Paxcel Technologies Pvt. ltd.

HDFS

Distributed File System

Part 1 (Exploring HDFS internals)

Kuldeep Saxena
[04 July, 2010]

Table of Contents

1 OVERVIEW

2 HDFS ARCHITECTURE AND COMPONENTS

3 COMPONENT DETAILS

a) PROTOCOL FORMAT

b) NAME NODE

i) Name Node Startup

ii) Storage

iii) Request Processing

iv) Services Provided by Name Node

1) *Client to Name Node communication*
2) *Data node to Name Node communication*
3) *Secondary name node to Name Node communication*

c) DATA NODE

i) Data Node Startup

ii) Services Provided by Data Node

iii) Storage

d) File System

4 COMPONENT COMMUNICATION/FLOW

a) Write Operation
b) Read Operation
c) Other Operations

~~*5 SETUP HDFS*~~

~~*6 Hadoop Utilties for HDFS*~~

~~*7 JAVA API EXAMPLES*~~

~~*8 MONITORING*~~

~~*9 ADMINSTRATION*~~

Above red color strike through topics are parts of next paper.

Please note that the information provided in this paper is mainly from source code available of HADOOP version 0.20. There is possibility of partially available source code or improper understanding by the author of this paper. Also contents provided here only applicable to HADOPP 0.20 release.

OVERVIEW

Before Starting on HDFS, here is a brief introduction of Hadoop project (of which HDFS is a sub project)

Hadoop Project was designed to provide Distributed Services, with these main features in mind:

1) Reliability
2) Scalability
3) Distributed Service which run on commodity hardware
4) High throughput on large data set

For different needs of distributed computing Hadoop have different sub projects, HDFS for storage of data in distributed environment, Map Reduce for processing on large data in distributed environment, HBase, a database for storing large table data in structured way, Zookeeper as a coordination service which application can use while running in distributed environment.

***** There are more sub projects for different requirements a distributed application requires. These projects are not discussed here, as this paper series concentrates on above mentioned 4 subprojects – HDFS, Map Reduce, Zookeeper and Hbase. Further, this particular paper discusses HDFS only; other projects will be covered in their respective papers.***

We have data and we have to store it for future processing, If we use a normal file system(dedicated to a single node/system) to store data in distributed application environment, that has many problems:

1) Congestion at the disk node, as all of our application running in distributed has single point of data access, this degrades performance a lot
2) Failure of disk can corrupt data, you can have backup of your data but that need manual operation to put that back and bringing back this data takes time, during that time your application has to wait, which is very critical in production applications.
3) Some databases provide distributed clusters to manage replication and availability, but Hadoop use data locality for fast processing; also size of HDFS file system is large to process on big chunk of data, where in database clusters if you have large data there is good possibility of data distributed across different nodes in clusters, makes some of operation on data difficult, like Join query operation in a database cluster environment. HBase (build over HDFS) make these operations easy with structured data storage. Also administration of database cluster is difficult.

We can have different systems (disk on different machine) to store data; in that case it is difficult to synchronize data on different system without a proper system in place. This is where distributed file system come into play, which provide user transparency and perform many operations behind the scene.

HDFS is such a system, which is used to store data on distributed system spread across network. HDFS has following features:

1) HDFS stores data in form of Block (typically a block of 64MB, a block corresponds to one file in file system). This block size can be increased to take advantage of high

processing disk drives. Large block size gives advantage of fast disk read, and in many situations you will find most of data you currently working on fit in a single block.

2) Replication of data, HDFS replicates data intelligently to provide – failure management, fast access to data.
3) The operation performed by HDFS is transparent to user, for user it is same normal file operation, all management operations are done by HDFS behind the scene. User has no knowledge of which replication copy he is working on.
4) Single Writer – HDFS has only a single writer to write to a file, it does not provide random writes to a file or file write by more than 1 process. All update are written at the end of block.
5) A file can have more than one block, but 2 files not in same block, this makes operation easy for HDFS to find and manipulate data.
6) Scalability – HDFS provides linear scalability, if you see your data nodes (machines/drives which used to store data) are filling up, you can add up more data nodes as needed, you don't have to restart existing application, newly added data node will become part of cluster and will start performing operation instantly.

HDFS ARCHITECTURE AND COMPONENTS

Basic components of HDFS are shown in following figure:

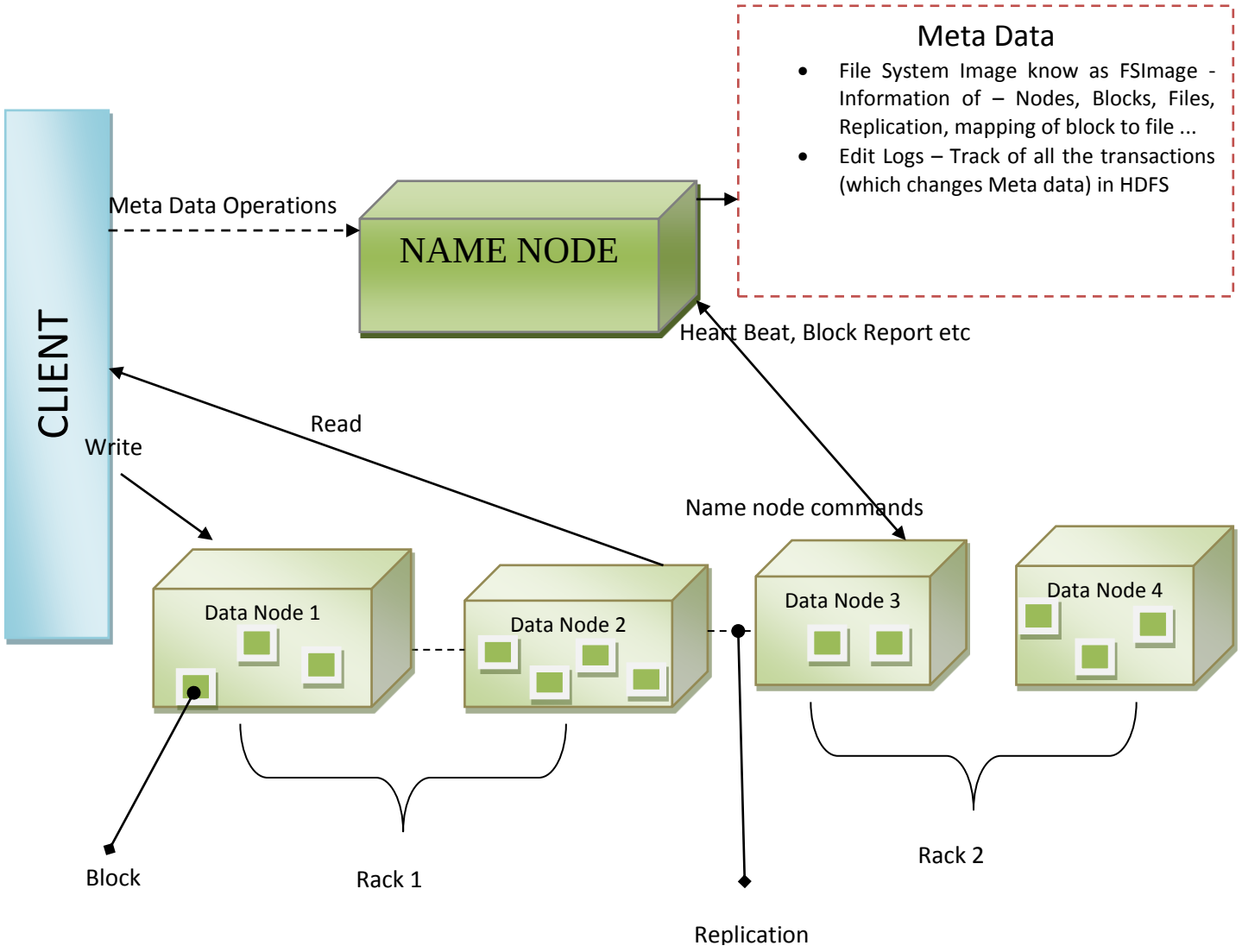

Figure 1: HDFS Components

HDFS has two basic components – Name Node and Data Node.

Name Node – This is the controller or Master of Cluster, which manages all the operations requested by Client. Name Node performs following major operations –

1) Allocation of Blocks to File
2) Monitoring Data Node for Data Node Failure and new Data Node addition
3) Replication Management
4) User requests management – like writing file, reading file etc.
5) Transaction Tracking and logging of transaction

Data Node – This is where data is stored in HDFS, Data node does not aware of what file data belongs, it writes data in local file system in form of blocks. Data node has following major functionalities –

1) Write/Read Block to/from Local File
2) Perform operation as directed by Name Node
3) Register/Heartbeat itself with name node and provide Block report to name node

Meta Data – This contains all information about File System, where blocks related to file are, where blocks are replicated, number of replication copy, where data nodes running, what space available on each, directory structure etc. It also keeps tracks of transaction happening in system in Edit Log.

Client – Client shown above is the one which is communicating to HDFS using predefined API, it could be FS utility provided in Hadoop, Java API provided by Hadoop, or some other API to communicate to Hadoop in language independent manner. In the following section we will be covering flow of different operation, where each component is coming into picture during operation that will clear out how it works internally.

Components Details

In this section we will be exploring major components of HDFS, Name Node and Data Node. And then will see how a File System uses the protocol to interact with these and provide services to users.

Before getting in to any of component, it is worth to see the following figure- it represents the protocol supported at each component.

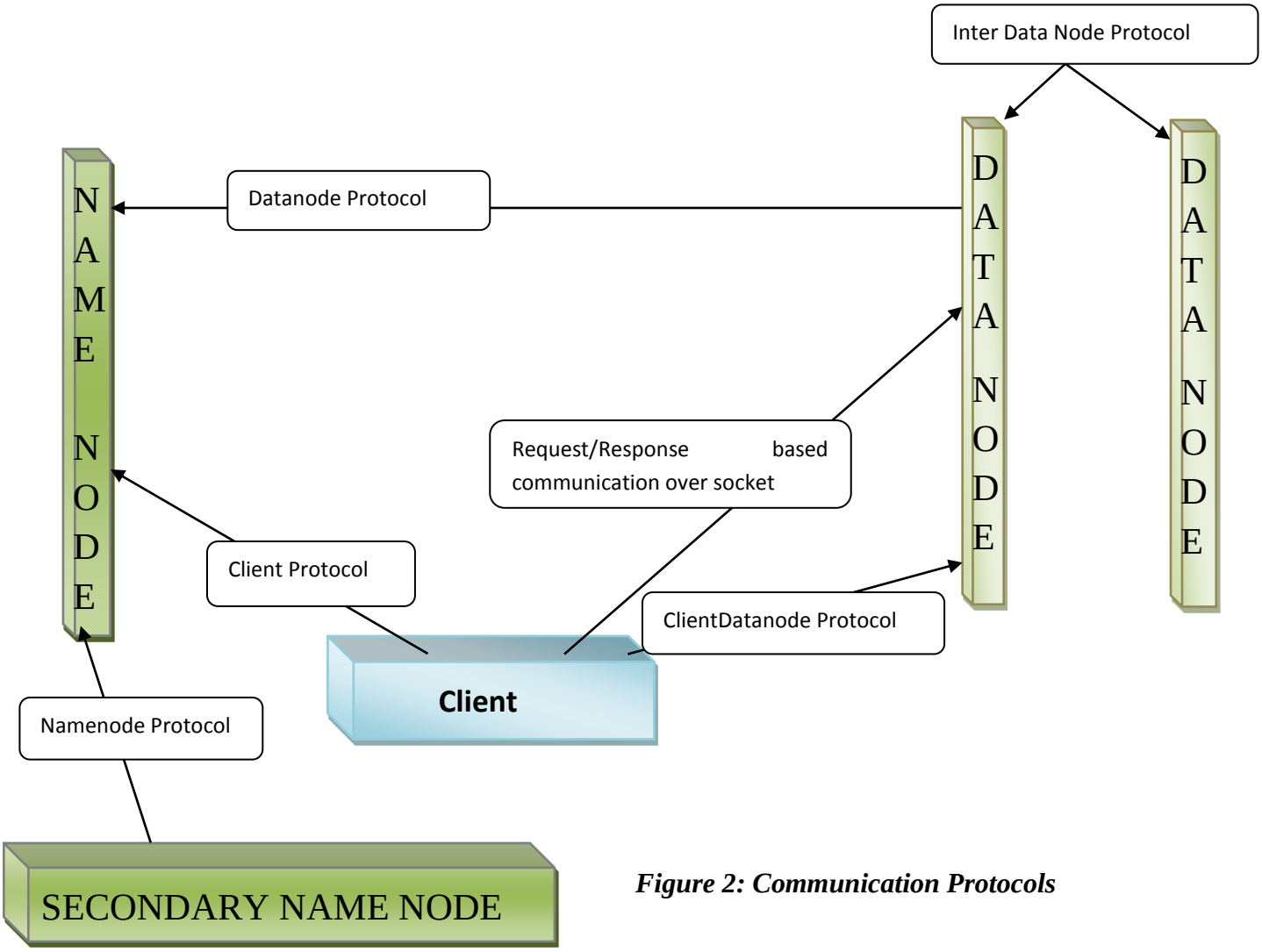

Figure 2: Communication Protocols

PROTOCOL FORMAT

HDFS uses RPC as communication protocol, build upon TCP/IP. Following is the format of a RPC communication –

<Protocol Header> <Connection Header><Data>

- Protocol Header – consists of 2 things –
  - Communication protocol to use – 4 bytes – value "hrpc"
  - Version number – 1 byte – current value 3
- Connection Header consists of following –
  - Header Length
  - Protocol String Length + Protocol String

    - True/false user group information present?
    - If user group information present <length of next string> <string>– UGI_TYPE – value STRING_UGI
    - <length><user name>
    - Number of groups for each group
    - <length> <group name>
- Data – data part contains user request –
    - <id> - All communications in HDFS works on Non-Blocking Network I/O, this id is used to identify user request and response.
    - <data length>
    - <Data> - marshaled data, method, parameter etc.

A complete example of protocol –

<hrpc><3> <length 4 bytes><protocol string length 4 bytes java int><protocol name><true><length for string><STRING_UGI><length username><username><numberof groups><length for next groupd name><group name> <length><call id><method name length UTF8 2bytes><methodname><number of parameters><parameter type length><parameter type> …..

NAME NODE

Name node exposes its services to data node, client, and secondary name node using different protocols. It is the responsibility of different components to communicate with Name node according to rules and perform operations. Name node can also be viewed as a repository for HDFS file system which can be queried to fetch File System information. Name node never initiates any action to other components; it returns information (or command) as response to the request from components. Following sections explores some internals of Name Node –

Name Node Startup

> ** Note: Here we are not exploring configuration and setup; it is part of next paper. We only providing internal details of how Name Node works after the proper setup done and it is running.

Name node can be started with different parameters; however we are not going to discuss the parameters and their effect, because it is part of next paper where we will see how to setup

HDFS. Here we assume everything is setup properly and we started HDFS (using start-dfs.sh) which starts Name Node and other servers. Name node should be formatted, don't confuse with format, it is setting up file structure for name node. For now you can assume all things are in place and system is up and running.

Following things happens when a Name Node Starts –

1) It Loads Default Configurations and your modified configuration
2) Then it creates following servers to process user request –
    a. RPC Server – This is custom protocol implementation for different components. Based on different protocols clients (data node, File System client, Secondary name node can communication to it).
    b. Http Server to provide monitoring information
    c. Trash Emptier to manage user temporary files
3) Initializes different metrics and MBeans for monitoring
4) Initializes Name System – we will see Name system and FS Image (Meta Data) in Storage section
5) In turn of RPC Server startup – following Threads are started –
    a. Listener – This thread listen for any incoming request
    b. Handler – This thread handles user input request and provide output to the Responder
    c. Responder – This thread is responsible to respond to user request
6) In turn Name System starts following threads (which are internal to FSName System) –
    a. Heart beat monitor – checks whether data node is dead or running
    b. Lease manager monitor – checks lease associated with client has expired or active
    c. Replication monitor – monitors replication of blocks and replicate block as needed
7) At start up Name Node remains in Safe Mode – Safe Mode is actually read only mode for the HDFS file system, the reason for being Safe Mode is block to machine mapping is created each time Name node starts and it is done by using data coming from different data nodes, It remains in Safe mode until it has appropriate block information which stratifies the replication etc.
8) It starts processing Client request after exiting Safe mode.

STORAGE

Let's see what is stored at Name Node –

Name Node contains following information to process client and data node requests –

1) File related information like name, replication factor etc
2) Block mapping for the files names
3) List of Data nodes available
4) Blocks to data node mapping
5) Network metrics
6) Edit Logs

At startup name node load FS Image (this is subset of above mentioned information), and edit logs. Name node does not stores Data node specific information, it stores Directory structure, File information (Replication, Modification time, access time, Block Size, total blocks, for each block – block id, number of bytes in block, generation timestamp), Under construction file information (under construction file is which is allocated to client with lease, but not completed yet), File Security permission.

Name node also contains Edit Log, which contains information of all the transaction performed on the File system, at load this transaction performed on FS image and new FS image is written to system and starts operation with empty edit logs.

Name node stores file structure in memory; the structure is similar to normal file system, See following figure –

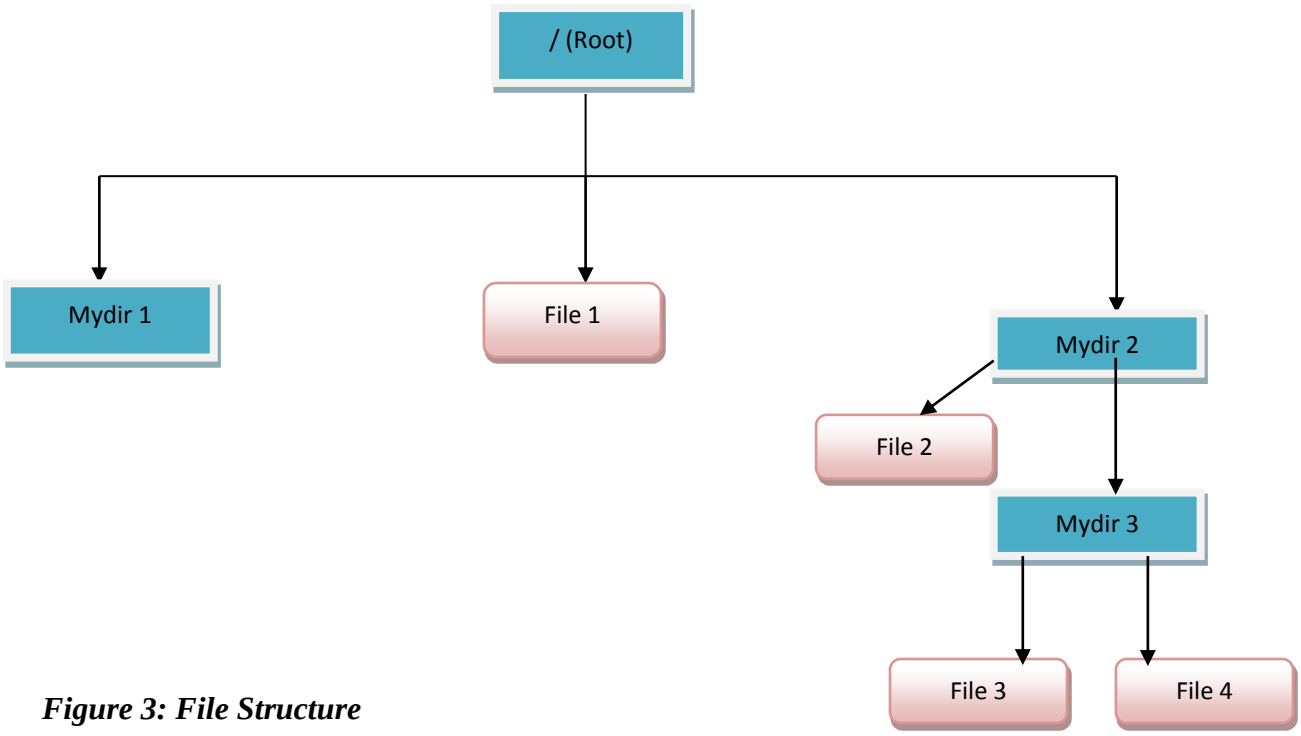

Figure 3: File Structure

Name node stores Meta information and edit images in file. You can find files under name node storage directory –

Following are main files –

Directory Name	Files
current	• **fstime** • **edits** • **fsimage** • **VERSION**
image	• **fsimage**

fstime – This file contains check point time, this file is created again with new checkpoint time at each check point event. It is an 8 byte long value

edits – This is edit file, it contains all the information of transaction occurred in system.

Following transactions are logged in edit file –

- OP_INVALID – invalid code to verify read of edit file
- OP_ADD – add open lease record to edit log
- OP_RENAME - rename
- OP_DELETE - delete
- OP_MKDIR - make directory
- OP_SET_REPLICATION - set replication
- OP_DATANODE_ADD (for backward compatibility)
- OP_DATANODE_REMOVE (for backward compatibility)
- OP_SET_PERMISSIONS – set permission for a file
- OP_SET_OWNER - owner
- OP_CLOSE - close after write
- OP_SET_GENSTAMP – generation time stamp
- OP_SET_NS_QUOTA (for backward compatibility)
- OP_CLEAR_NS_QUOTA (for backward compatibility)
- OP_TIMES - modification and access time
- OP_SET_QUOTA – sets name and disk quota

fsimage – This file contains a File System Image (fsimage) of HDFS system. However you cannot figure out what all are content at looking the file, these are the contents written

- Layout Version – Layout version
- Name Space ID – Name space id
- Number of Items in the root – number of child in root directory
- Generation Time Stamp – File System Generation time stamp
- length of file/dir name (next string to be written length) 0 for root, then it will be 1+dir/file name length … here 1 used for Path separator
- write file/dir name, then following properties for file
    - replication – 0 for (directories)
    - modification time
    - access time – 0 (for directories)
    - preferred block size – 0 (for directories)
    - number of blocks - -1 (for directories)
        - for files – blockid, number of bytes, generation time stamp is written here
    - NS Quota – name space quota
    - DS Quota – disk space quota
    - user name
    - group name
    - permission
- Print recursively this information for each file in directory and sub directories
- Then it also writes any under construction files –
    - path of file
    - replication
    - modification time
    - preferred block size
    - number of blocks
    - block information – block id, num bytes, generation time stamp
    - username, group name, permission
    - client name
    - client machine

VERSION – This file contains namespace information. Entries you can see in this file –

a. namespaceID – Name Space ID, this ID is assigned at time of formatting of Name node, and all data node will get same namespaceID when connected to Name node.
b. cTime – Creation Time – currently 0
c. storageType – Storage type (NAME_NODE)
d. layoutVersion – Version (-18), decremented with each new release of HDFS

fsimage (image directory) -The file under image directory is of no use, it is used to overcome compatibility issue of old version. It is created when format or upgrade is done.

REQUEST PROCESSING

Name node only responds to the user request, it never initiates a request. Name node implements different protocols to communicate different component. It is worth mentioning that all protocols are built on TCP-IP which Non-Blocking I/O. At start up name node expose its services using these protocols. All communication within HDFS uses RPC mechanism and Client uses Proxy objects to communicate to Name Node which built upon Non-Blocking I/O. Following figure shows a basic request processing at Name Node –

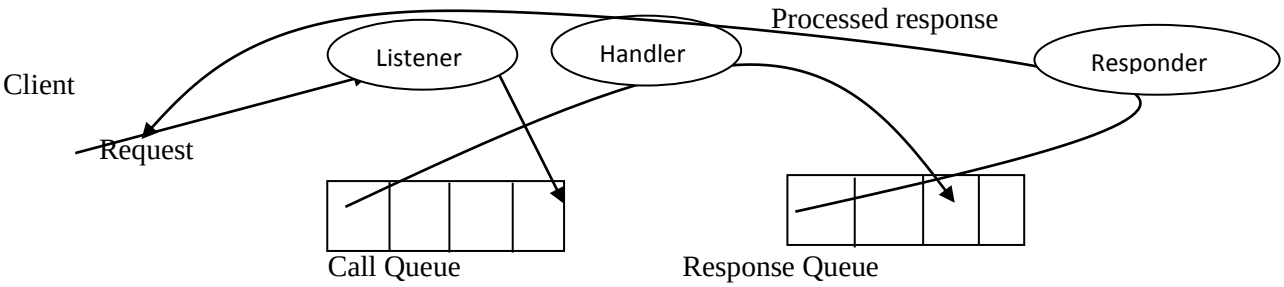

Figure 4: Request processing at name node

Above image have three major components and 2 queues

1) Listener – this listens for any new request from the client, listener put client request to call queue
2) Handler – handles client request, processing it and updating response queue
3) Responder – this responds to client after taking element from response queue

Following section details what operations name node provides to different components –

Services Provided by Name Node

- **Client to Name Node communication**

Client communicates with Name node using Client Protocol implemented by Name Node. Following are main operations Client performs on Name Node (we are not giving API details rather focusing on operation objectives) –

a) Locating Block – Name node can be queried by the Client to locate block related to given file. Name node provides information block along with data nodes which contains block. This information is then used to communicate to data nodes to fetch actual data.
b) File Create – This is used to create a new file in File System, Name node creates file in repository and add a new lease to the client. Further block are added by client request. To avoid issue of unexpectedly die of client is managed by renewing the lease call by client periodically.
c) Block creation – After a file created, name node need block to write data to, it then request Name node to provide block to write to, Name node uses Data node statistics and returns data nodes to the client with block information, similar to locating block operation but this time new block are created and this for writing purpose.
d) Complete – this to mark Name node that it finished writing to file (a form of close), and name node perform operation like release lease, create permanent INode(file node of storage) entry, see replication done etc.
e) Bad blocks – Client can report corrupted blocks it found during its operations on data node. Name node then replicates that corrupted block to other data node.
f) Data node report – Client can query all the data nodes available in current HDFS system.
g) Block Size – Client can query block size for a given file.
h) Safe Mode – Client can query Name node status, whether in safe mode or not, also it can set mode to safe or reset it.
i) Name Space – Client can ask name node to save namespace image and reset edit logs.
j) Other Meta data operations – Client can create directory, delete file/directory, rename file/directory, query file status,

- **Data node to Name Node communication**

Data node communicates with Name node using Data Node Protocol. Following are main operations which Data Node performs on Name Node –

a) Registration – When a data node want to be part of HDFS system, it need to register itself with Name node. Name node registers data node and allocates storage id (if not already had) and updates with registration id for further communication.

b) Block report – Data node tells name node about locally stored blocks and name node can send Data node command in response of this call.
c) Block received – when a client writes to data node, and a block is completed, data node notifies that it received a block; this is also called when block is replicated to data node by another data node.
d) Bad Blocks – data node can tell name node about corrupted blocks it found.
e) Generation time stamp – Data node can query for the generation time stamp.
f) Version information – Current HDFS namespace version
g) Block Synchronization update – To tell name node update of block synchronization

- **Secondary name node to Name Node communication**

We have not discussed Secondary Name node yet, neither we suppose to explore that in details, for general understanding – A secondary name node works as copy of Name node, it maintains part of state of name node, secondly it is not which automatically switch to when Name node failed, it needs manual interaction to setup this as name node. However we shows what operation available to secondary name node –

a) Blocks Information – Using this protocol name node can be queries to get blocks available on a given data node up to the size.
b) Edit log size – Size of edit log
c) Edit log roll – Rolls a new edit log to the name system
d) FS Image roll - Rolls a new image to the name system

Secondary name node uses Name Node http API to copy FS Image and edit log from the name node.

DATA NODE

Data node is where actually data storage is done, Data node creates file for block as requested by Client and updates its information to Name node. Data node implements InterDatanodeProtocol to offer service to other data node and ClientDatanodeProtocol to offer services to Client. However, much of communication is done over direct socket communication to speed up communication and avoid delay caused by using Proxy objects. Data node manages block checksum, chunk checksum (within a block file), and update Name node with Block Report. Data node does not know what a file is, it stores blocks in local file system as a file for each block, at startup it reads data directory and restores all the blocks it has and updates Name node with block report, name node uses this information to process client requests. It is always possible that one data node contains some parts (blocks) of your file and other data node contains

other blocks, it is all Name node which stores information about the file and corresponding blocks with proper sequence and File system to read blocks directly from data node.

DATA NODE STARTUP

Data nodes can be started by start-dfs.sh script, and can be added at runtime. When a data node starts below sequence is followed –

1) It Loads Default Configurations and your modified configuration
2) Check if data directory is available or can be created and setup them
3) Initiate Name node connection, after fetching Name node information from configuration
4) Fetch Name space information from name node, this required to verify that version is same in HDFS.
5) Format data directories if not formatted
6) Load Block information from data directories
7) Create RPC instance which provide implementation of InterDatanodeProtocol and ClientDatanodeProtocol.
8) Starts thread which sends Heart beats, block reports and notifies for block received. Block received instruction (when a block is successfully received either from a client or as replication request) does not conveyed to name node as soon block is received, it is added to received block list and this thread send this notification as it get time.
9) Resisters beans for monitoring and starts HTTP server
10) Process data node commands received as response of heart beat message. Followings are main commands returned to data node –
    a. DNA_TRANSFER – transfer copy of blocks to another node
    b. DNA_INVALIDATE – this command normally in response of next point #10, this tell data node to delete/invalidate blocks from local system which are obsolete.
    c. DNA_SHUTDOWN – Shutdown current data node
    d. DNA_REGISTER – Name node needs registration
    e. DNA_RECOVERBLOCK – Recover blocks using target data nodes
11) Process Data node command received as response of block report, this response contains data node to delete/invalidate blocks which are obsolete.
12) Start Block Scanner if configured.
13) Start DataXceiverServer to listen for client requests
14) Start DataXceiver for each client connection to process request

Services Provided by Data Node

Data node implements InterDatanodeProtocol and ClientDatanodeProtocol, but most of the communications done over socket connection using data header as command to speed up transfer

of data from and to data node. We will see what these 2 protocol offers, here is what can be issued to data node over socket –

OP_READ_BLOCK – This command is used to initiate read block request with data node – to read block data node initiates a BlockSender thread.

OP_WRITE_BLOCK – This command is used to initiate write block request to data node – to process this command data node initiates BlockReceiver thread

OP_READ_METADATA – this command is used to return Meta data information for a block

OP_REPLACE_BLOCK – This command copies block from given (source) node to itself and updates name node with received block with hint to delete source node block. This is used in balancing purpose.

OP_COPY_BLOCK – this command is used to copy a block to destination data node

OP_BLOCK_CHECKSUM – this command is used to get checksum for a block.

Data node also offers services via two protocols – InterDatanodeProtocol – for other data node, and ClientDatanodeProtocol to client (File System) API

1) InterDatanodeProtocol – As mentioned Data node does all transfer operations using direct socket read/write, it provides other data node protocol to query and update block details – it offers only 2 services –
    a. Get Block Meta Data Information – Other data node can query for Meta data information of given block. This is used in data block recovery process, which is initiated as a response of heart beat from name node
    b. Update Block – this is used to update data block with new generation time stamp. This is used in data block recovery process.
2) ClientDatanodeProtocol – This protocol only offers one service –
    a. Recover Block – This is similar to above command name node can give DNA_RECOVERBLOCK, it does same for a single block and returns the data block recovered.

STORAGE

Data node stores actually file data. One file is created for each block. It has no knowledge of what file block belongs, that is managed by Name Node, on query name node directs clients where data can be found and client communicates directly to data node. Following main files can be found at data node's data directory –

Directory Name	Files
current	• **block files** • **block meta files** • **block sub directory** • **VERSION**
detach	**Copy of block in case of resend**
previous	**This directory contains copy of data in case of rollback**
tmp	**Temporary files stored during operations**
$data dir root	• **in_use.lock**
$data dir root	• **storage**

block files – this file contains your data, the file name created with following format –

blk_ - to specify if is a block file

then a –

then block id

sample - blk_-4733032870511303655

Meta files – this contains meta data about block file, there is one meta file for each block file

File name format is –

blk_ - to specify if is a block file

then a –

then block id

then generation time stamp

then .meta

This file contains file checksum for the block, which is used to verify block at read

sample - blk_-4733032870511303655_1007.meta

block sub sir – sub directories are created as limit of file in a directory reaches. the format for sub directory is – subdir+<incremented index of parent directory>

VERSION file contains similar information as name node

- Namespace ID – Name space ID, this value should be same for data node and name node
- Storage ID - Storage id, format of data node, an id is provided
- Creation Time (cTime) – Creation time
- StorageType - DATA_NODE – type of storage
- Layout Version - Version (-18), decremented with each new release of HDFS

detach folder – contains copy of block. If the block was successfully finalized because all packets were successfully processed at the Datanode but the ack for some of the packets were not received by the client. The client re-opens the connection and retries sending those packets. this file is copied to detach folder, such that this copy should not affect other copy of block.

tmp folder – this folder contains temporary files created during operations.

in_use.lock file is used to acquire a lock on the storage, this file is created on startup and removed on exit.

Storage file has same data as name node fsimage file of image directory (corrupted data).

We will explore internal communication between data nodes and client to data node when we explore read and write operations.

File System

Distributed File System is the client which interacts with Name node and Data node on behalf of user request. User communicates to HDFS using File System API and normal I/O operation, File System API processes user requests and provide response. User don't need to care behind the scene working, it simply has to do normal file operations using File System API. We will explore read/write request where we will see how File system API is communicating with other components of the system.

COMPONENT COMMUNICATION/FLOW

In this section we will be exploring 2 major operations read and write. And overview of other Meta operations is provided as they don't need much explanation.

1) Write – Create file, write to old file
2) Read – Read data from file
3) Repository Operations – List File/Directory, Block Size, Data Nodes Search etc

We will be covering each operation step by step and will try to present where HDFS components playing what role.

Write Operation –

A user issue a write (create) command either using wrapper APIs or using Command Shell provide in Hadoop. The following flow is executed by Hadoop –

** HDFS .20 (latest version available) does not support rewriting to file, A file once closed is immutable, the only way to write to file is create a new one. This operation was supported in earlier releases; please see online web contents for issue why it has been dropped.

** Also here we not discussing protocol working at Name node or data node, it is already covered.

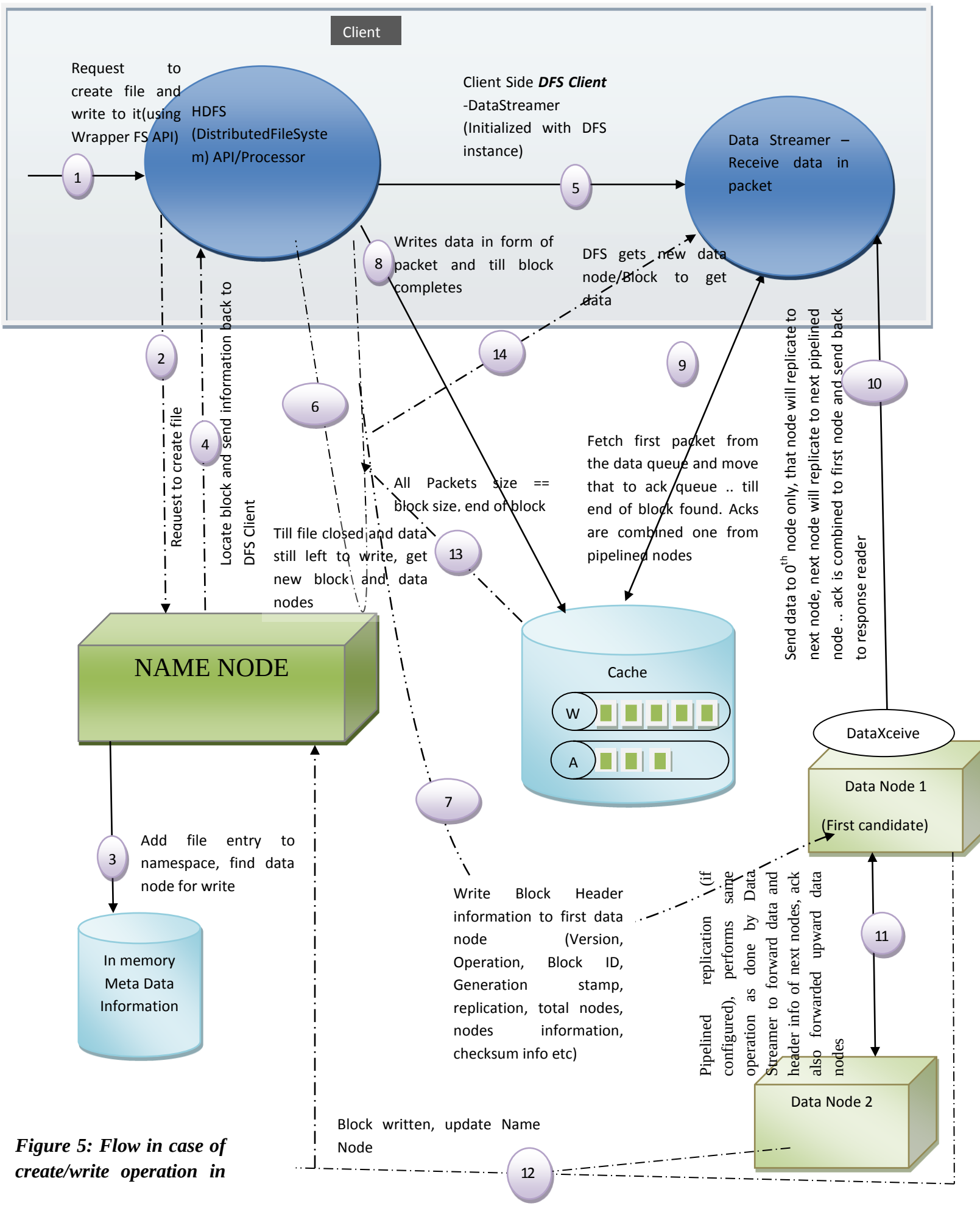

Figure 5: Flow in case of create/write operation in

Before going into details of above figure, briefly discuss terms and component used in above figure –

- **DFS Client** – DFS client is part of default distributed file system implementation provided by HDFS. It does major communication work on behalf of client call, it does this by creating internal threads needed to perform client action. it functionality includes –

  1) Data nodes information is provided to DFS client initially
  2) Packet creation (wrapping user data to small chunks of bytes)
  3) Packet communication
     a. Packet insertion to Data queue
     b. Packet transfer from data queue to target data node (done by DFS Client internal component/thread Data Streamer)
     c. Acknowledgement for packets (done by DFS client internal component/thread ResponseReader)
     d. Update ack queue in case receive of acknowledgment
  4) When a block fills new blocks are allocated with data nodes information to write to, DFS client runs in loop to process any packet it is getting filled in data queue. It has no knowledge of Block, however packet inserted to data queue contains information of whether the packet is last packet of block, in that case DFSClient request for new output stream/data nodes to write packets too. As you can see step 7 of above figure on creation of new block, block header is written to first data node at creation time itself. and DFS Client starts sending data to data node.
- **Packet** – This is used transfer data between data nodes and from DFS client to data node. It is by default 64K of size. Following are the main things a packet contains.
  - Packet length
  - Packet sequence number
  - Offset in block
  - Whether this is last packet of the block? (writes 0 to data node)
  - Data Length (excluding checksum)

  A packet is a small chunk (or subset of block) of data which is used during transmission. And block is what is used to store data and process data. Data node only stores block, client API transfers this block in small chunks called packets.

- **Data queue** – It is where data packets are stored for further transmission. Each element in data queue is a packet, which identified by sequence number. Data Streamer take one packet at a time and send that to data node
- **Ack queue** – Once a packet is transmitted by Data Streamer it is moved to Acknowledgement queue to wait for the acknowledgement of packets.

- **DataXceiver** – This is a thread run for each client/data node connection for data communication, DataXceiverServer waits for client connection and starts DataXeciver thread for the request processing.

Digging deeper (Flow Explained)

Entry point to communicate to HDFS is getting File System. First a client application gets File system by using FileSystem.get (URI, configuration) – this returns the file system to work with; in default case DistributedFileSystem class is returned.

This distributed file system is initialized at this time itself, which does following –

1) Gets Name Node information using URI passed
2) Create a DFS Client object

Next client tries to create file using fs.create (Path) – the path is the path user wishing to give to destination file.

This method returns FSDataOutputStream object to client to write to which is wrapped in FSDataOutputStream. Also at this time, file creation request if processed using namenode call. Name node creates file and provides a lease to it, by setting status of file as under construction, this lease has duration till client can be idle, if lease expiries, file is removed. Client need to renew lease at specified intervals if it is idle due to data node transfer, any further call to name node by same client for same file, renews lease automatically.

DFS Client is initialed and ready for operations. DFSClient starts DataStreamer to transfer user data. Now we done with creating a file, it is ready for read (with no block associated but not for delete and rename).

When a user write to stream provided, it is handled by wrapped stream, this stream converts user data to packet and add that to data queue, and Data Streamer takes packet from queue and send that to the data node.

The data streamer fetches Data nodes information at the time of first write as well as when a block fills. The data streamer works on 2 queues DATA queue and ACK queue. The Data Streamer transfers data (a packet a time) to first data node in the list and wait for combined ACK. It continues this way till all the processing is completed, when a block is filled, it asks for new data nodes from name node.

The data node captures packet and write to the local file system (using BlockReceiver – reading packets and converting to block) and when it finishes a Block it notifies Name node that it completed writing block. For packets Data node forward ack request back to the previous data

node, or if it is the first node in data node list, ack to DFS client (ResponseReader) component for successful processing of the packet.

When block information is written? How data node is storing data in block?

At the time of allocating new data blocks for file, DFS Client gets Block information as well as data nodes Information from name node. DFS Client get data node list to store data to, it uses first node returned to write header data to – the information written here is –

(---- BLOCK Header Information -----)

1) Data protocol version
2) Operation type – (Write/Read … other)
3) Block ID
4) Generation Time Stamp (global time stamp created by name node for every new file)
5) Total nodes length
6) Recovery flag (is this a block recovery)
7) Client
8) Sending source node information? False
9) Replication (nodes - 1) at initial
10) Each node information –
    a. Node name
    b. Storage ID
    c. Info port
    d. IPC port
    e. Capacity
    f. DFS used
    g. Remaining
    h. Last update
    i. XceiverCount
    j. Location
    k. Host name
    l. Admin state
11) Check sum type
12) Check sum size (Bytes per check sum)

As we see here data replication is pipelined, so each data node has responsibility to send and manage (receive data) at its own. when data node receives packets, it sends to next data node,

this way data is written to next data node and so on. Transfer of data from one data node to another is similar as client to data node.

When file is closed, Distributed File System updates Name Node about completion of file.

As you can see data flow never goes through Name Node. Name Node provides information meta data information as response to request.

Read Operation

Following figure shows the flow for the read operation –

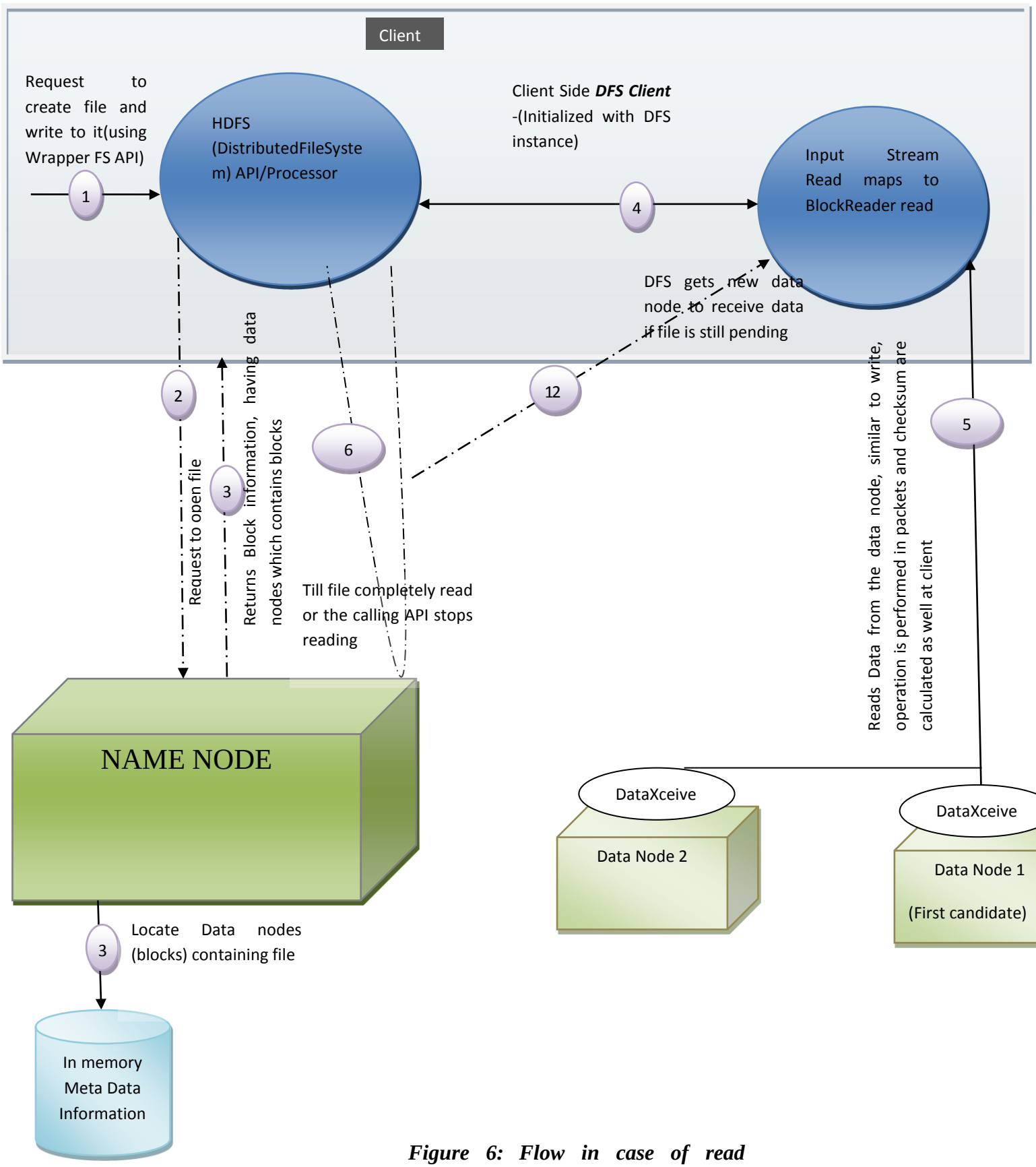

Figure 6: Flow in case of read operation in HDFS

The read operation works by using same components, Name node is used to locate data nodes containing file (blocks), Data node works similarly sending file (block) data in packets to the requested client.

At the startup you need same DistributedFileSystem to interact with HDFS, to do any operation you need to open that file, this is done by getting FileSystem and calling open on that instance, open method does following –

1) Initiates DFS Client for the read request
2) Request for meta data from name node, in response gets sorted data nodes where block can be found (here sorting done on base of nearest data node from client)
3) Provides stream to read data from

User/Client uses normal API method of reading up to byte of data in buffer, behind the scene, DFS Client reads data and fills client buffer up to requested length. This is what performed on read –

1) Target block is located from the returned list (using the current location and data node information)
2) Block reader gets initialized at with the target Block/Data Node information, at the time of initialization block reader open stream to the data node and write following information to the Data Node -
    a. DataTransferProtocol.DATA_TRANSFER_VERSION
    b. DataTransferProtocol.OP_READ_BLOCK
    c. Block Id
    d. Generation Time Stamp
    e. Start Offset to read from
    f. Length of Data to read
    g. Client Name
3) Block reader reads data and performs these operations –
    a) Data read is in form of packet, this is not cached to any queue but rather copied directly to input buffer client provided. The Data node is streaming the data back to the client, the DFS client is reading and performing check sum operation and updating client buffer
    b) Checksum is performed and only data part is copied to the buffer

c) This process continues till user requested bytes are read or EOF is found
d) If data node down at run time or any error, it switches to next data node if any, where replica can be found (name node returned all the data nodes which contains block at initial call)
e) if block is read and file has more data to read, next blocks are requested to read data from (at the time of call to name node for block search name node use configuration property to send number of blocks, this is usually 10, such that frequent call to name node can be avoided, if all 10 blocks are read and still data left new blocks are requested)
f) User/Client is unaware of read operation behind the scene; it reads data as it was written.
g) Once stream is closed block reader and socket is closed

At the Data Node side it is same DataXceiver which manages the operation for read too. It performs following –

It Initiates a Block Sender instance to send data, Block Sender does following –

a. Reads information written by Block Reader
b. See if Data version matches, also file offset and length within the block id provided, and some checks like meta data, checksum
c. It writes status OK if there is no issue otherwise error status is returned
d. Then it writes checksum type and bytes per checksum, also the offset of first checksum
e. Then it starts writing packet to the stream

Other operation performed by Client

Directory create

User can create directory in HDFS, a directory created is actually virtual, and data node does not has any structure for the directory created. This operation is only done at Name Node, Name Node stores Directory and Files in a form of tree, where each leaf is a file. Name Node has logic to find blocks related to file.

Flow –

Entry point to create directory is same as in case of read/write. A FileSystem instance is fetched and method to make directory called on that. Similar to other operation a DFS Client is created on make directory command. This time DFS client only communicated to Name Node to create directory. Name node does validate – valid directory name, directory exists etc and returns with the status success or failure of operation.

Query operations

You can query Name Node for various information like – file/dir listing, file permission, replication etc. Following are the properties available at Name Node about the File System; you can use different method of File System API to get details –

1) File path
2) File length
3) Whether a directory or file
4) Block replication for the file
5) Block Size
6) File Modification time
7) File access time
8) File Count
9) Directory count
10) Directory Quota
11) Space Consumed
12) Space Quota
13) File Permission (user/group/other) – permission of read/write/execute etc. UNIX style
14) File owner
15) File owner group
16) Block Location
17) Used Space in file system

Again these operations are only performed on Name node, working flow is similar to what used for directory create, this time file/directory information is queried from the name node.