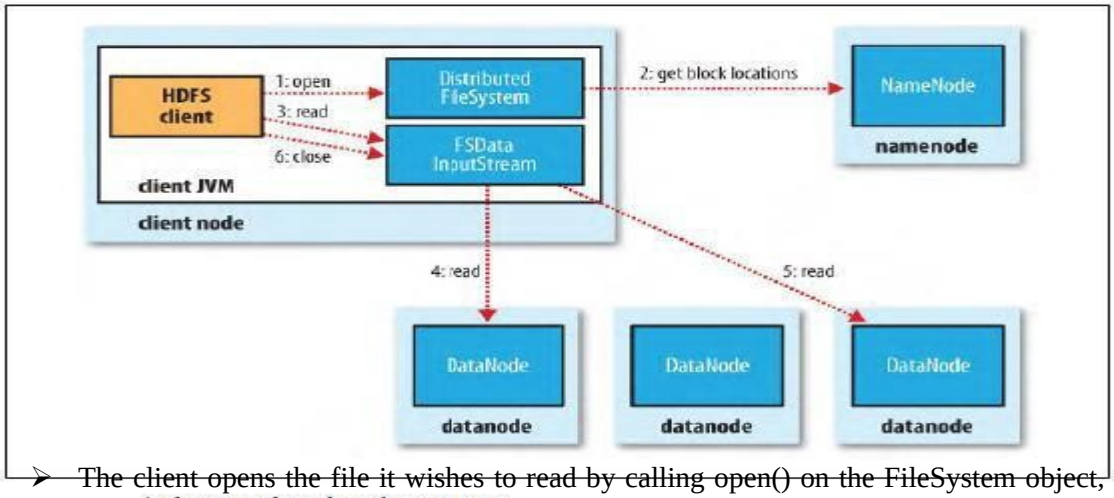
- The system employs a range of fencing mechanisms, including killing the namenode's process, revoking its access to the shared storage directory, and disabling its network port via a remote management command.

4.8 DATAFLOW OF FILE READ & FILE WRITE.

Part-B	1.Illustrate data Flow in HDFS during File Read/Write Operations with Suitable Diagram(AU/Nov/Dec 2017)

DATA FLOW OF FILE READ

- To get an idea of how data flows between the client interacting with HDFS, the namenode and the datanode, consider the below diagram, which shows the main sequence of events when reading a file.

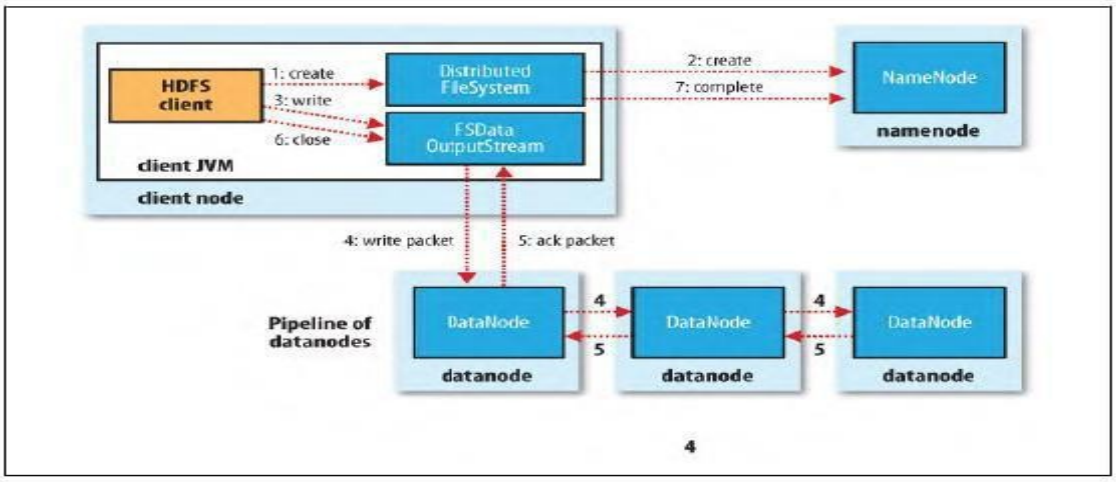


A client reading data from HDFS

- The client opens the file it wishes to read by calling open() on the FileSystem object, which for HDFS is an instance of DistributedFileSystem (step 1).
- DistributedFileSystem calls the namenode, using RPC, to determine the locations of the blocks for the first few blocks in the file (step 2).
- For each block, the namenode returns the addresses of the datanodes that have a copy of that block. Furthermore, the datanodes are sorted according to their proximity to the client. If the client is itself a datanode (in the case of a MapReduce task, for instance), then it will read from the local datanode.
- The DistributedFileSystem returns a FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the datanode and namenode I/O. The client then calls read() on the stream (step 3).
- 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. Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream (step 4).
- 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 (step 5).

- This happens transparently to the client, which from its point of view is just reading a continuous stream. 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 (step 6).
- One important aspect of this design is that the client contacts datanodes directly to retrieve data, and is guided by the namenode to the best datanode for each block. This design allows HDFS to scale to large number of concurrent clients, since the data traffic is spread across all the datanodes in the cluster.
- The namenode meanwhile merely has to service block location requests (which it stores in memory, making them very efficient), and does not, for example, serve data, which would quickly become a bottleneck as the number of clients gr**ew.**

DATAFLOW OF FILE WRITE



A client writing data to HDFS

- The client creates the file by calling create() on DistributedFileSystem (step 1).
- DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it (step 2).
- The 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 these checks pass, the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an IOException.
- The DistributedFileSystem returns a SDataOutputStream for the client to start writing data to. Just as in the read case, FSDataOutputStream wraps a DFSOutputStream, which handles communication with the datanodes and namenode.
- As the client writes data (step 3), DFSOutputStream splits it into packets, which it writes to an internal queue, called the *data queue*.
- The data queue is consumed by the DataStreamer, whose responsibility it is to ask the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline—we'll assume the replication level is 3, so there are 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. Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline (step 4).