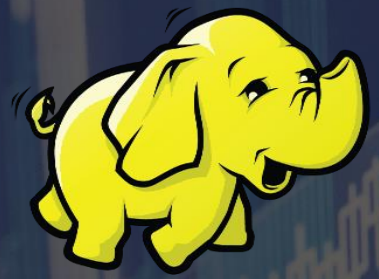
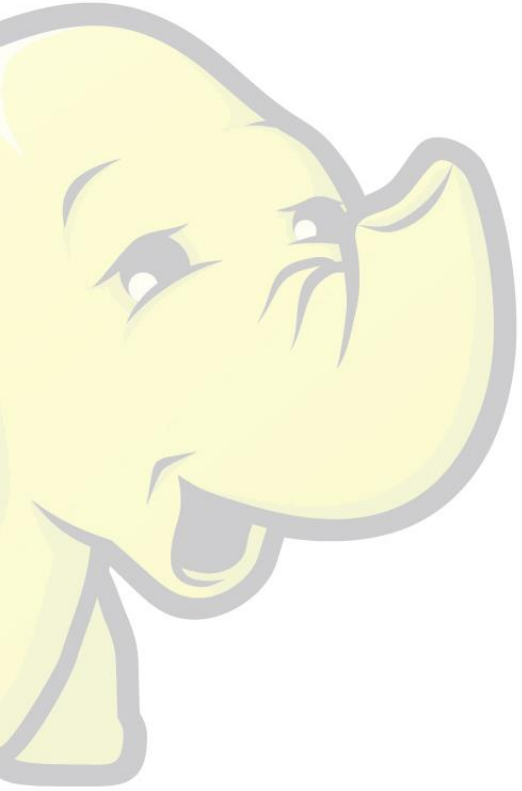
ACADGILD

LEARN. DO. EARN

BIG DATA

DEVELOPMENT





Hadoop Java API's

Session 4

Agenda

ACADGILD

SL NO	Agenda Title
1	Working with Eclipse & IntelliJ
2	Java API to Read HDFS File
3	Demo
4	Java API to Write HDFS File
5	Demo
6	Java API - Listing of File in HDFS
7	Important Java Classes to Read from HDFS
8	Anatomy of File Read from HDFS

SL NO	Agenda Title
9	Data Read Steps
10	Checksum and Data Integrity
11	Data Read from HDFS: Additional Points
12	Important Java Classes to Write Data to HDFS
13	Anatomy of File Write to HDFS
14	Writing File to HDFS: Steps
15	Handling Failures During Writing a File
16	Questions and Answers

Working with Eclipse & IntelliJ

ACADGILD

Demo

Java API to Read HDFS File

ACADGILD

Configuration

- Provides access to configuration parameters.

  Configuration conf = new Configuration();

URI

- Represents a Uniform Resource Identifier (URI) reference

  create(String string)

  Creates a URI by parsing the given string

Path

- Names a file or directory in a FileSystem

Java API to Read HDFS File (Contd.)

ACADGILD

FileSystem

- An abstract base class for a fairly generic filesystem.
- It may be implemented as a distributed filesystem, or as a "local".

  get(URI uri, Configuration conf)

  Returns the FileSystem for this URI's scheme and authority.

  open(Path path)
- Opens an FSDataInputStream at the indicated Path.

  FileSystem fs = FileSystem.get(URI.create(uri), conf);

Java API to Read HDFS File (Contd.)

ACADGILD

FSDataInputStream
Lets the application read from a file

```
FSDataInputStream in = null;
in = fs.open(new Path(uri));
```

IOUtils
An utility class for I/O related functionality.

copyBytes(InputStream in, OutputStream out, int buffSize, boolean close)
Copies from one stream to another.

```
IOUtils.copyBytes(in, System.out, 4096, false);
IOUtils.closeStream(in);
```

Demo

ACADGILD

- HDFS File Read

Java API to Write HDFS File

ACADGILD

FileSystem

create(Path path)

Creates an FSDataOutputStream at the indicated Path

```
InputStream in = new BufferedInputStream(new FileInputStream(localSrc));

Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(URI.create(dst), conf);
```

FSDataOutputStream

Lets the application write a file

```
OutputStream out = fs.create(new Path(dst));
IOUtils.copyBytes(in, out, 4096, true);
```

Demo

ACADGILD

- HDFS File Write

Java API - Listing of File in HDFS

ACADGILD

FileSystem

listStatus()

List the statuses of the files/directories in the given path if the path is a directory

```
Path path = new Path(args[0]);
Configuration conf = new Configuration();
FileSystem fileSystem = FileSystem.get(path.toUri(), conf);
FileStatus[] fileStatus=fileSystem.listStatus(path);
```

Java API - Listing of File in HDFS (Contd.)

ACADGILD

FileStatus
Interface that represents the client side information for a file.

isDirectory()
isFile()
getOwner()
getPath()
getPermission()
getLen()
getReplication()

```
if (fStat.isDirectory()) {
        System.out.println("Directory: " + fStat.getPath());
}
```

Important Java Classes to Read from HDFS

FileSystem

- an abstract file system class.
- provides methods to work with the file system like reading and writing files.

DistributedFileSystem

- an implementation of FileSystem for a distributed file system.

Important Java Classes to Read from HDFS (Contd.)

ACADGILD

FSDataInputStream

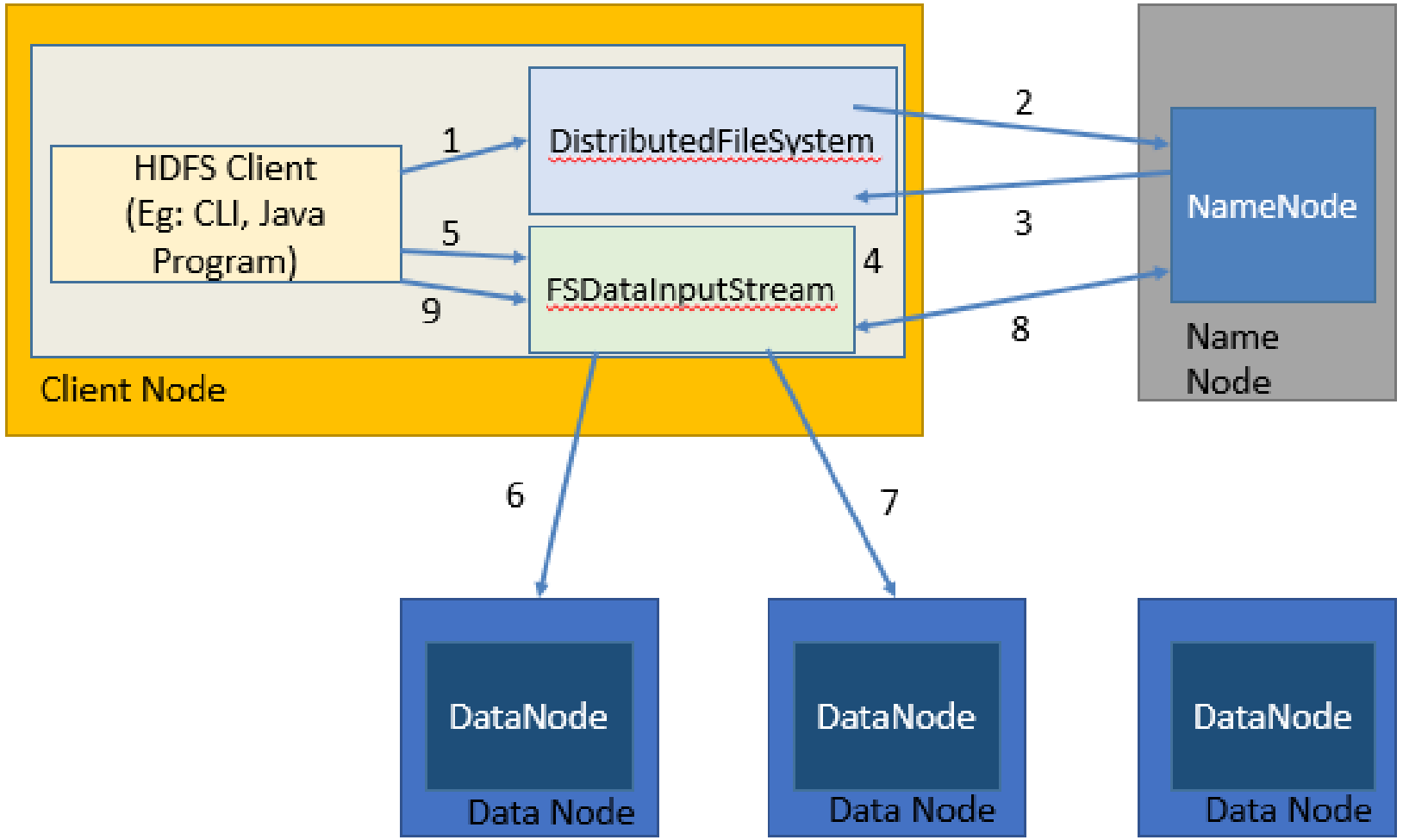
- Provides stream (channel) for reading data.

DFSInputStream

- Handles communication of the namenode with various datanodes.
- Handles integrity of data contained by the blocks.
- Manages data read activity in case of datanode failure.
- Called internally by FSDataInputStream.

Anatomy of File Read from HDFS

ACADGILD



Data Read Steps

ACADGILD

1. Client opens the file it wishes to read by calling open() on DistributedFileSystem object (which is an implementation of FileSystem class).

2. DistributedFileSystem calls the namenode, to determine the locations of the first few blocks in the file.

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

datanodes are sorted according to their proximity to the client. Proximity decreases as: different nodes on same rack> different racks > different data centers

If the client is itself a datanode, the client will read from the local datanode.

4. The DistributedFileSystem returns an FSDataInputStream to the client for it to read data.

FSDataInputStream uses DFSInputStream to manage the datanode and the namenode I/O.

Data Read Steps (Contd.)

ACADGILD

5. The client then calls read() on the stream FSDataInputStream.

6. DFSInputStream connects to the closest datanode for the first block in the file.

7. When entire block is read, the best datanode for the next block is found and read.

8. DFSInputStream calls the namenode to retrieve the datanode locations for the next batch of the blocks.

9. When the client has finished reading, it calls close() on the FSDataInputStream.

Checksum and Data Integrity

ACADGILD

- A checksum or hash sum is a small-size datum from a block of digital data for the purpose of detecting errors which may have been introduced during its transmission or storage.

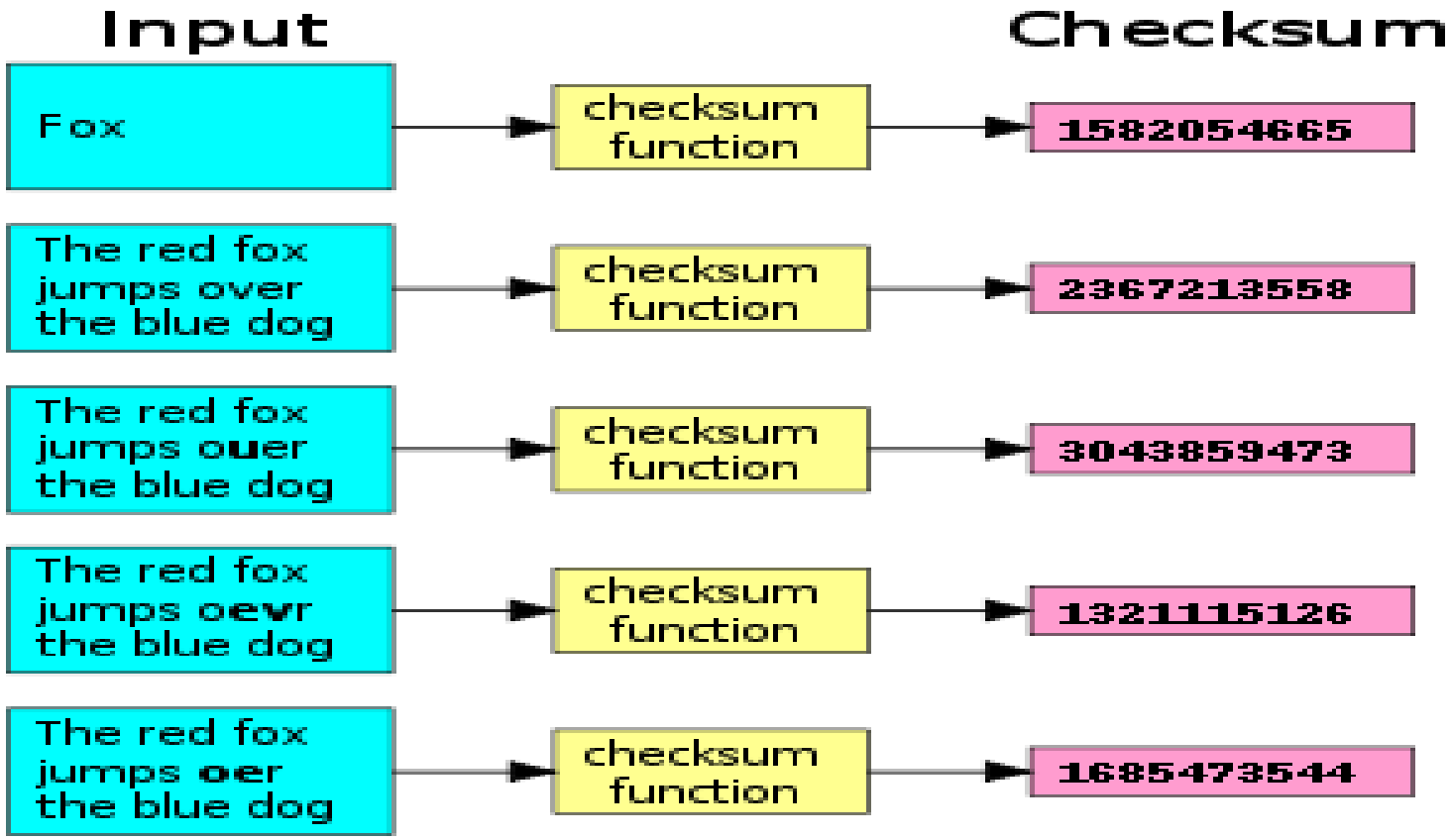
- HDFS checksums all data written to it and by default verifies checksums when reading data.

- A separate checksum is created for every dfs.bytes-per-checksum bytes of data.

- The default is 512 bytes, and because a CRC-32C checksum is 4 bytes long. The storage overhead is less than 1%.

Checksum and Data Integrity (Contd.)

ACADGILD



Checksum and Data Integrity (Contd.)

ACADGILD

- Datanodes are responsible for verifying the data they receive before storing the data and its checksum.
- When the clients read data from datanodes, they verify checksums as well, comparing them with the ones stored at the datanodes.
- -get command does the checksum verification during the data read.
- -copyFromLocal doesn’t perform checksum during data read.
- -ignoreCrc option with the -get is equivalent to -copyToLocal command.
- Disabling checksum verification is useful if we have a corrupt file that we want to inspect so that we can decide what to do with it.

Data Read from HDFS: Additional Points

ACADGILD

Scalability Friendly Architecture:

- Client contacts datanodes directly.
- namenode services block location requests only. So there is less load on namenode.

Handling Failure:

- During reading, if the DFSInputStream encounters an error while communicating with a datanode, it will try the next closest one for that block.
- It will also remember datanodes that have failed so that it doesn't needlessly retry them for later blocks.

Handling Corrupt Data:

- DFSInputStream verifies checksums for the data transferred to it from the datanode.
- If a corrupted block is found, the DFSInputStream attempts to read a replica of the block from another datanode and reports the corrupted block to the namenode.

Questions and Answers

Identify the correct statement(s).

1. -get command does checksum validation by default.
2. -copyFromLocal doesn’t perform checksum validation.
3. -ignoreCrc skips checksum validation when used with -get command.
4. Checksum manages the integrity of data and is 4-bytes in size in Hadoop.

Ans: All the statements are correct.

Important Java Classes to Write Data to HDFS

ACADGILD

FileSystem

- An abstract file system class.
- Provides methods to work with the file system like reading and writing files.

DistributedFileSystem

- An implementation of FileSystem for the distributed file system.

FSDataOutputStream

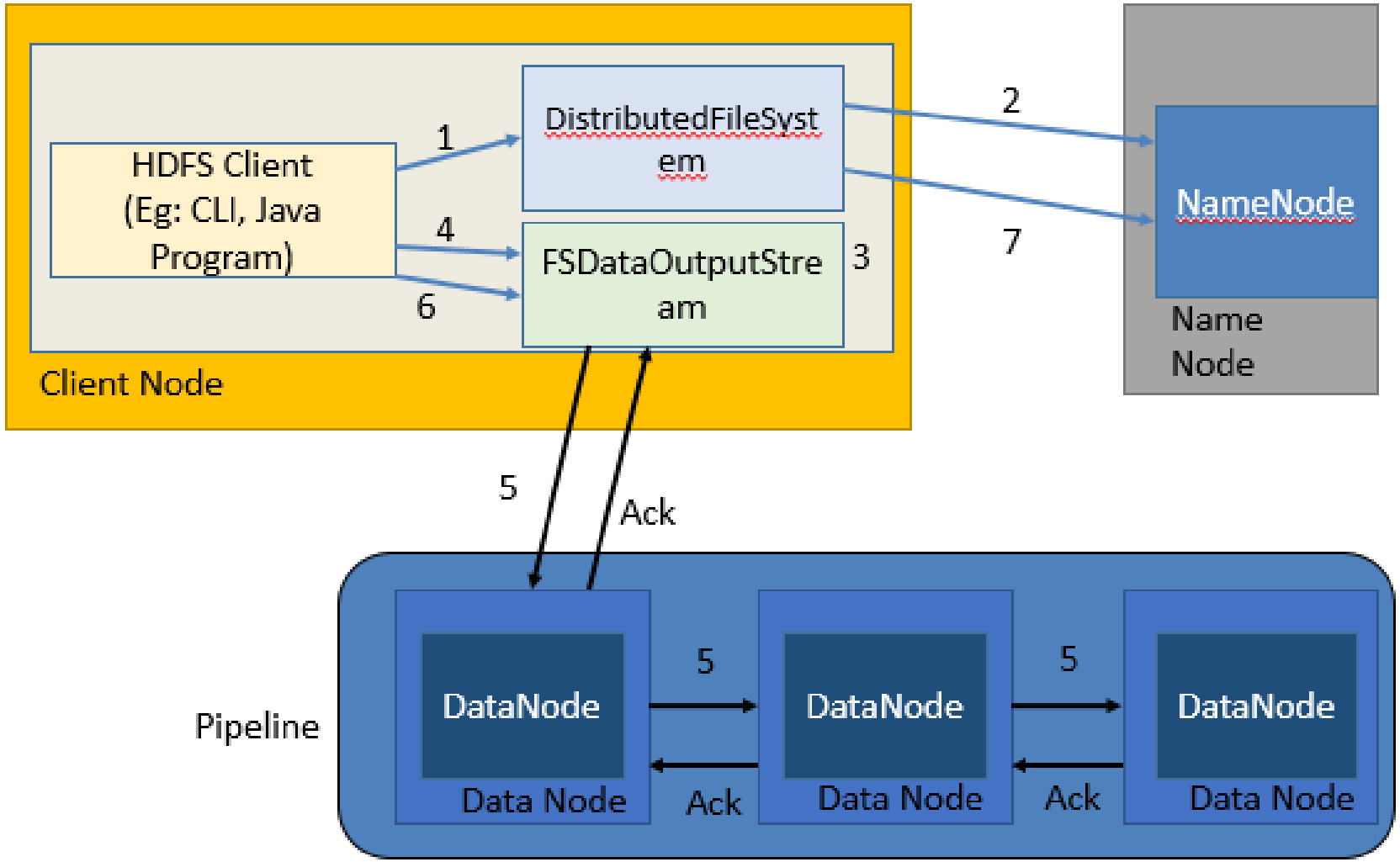
- Provides stream (channel) for writing data.

DFSOutputStream

- Handles communication of the namenode with various datanodes.
- Called internally by FSDataOutputStream.

Anatomy of File Write to HDFS

ACADGILD



Writing File to HDFS: Steps

ACADGILD

1. Client creates, calls create() on DistributedFileSystem.

2. DistributedFileSystem contacts namenode to create a new file in the filesystem's namespace, with no blocks associated with it. 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 (in edits)

3. DistributedFileSystem returns an FSDataOutputStream for the client to start writing data. FSDataOutputStream uses DFSOutputStream, which handles communication with the datanodes and namenode.

4. Client signals write() method on FSDataOutputStream.

Writing File to HDFS: Steps (Contd.)

ACADGILD

5. DFSOutputStream splits data into packets and writes it to an internal queue called the data queue.

- The data queue is consumed by the DataStreamer, which asks the namenode to give a location for the datanodes where blocks will be stored.
- The list of datanodes form a pipeline with a number of datanodes equals replication factor.
- The DataStreamer streams the packets to the first datanode in the pipeline.
- First datanode stores each packet and forwards it to the second datanode in the pipeline.
- Similarly, the second datanode stores the packet and forwards it to the third datanode in the pipeline.
- The DFSOutputStream also maintains an internal queue 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.
- So there are two queues: data queue (containing packets that are to be written) and ack queue (containing packets that are yet to be acknowledged)

Writing File to HDFS: Steps (Contd.)

ACADGILD

6. When the client has finished writing data, it calls close() on the stream. This flushes all the remaining packets to the datanode pipeline and waits for acknowledgments.

7. DistributedFileSystem contacts the namenode to signal that the file write activity is complete.

- The namenode already knows which blocks the file is made up of (because DataStreamer had asked for block allocations), so it only has to wait for blocks to be minimally replicated before returning successfully.

- Closing a file in HDFS performs an implicit hflush().

- After a successful return from hflush(), HDFS guarantees that the data written up to that point in the file has reached all the datanodes in the write pipeline and is visible to all new readers.

Handling Failures During Writing a File

ACADGILD

1. The pipeline is closed and any packets in the ack queue are added to the front of the data queue.
2. The current block on the good datanodes is given a new identity, which is communicated to the namenode
3. The failed datanode is removed from the pipeline, and a new pipeline is constructed from the two good datanodes.
4. The remainder of the block's data is written to the good datanodes in the pipeline.
5. The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node.
6. As long as dfs.namenode.replication.min replicas (which defaults to 1) are written, the write will succeed.
7. The block will be asynchronously replicated across the cluster until its target replication factor is reached (dfs.replication, which defaults to 3).

Questions and Answers

ACADGILD

- Which class handles communication of the namenode with various datanodes?

1. FileSystem
2. DistributedFileSystem
3. FSDataOutputStream
4. DFSOutputStream

Ans: 4. DFSOutputStream

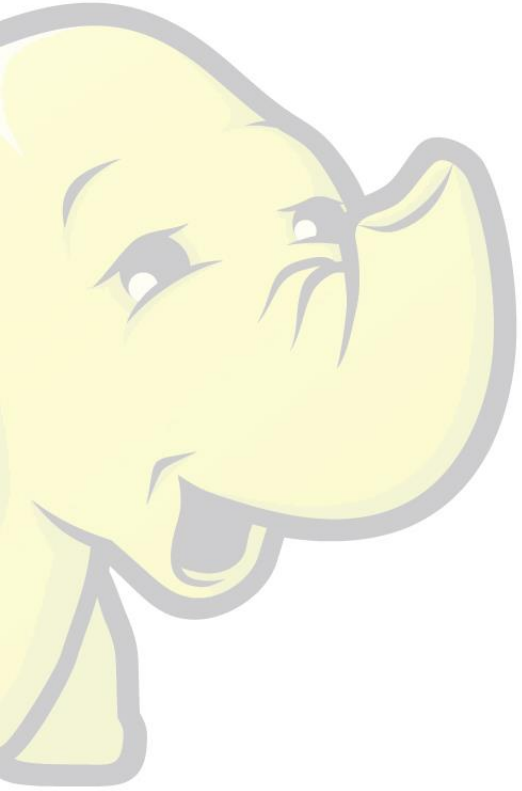
Questions and Answers (Contd.)

ACADGILD

- During writing a file in HDFS with replication factor of 3, what will happen if replication of a block to a datanode fails?

1. Write activity fails
2. Failed block will be stored at namenode
3. A new pipeline is constructed with good datanodes and block is replicated to those data nodes and earlier block is deleted.
4. Hadoop will ask user, whether to continue with failure or ignore the under-replicated warning.

Ans: 3. A new pipeline is constructed with good datanodes and block is replicated to those data nodes and earlier block is deleted.



THANK YOU

Email us at- support@acadgild.com