HDFS

What is HDFS ?

- Storage Component of Hadoop
- Distributed Filesystem

Features

- Scale-Out
- High Availability
- Balancing
- Replication
- Fault Tolerant
- High Throughput
- Easy storage of Large dataset
- Security
- Don’t require special hardware. Work on Commodity Hardware

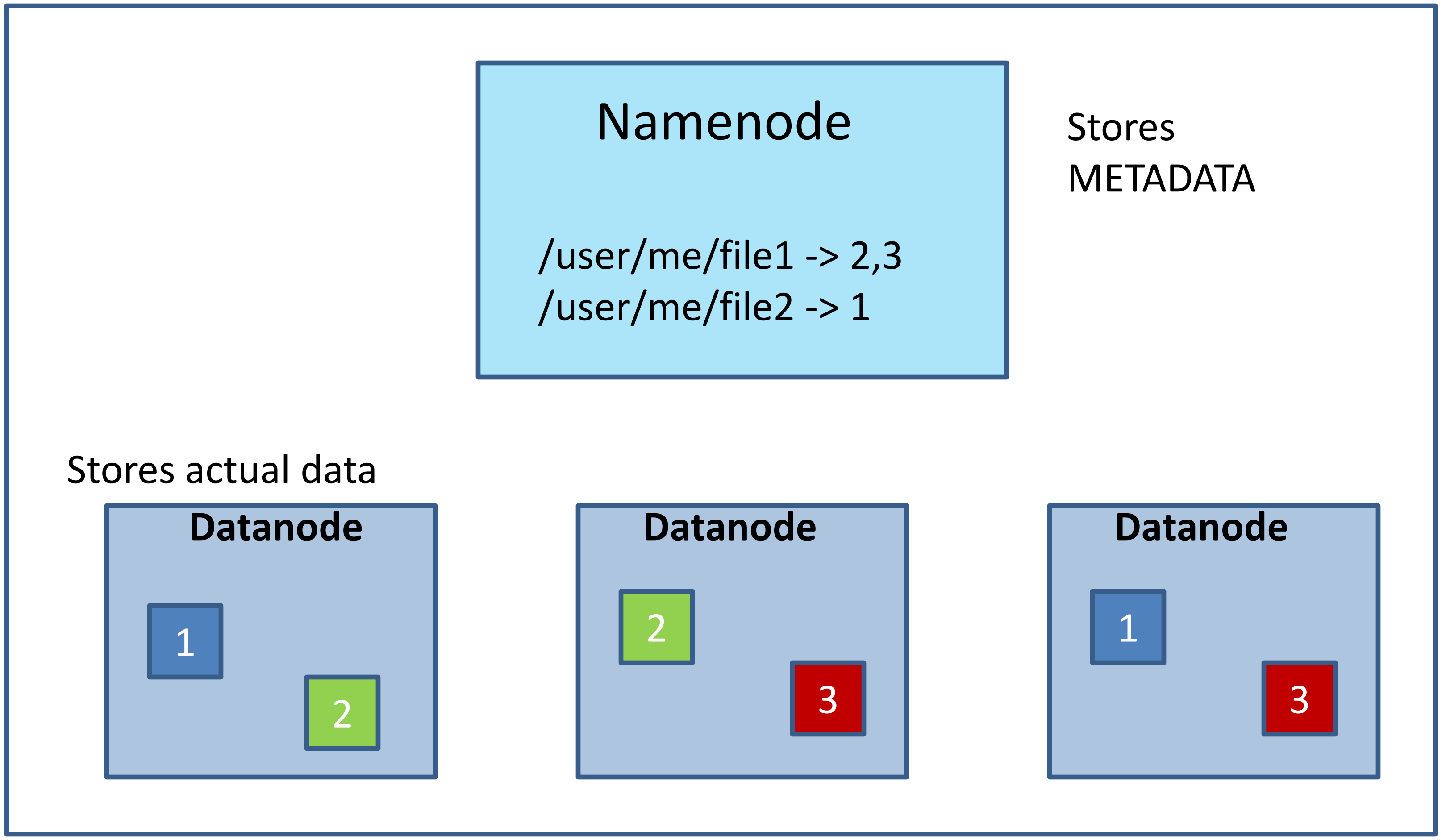
HDFS Basics

- Based on Google GFS
- Written in Java.
- Sits on native filesystem like ext4,ext3
- Works better with Large Files.
- Its “Write Once” and “Read Many” concept
- Optimized for streaming reads.
- Self Replication

Design Assumption and Goal

- Hardware Failure
- Streaming Data Access
- Large Dataset
- Write-once and Read-many Concept
- Transferring Code is cheaper than data.
- Adjust with heterogeneous H/W and S/W

HDFS Architecture



NOTE : Replication =2

Architecture contd.

- Master – Slave Architecture
  - Master : Namenode
  - Slave : Datanodes

- Block-structured Filesystem
- HDFS namespace for user to interact

HDFS Concepts

- Namenode
  - Master of Cluster
  - Only stores metadata. No actual data.
  - Runs in RAM
  - Java Process
  - Critical Machine of cluster
  - It was Single point of Failure. Now HA is present.
  - It should be running all time. Cluster is non-accessible , if namenode is down.
  - Keep record on disk for recovery from any kind of crash.
    - editlogs and fsimage

HDFS Concepts contd.

- Secondary Namenode
  - Its not a BACKUP of namenode
  - Separate deamon
  - Do housekeeping for namenode.
  - Merges edit-logs with fsimage .

HDFS Concepts contd.

- Datanode
  - Java Process
  - Stores Actual Data in blocks
  - Not so critical like Namenode
  - Send heartbeat to NN periodically to remain alive.
  - Send block report to NN periodically.

HDFS Concepts contd.

- Blocks
  - Smallest unit for data storage/read
  - Files are broken into blocks
  - Blocks are stored independent of file
  - Default block size is 64MB

How Files are stored?

- Each file split into blocks
  - Usually 64MB or 128MB
- Data is distributed across datanodes
- Blocks are replicated across the datanodes.
- Master node (Namenode) maintain register which contain metadata of each file.
  - Location of blocks, which datanode , which block for file etc..

Question

- What is Block size of typical Filesystem like ext4 , NTFS?

Question

- What is Block size of typical Filesystem like ext4 , NTFS?

  – Few kilobytes (4KB or 8KB)

Question

- Why HDFS block size much bigger compared to typical Filesystem?

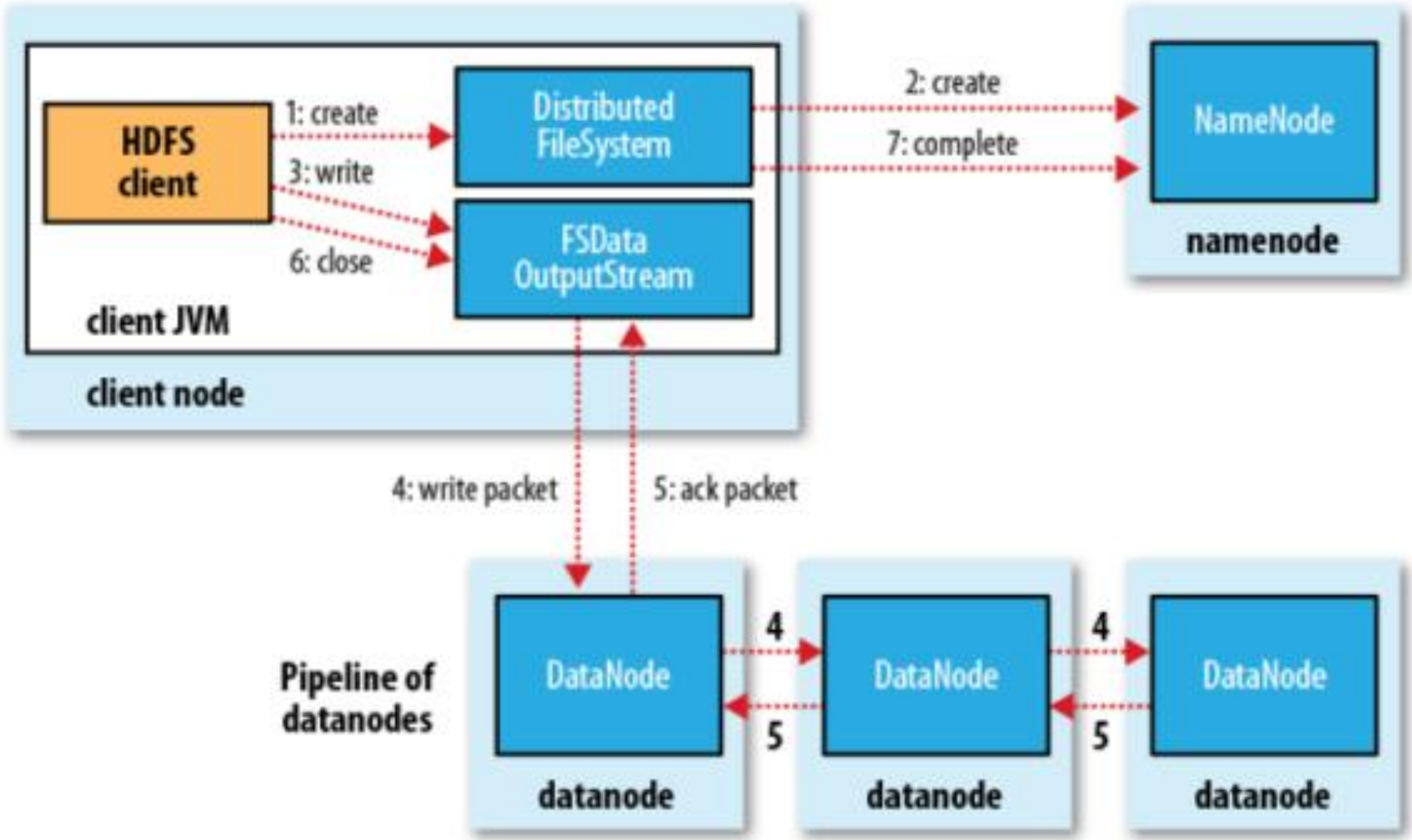
Question

- Why HDFS block size much bigger compared to typical Filesystem?
  - To Reduce cost of seek.

HDFS is not for…

- Low-Latency data access.
- Multiple writes
- Small Files
- File Update

Write Operation



Write Operation explained

- **Step 1:** The client creates the file by calling create() method on DistributedFileSystem.

- **Step 2:** DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it.

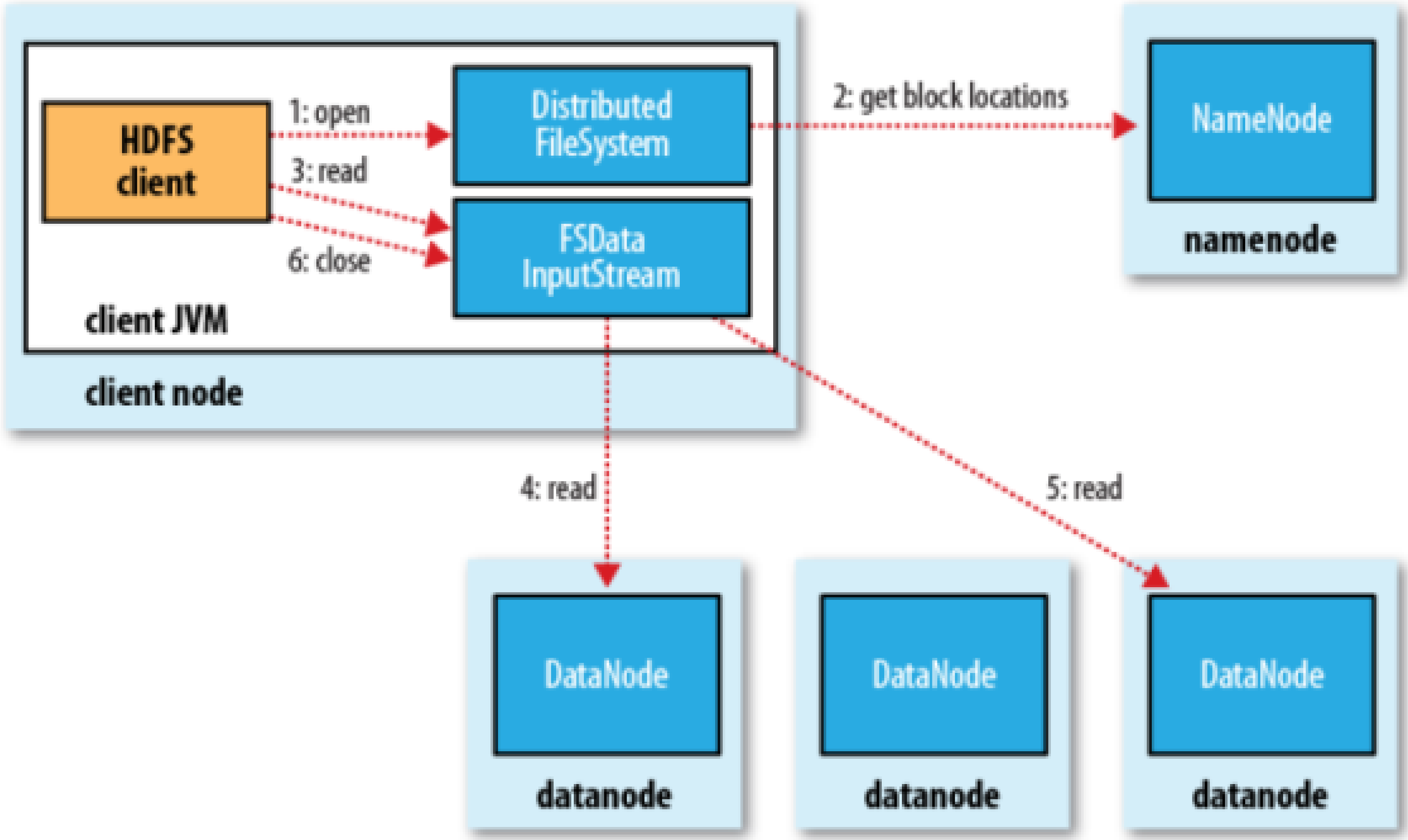
  The namenode performs various checks – file existence, user permissions .If these checks pass, the namenode makes a record of the new file; otherwise, file creation fails. The DistributedFileSystem returns an FSDataOutputStream for the client to start writing data to.

- **Step 3:** As the client writes data, DFSOutputStream splits it into packets, which it writes to an internal queue, called the **data queue**. The data queue is consumed by the DataStreamer, which is responsible for asking the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline, and if the the replication level is three, then three nodes in the pipeline. The DataStreamer streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline.

Write Operation explained

- **Step 4:** Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline.
- **Step 5:** DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the **ack queue.** A packet is removed from the ack queue only when it has been acknowledged by all the datanodes in the pipeline.
- **Step 6:** When the client has finished writing data, it calls **close()** on the stream.
- **Step 7:** This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete .The namenode already knows which blocks the file is made up of, so it only has to wait for blocks to be minimally replicated before returning successfully.

Read Operation



Read Operation explained

- **Step 1:** Client will open the file by giving a call to open() method on FileSystem object.

- **Step 2:** DistributedFileSystem calls the Namenode, using RPC, to determine the locations of the blocks for the first few blocks of the file. For each block, the namenode returns the addresses of all the datanodes that have a copy of that block.

- **Step 3:** The client then calls read() on the stream. DFSInputStream, which has stored the datanode addresses for the first few blocks in the file, then connects to the first closest datanode for the first block in the file.

Read Operation explained

- **Step 4:** Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream.

- **Step 5:** When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block. This happens transparently to the client, which from its point of view is just reading a continuous stream.

- **Step 6:** Blocks are read in order, with the DFSInputStream opening new connections to datanodes as the client reads through the stream. It will also call the namenode to retrieve the datanode locations for the next batch of blocks as needed. When the client has finished reading, it calls close() on the FSDataInputStream

Interacting with HDFS

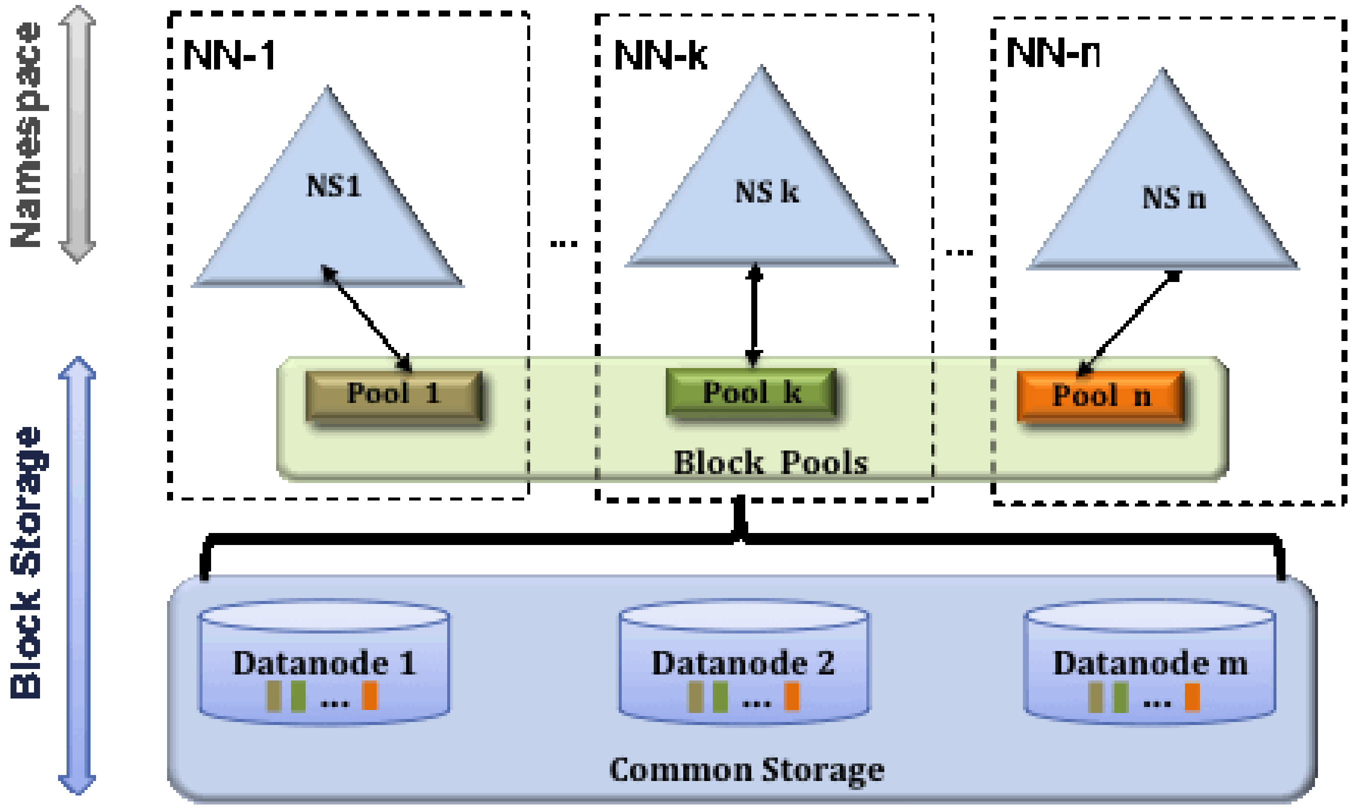
- HDFS CLI
- JAVA API

HDFS Limitations

- Single Namenode
  - Keeps all metadata in RAM.
  - Performs all metadata related operations
  - Single Point of Failure (SPOF)

HDFS Federation

- Partition filesystem namespace over multiple Namenode



HDFS High Availability

- Secondary NN and HDFS Federation does not solve issue of NN SPOF.

- HDFS HA introduce pair of NN
  - Active NN and Standby NN

- If Active NN goes down (expected or crash) , StandBY NN takes control

- Datanode send block report to both NN.