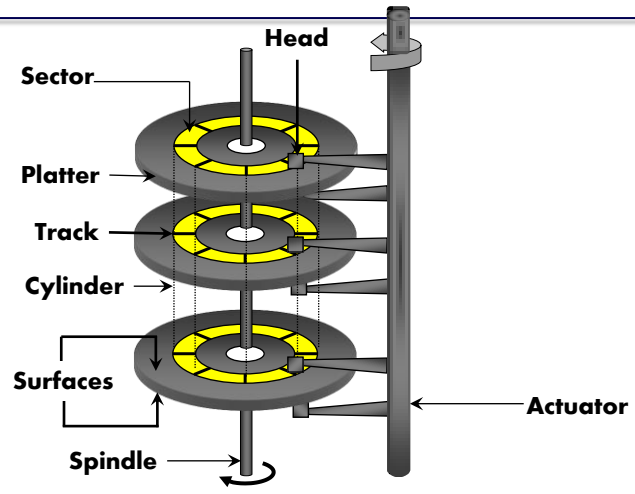
- HDFS supports applications where dataset is typically generated or copied from a source, then various analyses are performed on that dataset over time.
  - Each analysis involves a large proportion, if not all, of the dataset
    - Time to read the whole dataset is more important than the latency in reading the first record of the set

8

# Hadoop Distributed Filesystem

## HDFS Design

## Disk drive structure




10

# Hadoop Distributed Filesystem

## HDFS Design

## Hard Disk Drive Latency

- A read request must specify several parameters
  - Cylinder #, Surface #, Sector #, Transfer Size, and Memory Address
- Disk Latency
  - **Seek time**, to get to the track – it depends on # of tracks, arm movement and disk seek speed
  - **Rotational delay**, to get to the sector under the disk head – it depends on rotational speed and how far the sector is from the head
  - **Transfer time**, to get bits off the disk – it depends on data rate of the disk (bit density) and the size of access request
- Disk Latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead

12

## Applications Not Suited for HDFS

- Applications that require low-latency access, as opposed to high throughput of data
  - HBase is better suited for these types of applications
  - Applications with a large number of small files require large amount of metadata and may not be suited for HDFS
    - These applications may require large amounts of memory to store the metadata
  - HDFS does not support applications with multiple writers, or modifications at arbitrary offsets in the file
    - Files in HDFS may be written to by a single writer, with writes always made at the end of the file

13

## HDFS Blocks

- A disk block represents the minimum amount of data that can be read or written
- A file system block is a higher-level abstraction
  - Filesystem blocks are an integral multiple of the disk block size,
    - Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes.
- HDFS supports the concept of a block, but it is a much larger unit—64 MB by default.
  - Files in HDFS are broken into block-sized chunks, which are stored as independent units

14

## HDFS Block Size

- HDFS blocks are large to minimize the cost of seeks.
  - Large size blocks reduces the transfer time of the data from the disk relative to the time to seek to the start of the block
    - Time to transfer a large file made of multiple blocks operates at the disk transfer rate.
    - For a seek time of 10ms and a transfer rate of 100 MBps, a block size of ~100MB is required to make the seek time 1% of the transfer time
      - HDFS default is 64 MB, and in some cases 128 MB blocks

15

## Block Abstraction Benefits – Distributed Storage

- Block abstraction are useful to handle very large data set in a distributed environment
  - A file can be larger than any single disk in the network
    - Blocks from a file can be stored any of the available disks in the cluster.
    - In some cases, blocks from a single file can fill all the disks of an HDFS cluster

16

# Block Abstraction Benefits – Improved Storage Management

- Making a block, rather than a file, the unit of abstraction simplifies the storage subsystem
  - Provides needed flexibility to deal with various failure modes, an intrinsic feature of HDFS clusters
- Blocks have fixed sizes, which greatly simplifies the storage subsystem and storage management
  - Makes it easy to determine the number of blocks that can be stored in a disk
  - Removes metadata concerns – Blocks are just a chunk of data to be stored and file metadata such as permissions information does not need to be stored with the blocks
    - Another system can handle metadata orthogonally

17

# Block Abstraction Benefits – Improved Failure Tolerance

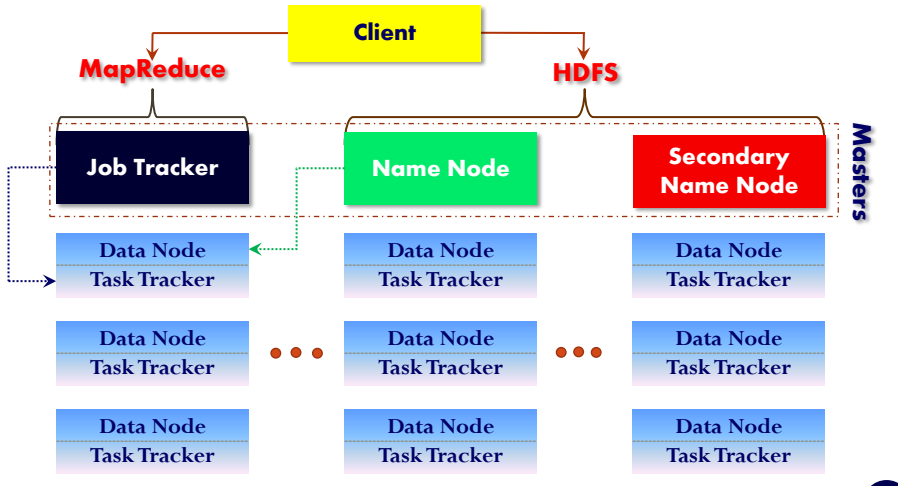
- The block abstraction is well-suited for replication to achieve the desired level of fault tolerance and availability
  - To insure against corrupted blocks and disk and machine failure, each block is replicated to a small number of physically separate machines
    - The default replication factor is three machines, although some applications may require higher values
    - The replication factor is maintained continuously
      - A block that is no longer available is replicated in alternative location using remaining replicas

18

# Hadoop Distributed Filesystem

## HDFS ARCHITECTURE

## Hadoop Server Functionality




20

## Node Categories

- **Client** node is responsible for workflow
  - Load data into cluster (HDFS Reads)
  - Provide the code to analyze data (MapReduce)
  - Store results in the cluster (HDFS Writes)
  - Read results from the cluster (HDFD Reads)
- A HDFS **Name Node** and **Data Nodes**
  - Name node – **master** node – overseas and coordinates the data storage functions of HDFS
  - A datanode stores data in HDFS
    - Usually more than one node with replicated data
- **Job Tracker** overseas and coordinate parallel processing of data using MapReduce

21

## HDFS Namenode and Datanodes

- Namenode maintains the file system tree and the metadata for all the files and directories in the tree.
  - This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log.
  - The namenode also knows the datanodes on which all the blocks for a given file are located,
  - The namenode does not store block locations persistently
    - This information is reconstructed from datanodes when the system starts

22

## HDFS Datanodes

- On startup, each datanode connects to the namenode
  - Datanodes cannot become functional until namenode services is up
- Upon startup, datanodes respond to requests from the namenode for filesystem operations.
- Client applications can have access directly to a data nodes,
  - Clients obtain datanodes' location from the namenode

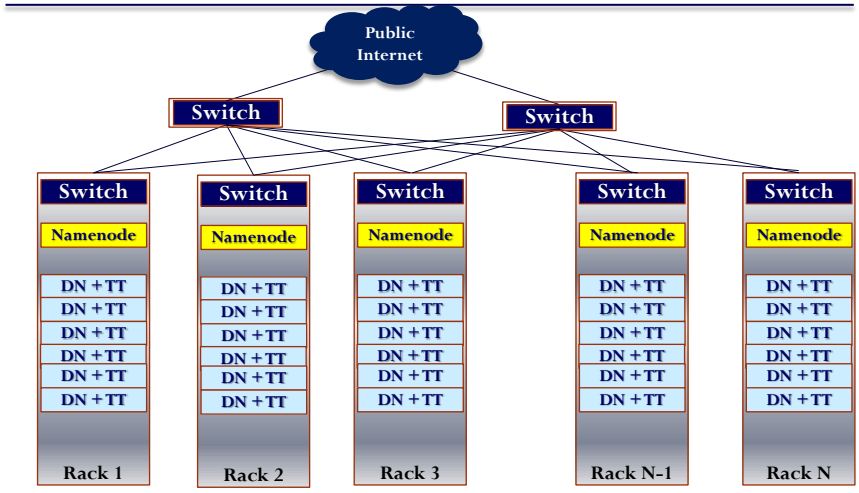
23

## HDFS Datanodes -- Heartbeat

- Datanodes send heartbeats to the Namenode every 3 seconds
  - Every 10th heartbeat is a "Block Report"
    - Data nodes uses Block Report to tell the Namenode about all the blocks it has
    - Block Reports allow the Namenode to build its metadata,
    - It ensures that **three** copies of each data bock exist on different data nodes
      - Three copies is HDFS default, which can be configured with the **dfs.replication** parameter in the **hdfs-site.xml**

24

## Cluster Topology




25

# Hadoop Distributed Filesystem

## HDFS REPLICA ASSIGNMENT

### Rack Awareness

## Replica Placement Tradeoffs

- Replica placement strategy achieves good balance between reliability, bandwidth and performance
  - Reliability – blocks are stored on two racks
  - Reduced bandwidth – writes() only have to traverse a single network switch
  - Read performance – Only a choice of two racks to read from
  - Cluster block distribution – clients only write a single block on the local rack

27

## Tradeoffs in Replica Placement

- Tradeoff between reliability, write bandwidth and read bandwidth
  - **One extreme** – placing all replicas on a single node incurs the lowest write bandwidth penalty, but offers no real redundancy .
    - Also, read bandwidth is high for off-rack reads.
  - **Other extreme** – placing replicas in different data centers maximizes redundancy, at the cost of bandwidth.
  - Hadoop designed placement strategy to achieve a balance between the two extremes

28

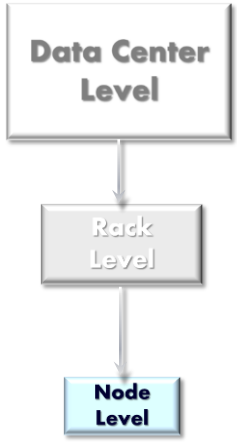
# Hadoop Distributed Filesystem

## HADOOP TOPOLOGY
## Distance Metric

## Hadoop Network Topology

- In the context of data intensive processing, **data transfer** rate is the **limiting factor**
- What is does it mean for two nodes to be close?
  - Ideally, measure should be expressed in terms of bandwidth
    - Difficult to measure in practice
- Hadoop's approach considers the network as a tree
  - Distance(N1, N2) = Sum of the distance to their closest ancestor

- Tree Topology




30

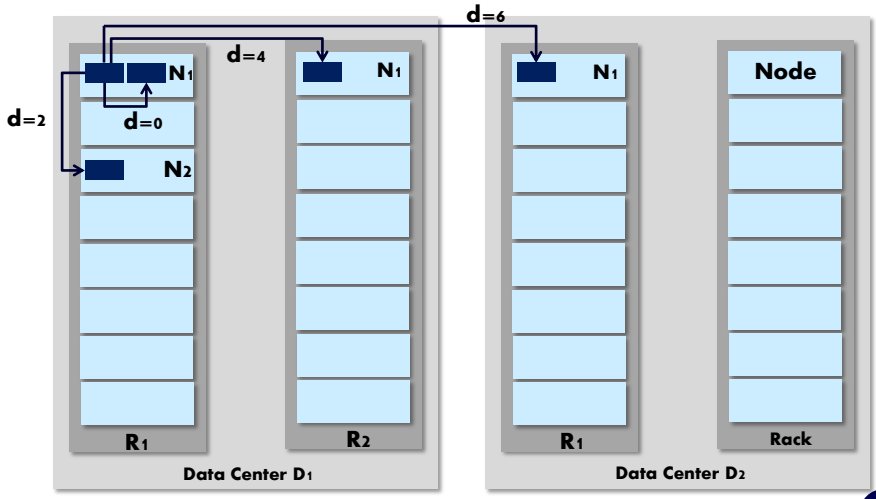
# Network Topology – Closeness Measure

- The tree levels reflect the idea that the bandwidth available decreases progressively
  - Processes on the same node
  - Different nodes on the same rack
  - Nodes on different racks in the same data center
  - Nodes in different data centers

- **Dist(/D1/R1/N1, /D2/R2/N2)** = Distance between node N1 on rack R1 in datacenter D1 and node N2 on rack R2 in datacenter D2
  - Dist(/**D1**/**R1**/**N1**, /**D1**/**R1**/**N1**) = 0
    - Processes on the same node
  - Dist(/**D1**/**R1**/N1, /**D1**/**R1**/N2) = 2
    - Different nodes on the same rack
  - Dist(/**D1**/R1,N1, /**D1**/R2/N3) = 4
    - Nodes on different racks, in the same data center
  - Dist(/D1/R1/N1, /D2/R3/N3) = 6
    - Nodes on different data centers

31

# Hadoop Topology – Node Distance



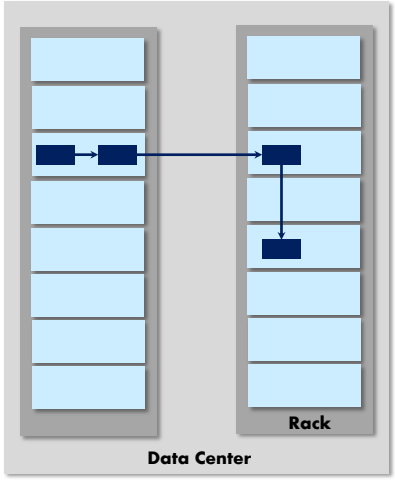

32

# Hadoop Replica Placement Strategy

- Place the first replica on the same node as the client
  - For clients outside the cluster, a node is chosen at random although the system tries not to pick nodes that are too full or too busy
- Place the second replica off-rack – on a **different** rack from the first, chosen at random
- Place third replica on **same** rack as the second, but on a different node chosen at random
- Further replicas are placed on random nodes on the cluster
  - System may try to avoid placing too many replicas on the same rack

33

# Hadoop Replica Placement Example




34

# Hadoop Distributed Filesystem

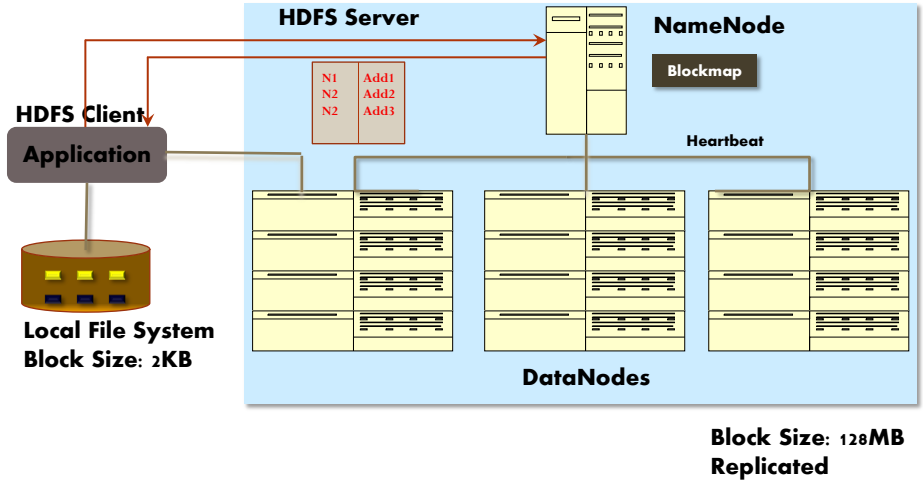
## HADOOP
## File Loading

## Loading Files in HDFS

1. Client breaks the file into blocks, $B_1$, $B_2$, … $B_N$
2. Client informs Namenode that it wasn't to write the file,
3. Client gets permission for Namenode to write the file, and receives a list of three Datanodes for each block
4. Client contacts Datanodes successively for each block, makes the datanode is ready to receive the block and sends the block to the Datanode
   - Data is replicated across the datanodes, as described in the list.

36

# Hadoop Distributed File System



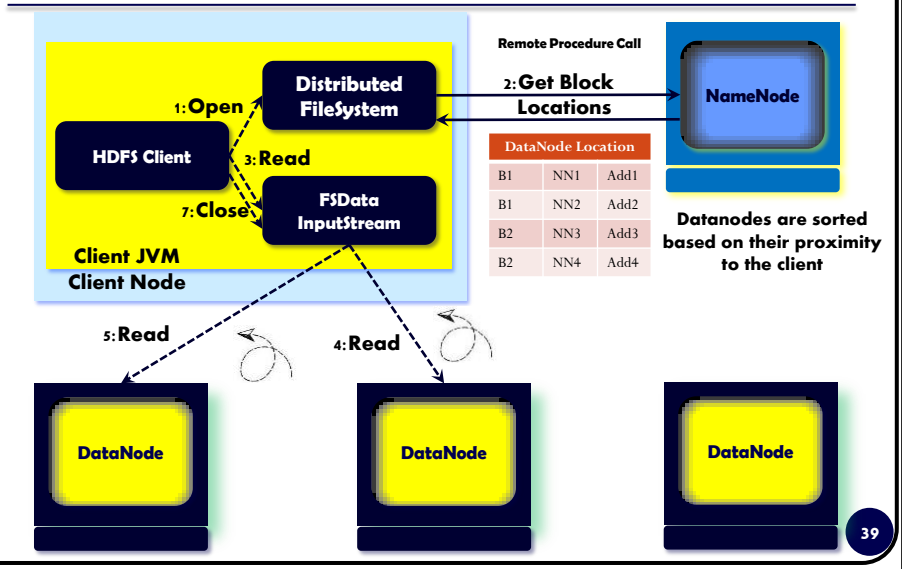

37

# Hadoop Distributed Filesystem

## HDFS OPERATIONS ANATOMY

## READ OPERATION

# Data Flow – File Read




39

# Read () Characteristics

- **Location Optimality**
  - Guided by the namenode mapping table, client contacts **best** datanodes directly to retrieve data for each block
- **Scalability** – HDFS scales to a large number of concurrent clients
  1. Data traffic is distributed across all the data nodes
  2. Namenodes merely services block location requests –
     - The entire (block, datanode) mapping table is stored in memory, for efficient access
  3. If the client is itself a datanode, e.g., **MapReduce task**, the read is performed locally

40

## Read() Operation

- The DistributedFileSystem returns a FSDataStream to the client
  - FSDataStream is an input stream that supports file seeks
  - FSDataStream wraps a DFSInputStream, which manages the datanode and namenode I/O communication
- Client repeadtly calls read() on the stream to read blocks
  - DFSInputStream uses the address mapping table to connect to the "closest" datanode to read blocks
    - Blocks are read in order, with the DFSInputStream opening new connections to datanodes as the client reads through the stream
      - DFSInputStream calls namenode to retrieve the datanode locations for the next batch of blocks as needed
  - All happens transparently from the user, as if a continuous stream is being read.

41

## Read() Operation – Case of Failure

- In case of communication errors with a datanode, client contacts the next closet block to read the block
- The client also remembers the node failure and no longer contacts the failed node for later blocks
- In case of a block checksum failure, the client declares the block as corrupted and reports the failure to the namenode
- The client then attempts to read a replica of the block from another namenode

42

# Hadoop Distributed Filesystem

## HDFS OPERATIONS ANATOMY
## WRITE OPERATION

## Data Flow – File Creation

- DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it
- Namenode performs various checks to make sure the file doesn't already exist, and that the client has the right permissions to create the file
  - If **success**, namenode makes a record of the new file
    - DistributedFileSystem returns a FSDataOutputStream for the client to start writing data to
      - DFSOutput Stream, handles communication with the datanodes and namenode
  - If **failure**, an exception is thrown at the client

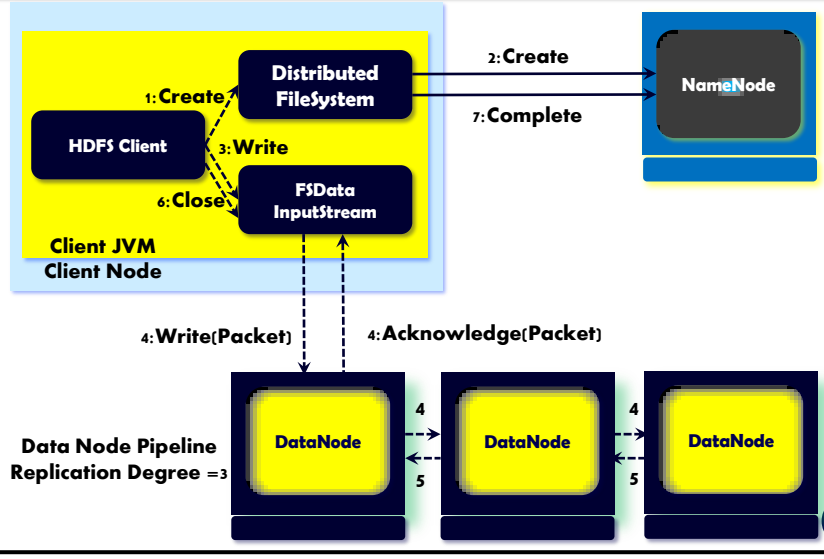
44

# Data Flow – File Write()

- DFSOutputStream splits data into packets
  - Pakckets are written to an internal queue, called the *data queue*, to be consumed by the Data Streamer,
    - Data Streamer asks namenode to allocate new blocks by selecting a list of suitable datanodes to store the replicas
      - The list of datanodes forms a pipeline, usually a replication of level 3
    - DataStreamer streams the packets to the first datanode in the pipeline, for replication across the pipeline
- DFSOutputStream maintains an internal *ack queue* of packets to be acked by datanodes
  - A packet is removed from the ack queue only when it has been acked by all the datanodes in the pipeline

45

# Data Flow – File Write




46

## Write Failure Recovery – Step 1

- Failure of a datanode during a write, leads to the following set of actions:
  - Pipeline is closed, and any packets in the ack queue are added to the front of the data queue
    - Gurantees that datanodes that are downstream from the failed node do not miss any packets
  - Current block on the good datanodes is given a new identity, which is communicated to the namenode
    - Gurantees that the partial block on the failed datanode will be deleted if the failed datanode recovers later on

47

## Write Failure Recovery – Step 2

- Failed datanode is removed from the pipeline and the remainder of the block's data is written to the two good datanodes in the pipeline.
- Namenode notices that the block is under-replicated, and arranges for a further replica to be created on another node.
  - Subsequent blocks are then treated as normal
- To overcome multiple datanode failures, block are asynchronously replicated across the cluster until its target replication factor is reached

48

# Hadoop Coherency Model

- A coherency model for a file system describes the data visibility of reads and writes for a file.
  - HDFS trades off some POSIX requirements for performance
    - Different behavior than expected in typical POSIX environments may be observed
- Hadoop coherency model semantics
  - After its creation, the file is visible in the file system namespace
  - Any content written to the file is not guaranteed to be visible, even if the stream is flushed.
    - File appears to have a length of zero

49

# Hadoop Coherency Model – Effect and Remedy

- The first block becomes visible to new readers only after more than a block's worth of data has been written
  - Same holds for subsequent blocks – it is always the current block being written that is not visible to other readers.
- HDFS sync() method on FSDataOutputStream forces all buffers to be synchronized to the datanodes
  - If sync() is successful, HDFS guarantees that the data written up to that point in the file is persisted and visible to all new readers.
    - A crash of a client or HDFS does not cause data loss
  - Behavior is similar to the fsync system call in Unix that commits buffered data for a file descriptor

50

# Conclusion

- Hadoop Design Issues
- Hadoop Data Flow
  - Read() and Write =() Operations
- Hadoop Replication Strategy
  - Hadoop Topology
  - Distance Metric
  - Replica Placement
- Hadoop Coherency Model

51

# Reference


- Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer.
- MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat,
- The Google File System, Sanjay Ghemawat, Howard Gobioff, and Shun-TakLeung,


52