GROUP 20

Design Document

YAHAS:: A HADOOP DISTRIBUTED FILESYSTEM

Table of Contents

LIST OF FIGURES

1. Introduction

1.1 Purpose

This document presents a detailed description of the design of a Hadoop like distributed file system called *YAHAS (Yet another HAdoop System)*. It describes the detailed architecture, module level decomposition and other relevant information required for implementation. This document is targeted for System Architects, System Designer and System Developers who would be working in this project henceforth.

1.2 Project Scope

The envisaged system is a distributed file system similar to Hadoop. It is henceforth referred to as YAHAS (**Y**et **A**nother **HA**doop **S**ystem). A distributed file system is file system which allows the storage and retrieval of files distributed across multiple machines. These machines are assumed to be at a single Data Centre and not distributed geographically through WAN or Internet. The system will be designed and developed to cater to the requirements of managing large volumes of data (*often referred to as Big Data*). The motivation of developing such a tool is that often a Relational Database System (*RDBMS*) would not scale beyond a point to cater to the requirement of large volumes of data effectively. Therefore a paradigm shift is required in the way we design data storage and retrieval mechanism for Enterprises handling large volumes of data.

YAHAS will be a distributed file system focusing on the following:

- *Fault Tolerant*: The file system should be designed and developed in such a way that it should not have a single point of failure. Suitable redundancy mechanism should be incorporated so that failure of a single system does not hamper the operation of the system. For example, failure of a single node should not render a file stored in the system as unusable.
- *Integrity*: Integrity of the data stored in file system is of utmost important. Therefore, system should implement mechanisms like checksum etc. to ensure integrity of data being stored.

- *Scalable*: Since YAHAS is being designed keeping in mind Big Data scenario, it should be designed in such a way that it should scale to handle large amount of data.

1.3 Definitions, Acronyms, and Abbreviations

DataNode	A Node which stores Data
NameNode	A Node which stores Metadata Information
Client	A client application software
ACK	Acknowledgement
NACK	Negative Acknowledgement
Heart Beat	Information send from Data Node to Name Node to let it know that a Data Node is active
Lease	Lease is granted when a file is being created to avoid issues when a Data Node dies before writing a block
RMI	Remote Method Invocation
UUID	Universal Unique Identifier
Block Report	Block Report is a report prepared by DataNode. It consists of a list of blocks that is currently available on a DataNode.

1.4 Overview

This section presents the overall organization of the Design document. Section 1 primarily presents the scope of the project YAHAS, along with a list of acronyms, definitions which have been used throughout the document. Section 2 presents the architecture and functionality of YAHAS. This section aims to provide a High Level Overview of the overall system and the major subsystem/module. Section 3 presents in details of the various sub systems of YAHAS and the interactions between them. Section 4 presents the class diagrams of the various components of YAHAS.

2. System Architecture

Figure 1, presents the major components along with their interfaces and interaction. YAHAS is composed of three main subsystems namely: Client(s), a NameNode, and a set of DataNodes.

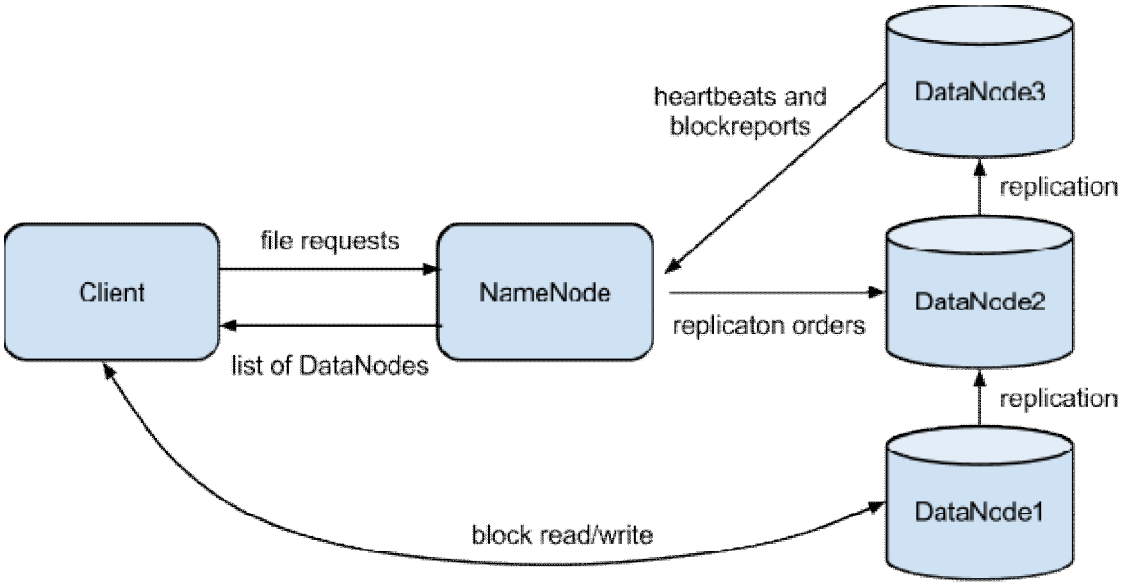


Figure 1: High Level Architecture

- The client is a computer through which a user can interact with the file system.
- The NameNode is a master server which manages client's access to the files and keeps the metadata information about files.
- The DataNode is a system on which actual storage of blocks takes place.

All the three entities namely, the client, NameNode and the DataNode are software components that run of different machines.

Client

A client is a computer through which a user can interact with the distributed file system. YAHAS provides an user interface at the client machine through which the client can sends file system commands like create, read, update, move and delete. For create and read commands the NameNode replies with a list of DataNode from which to request data or send data to. The client can then directly interact with these data nodes without going through the NameNode.

NameNode

The NameNode is a server which manages clients' access to the files and keeps the metadata information about files. The NameNode responds to commands sent by clients and can also send commands to the DataNodes such as: replicate a DataBlock to another DataNode, delete a Datablock and request a BlockReport which contains a list of blocks stored in a DataNode.

DataNode

The DataNode is a system on which actual storage of blocks takes place. The DataNode usually deployed as one instance per computer. The DataNode continuously loop, sending heartbeats to the NameNode. It interacts with other Data Nodes for replicating blocks. It further, send and receive data asynchronously as requested by the Client or other DataNodes. The DataNodes have no concept of directories, only blocks, all of the moving, renaming and listing of files to make it comprehensible for the user is taken care of single handedly by the NameNode.

Since YAHAS is being designed and developed using JAVA, each component communicates to another using JAVA RMI. Java RMI is a mechanism that allows one to invoke a method on an object that exists in another address space. The other address space could be on the same machine or a different one. The RMI mechanism is basically an object-oriented RPC mechanism. The use of RMI differentiates our approach to the distributed file system design as opposed to HADOOP which uses RPC.

While designing a Java RMI application, there are three processes that participate

- *Client* : This is the process that is invoking a method on a remote object.
- *Server:* This is the process that owns the remote object. The remote object is an ordinary object in the address space of the server process.

- *Object Registry* is a name server that relates objects with names. Objects are *registered* with the Object Registry. Once an object has been registered, one can use the Object Registry to obtain access to a remote object using the name of the object.

In our implementation, the DataNode and the NameNode will have RMI Interfaces which can be called by the Client Node. The communication between each of the components is through protocols as shown in Figure 2.

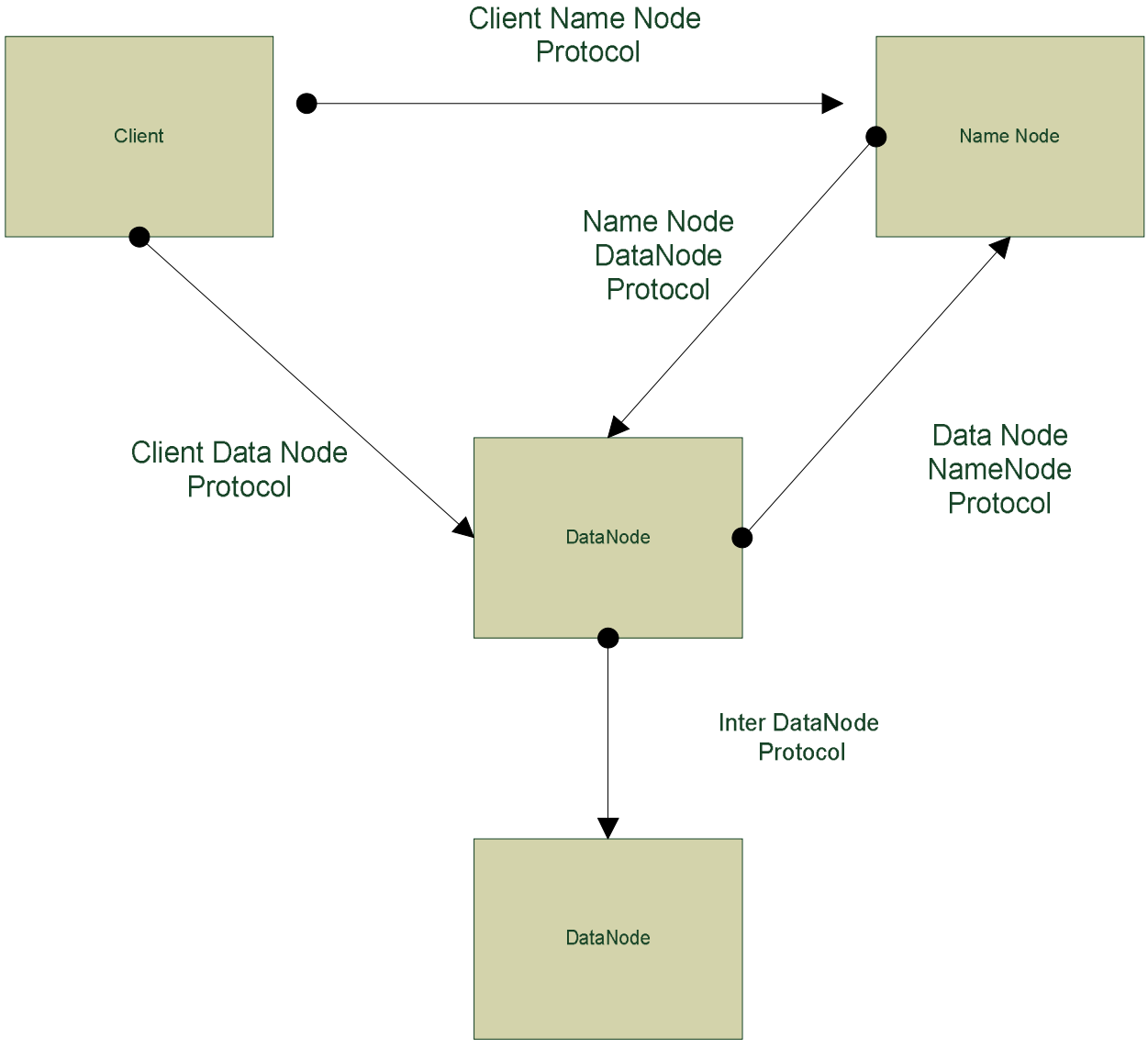


Figure 2: Communication between Major Modules in YAHAS

The Client Node interacts with NameNode using the Client Name Node Protocol. Through this protocol a client can initiate the process of creation of a file, read a file etc. The Client also interacts with the Data Node for actual writing of blocks. This is done through the Client Data Node Protocol. The DataNode interacts with the Name Node using the DataNode NameNode Protocol. This protocol specifies mechanisms to send block reports etc. The NameNode can also interact with DataNodes to initiate replication orders through the NameNode DataNode protocol. DataNode communicate with DataNodes using inter data Node protocol. This protocol is essentially used for implementing a DataNode Replication pipeline.

2.1 Module Decomposition

Figure 1 shows the major modules of YAHAS. This section mentions in details the design of these modules along with the services provided by these modules.

2.1.1 Name Node

The NameNode can be considered as the controller of the YAHAS cluster. It is responsible for managing all the operations that are requested by the client. It performs activities like allocation of Blocks to File, Monitoring Data Node for Data Node Failure and new Data Node addition, Replication Management Client requests management – like writing file, reading file etc.

Figure 3, presents the subcomponents of the Name Node. The Name Node server is implemented as an RMI Server with protocol handlers of DataNode-NameNode Protocol, Client Node-NameNode Protocol and NameNode-DataNode Protocol. These protocols are discussed in details in the subsequent sections. HeartBeat Monitor is a component of the NameNode which monitors Heart Beat from the data not and maintains a list of data nodes that are available. Components in YAHAS can query the NameNode to find out the list of DataNode that exist in the cluster. The Replication manager is a component which manages replication of blocks across DataNodes. When a NameNode receives a Block Report from the Data Node, it can looks for block which are under replicated and issue replication command to ensure the replication factor. The Pruner manager is responsible for removing files in the namespace.

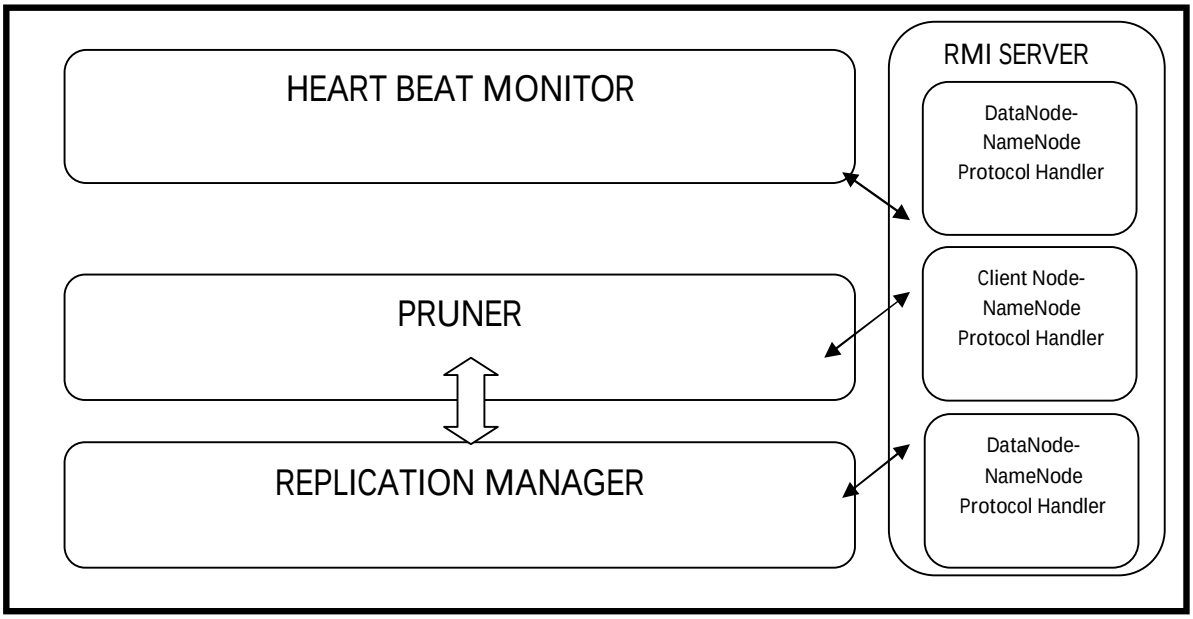


Figure 3: Name Node Subcomponents

2.1.1.1 Name Node Startup:

Figure 4, presents the startup mechanism of NameNode. The Name Node enters the safe mode during the startup. During the safe mode client requests are not entertained. Firstly it loads Configurations parameters required for its startup. These configurations are in the form of files which are read during the startup. Thereafter, it initializes MBeans for monitoring parameters. The MBean is used for collecting performance statistics, resource usage and setting application parameters of the NameNode. Once the MBean is initialized then, it forks three threads which RMI server, the Heart Beat Monitor and the replication Manager. Other components in the YAHAS subsystem can communicate with the RMI Server to invoke services provided the NameNode using a suitable protocol. The Heart Beat Monitor is a UDP server which listens to the heart beat from the DataNodes and also manages the list of data node that are currently available. The Replication manager receives block reports from the Data Nodes and the issues replication commands if a block is below the replication factor. Once these services are started without any errors, the NameNode exits out of Safe mode and will now be ready to receive request from other components of YAHAS subsystem.

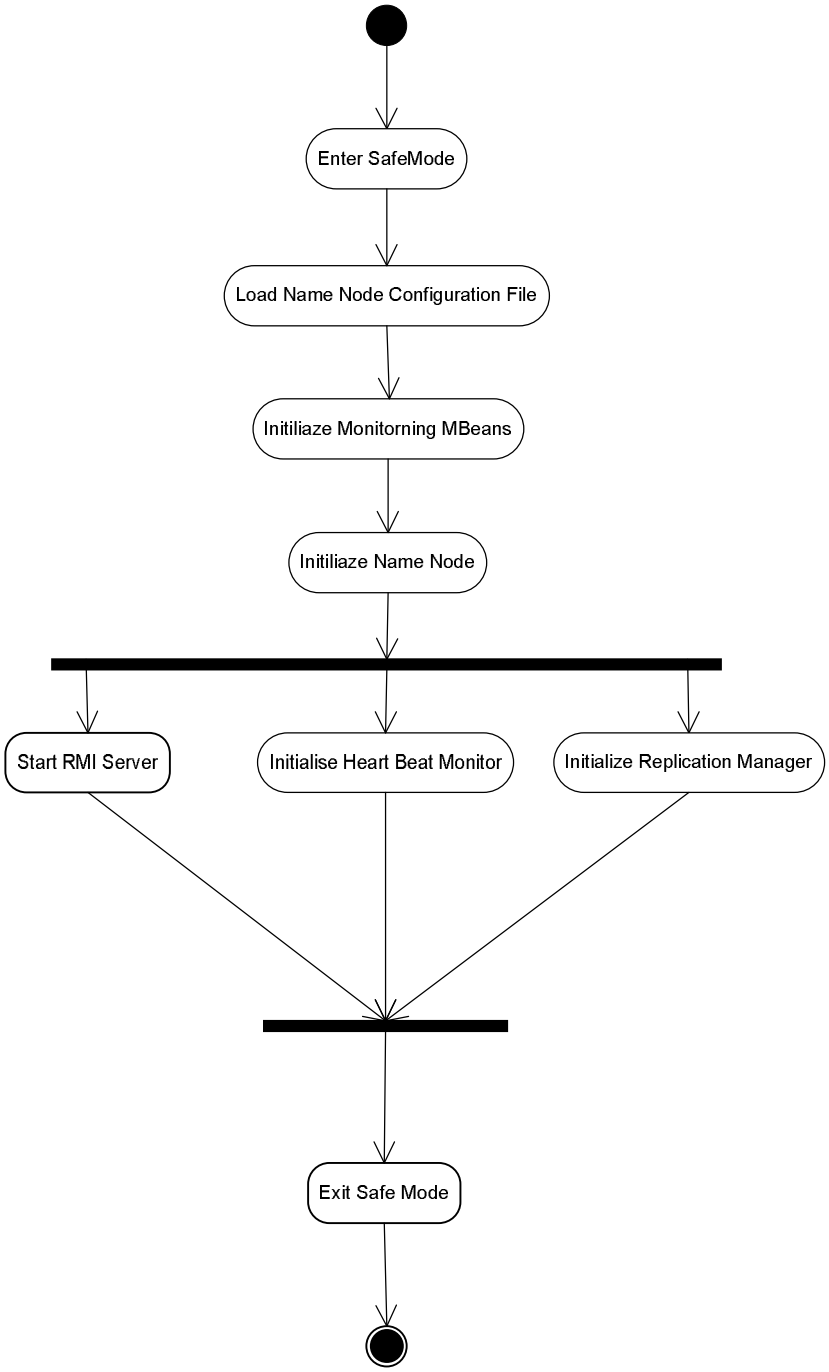


Figure 4:NameNodeStartup

2.1.1.2 Name Node operations

This section specifies the various operations that are supported by the Name Node.

Client to Name Node Communication

A client communicates with the Name Node using the Client Name Node protocol. The Name Node exposes the following functionalities that can be utilized by the client.

1. Creating a file: A client can request a NameNode to create a file in the Namespace of the cluster. A NameNode creates a file in the namespace and adds a lease to the client. A client renews the lease to continue operations on a file. The lease mechanism is added to avoid issues pertaining to when a client dies unexpectedly without fully writing a block. A client can then add blocks to a file. The sequence diagram for creating a file in Figure 5.

2. Reading a file. A client can request a NameNode for a file. A NameNode locates the namespace entry of the file and returns a list of DataNodes along with the Blocks that is with each of the DataNode. The Client can request for each block directly from the Data Node and while doing so fetch the file in parallel.

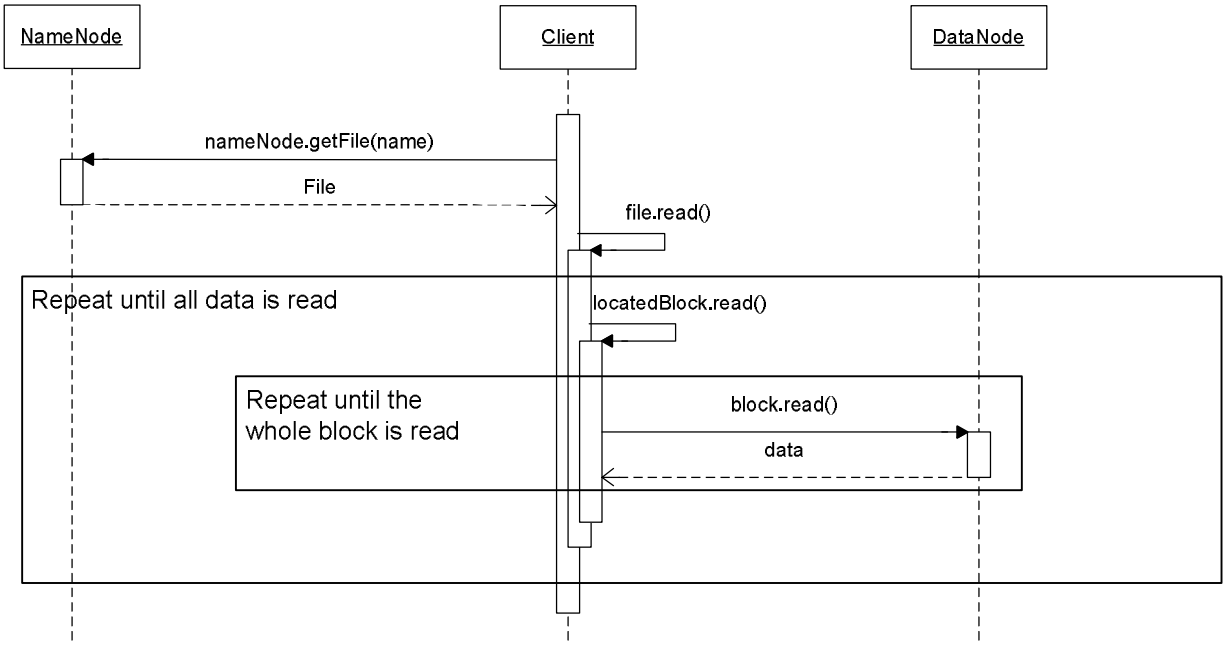


Figure 5: Sequence Diagram to Read a File

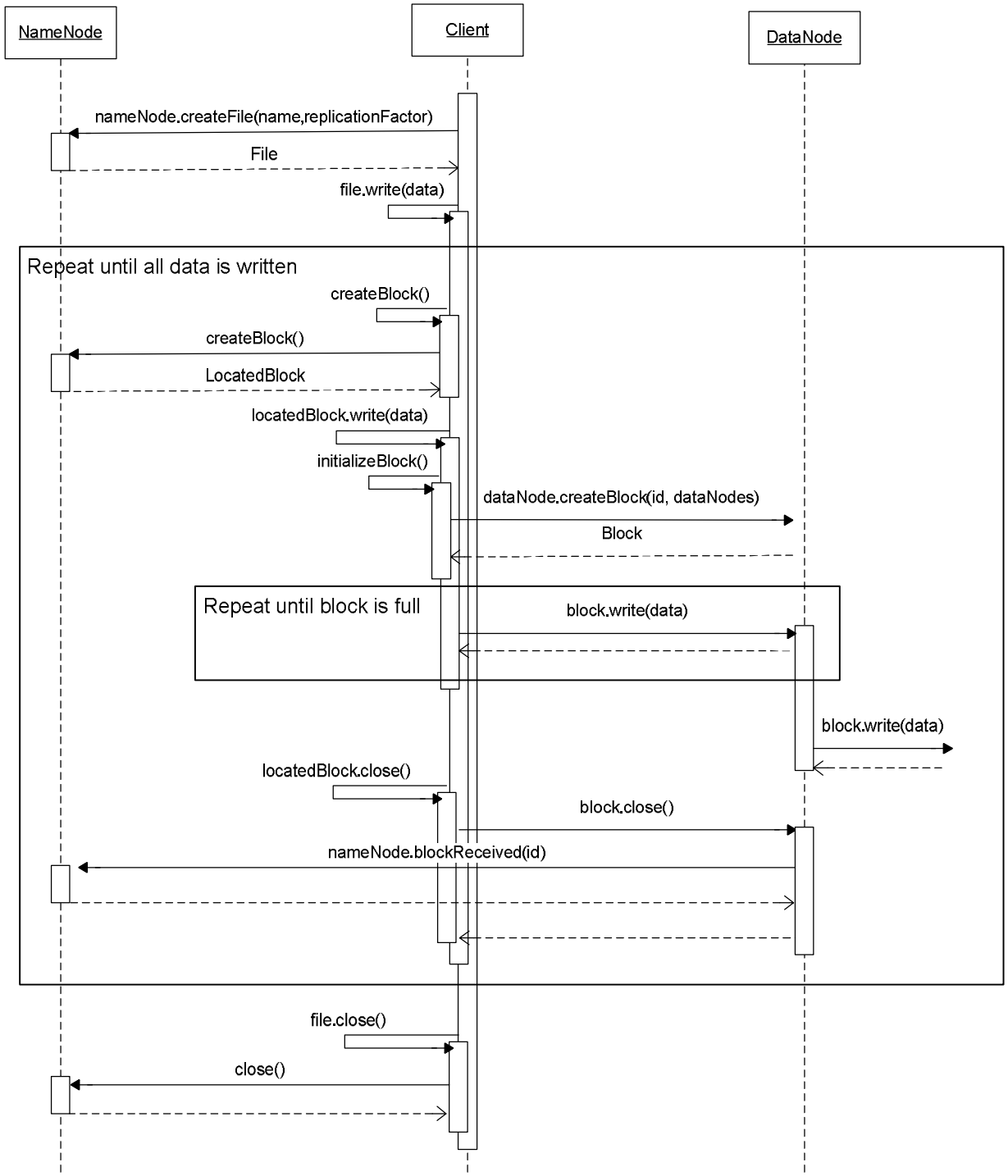


Figure 6: Sequence Diagram for Creating a File.

3. 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.

   The sequence diagram for Locating Block process is given below:

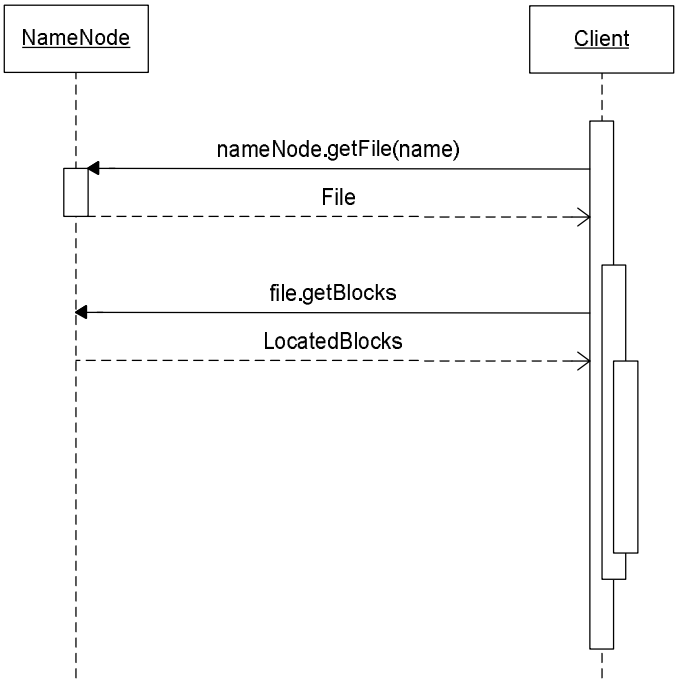


Figure 7:Sequence Diagram For Locate Blocks

4. Creation of a Block: After a file is created, the client needs a block to write data to. It then request Name node to provide a block to write to. The Name node uses Data node statistics and returns data nodes to the client with block information. This is similar to locating blocks however, this time new block are created for writing purpose. The client can then communicate directly with the DataNodes to create the block.

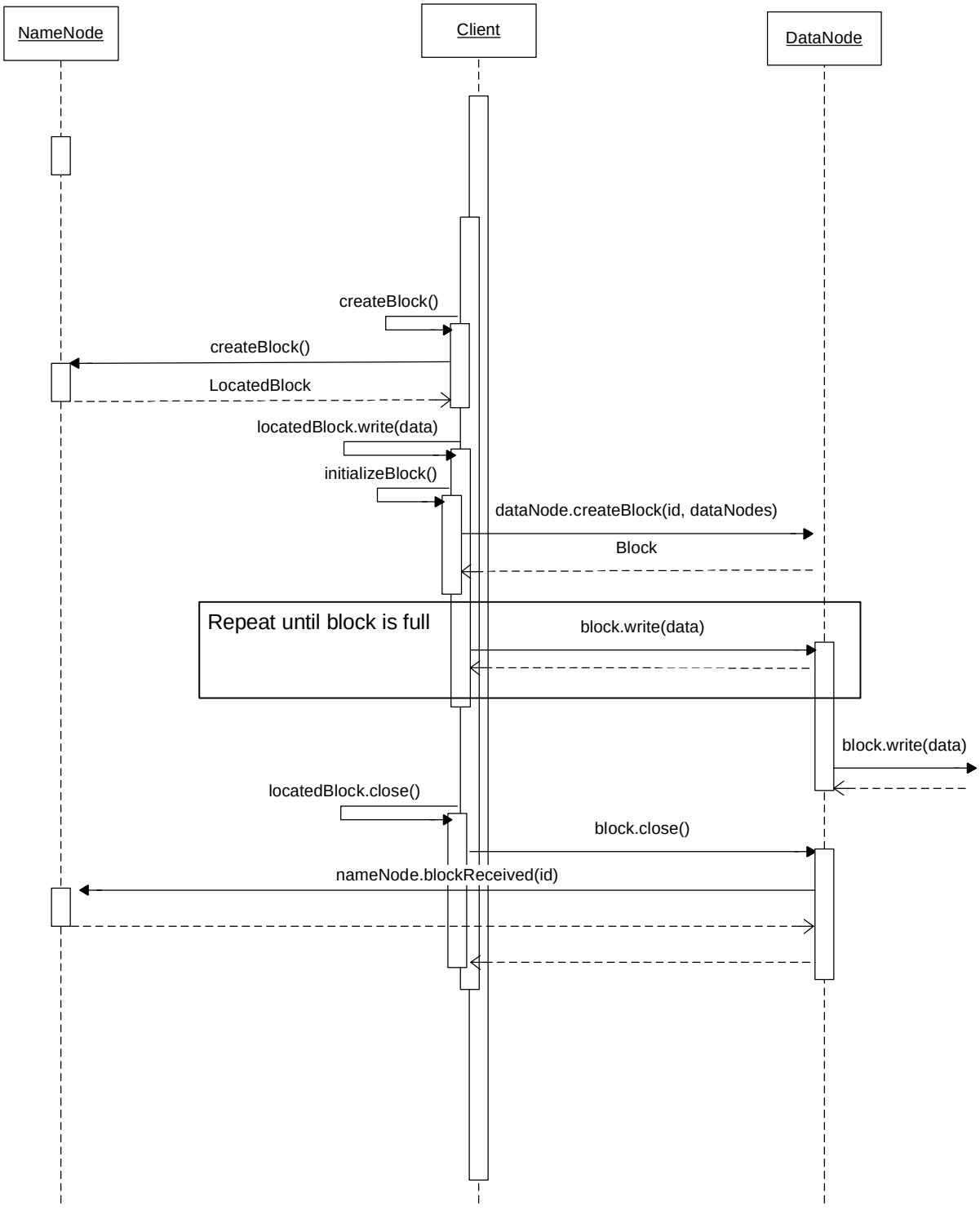


Figure 8: Sequence Diagram for creating a block

5. List Data Nodes: Client can query all the data nodes available in current YAHAS system.

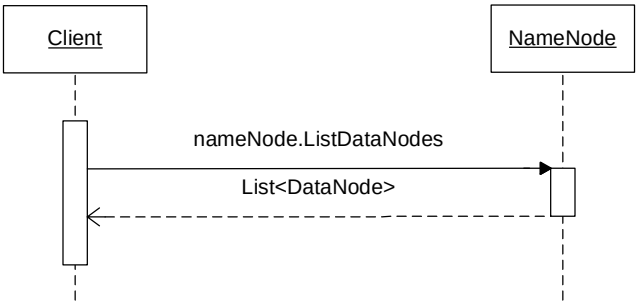


Figure 9: List Data Nodes

6. Close a File: After all blocks are written, the client can issue a request to close the file to the NameNode. The NameNode then perform operation like releasing the lease, issuing replication command etc.

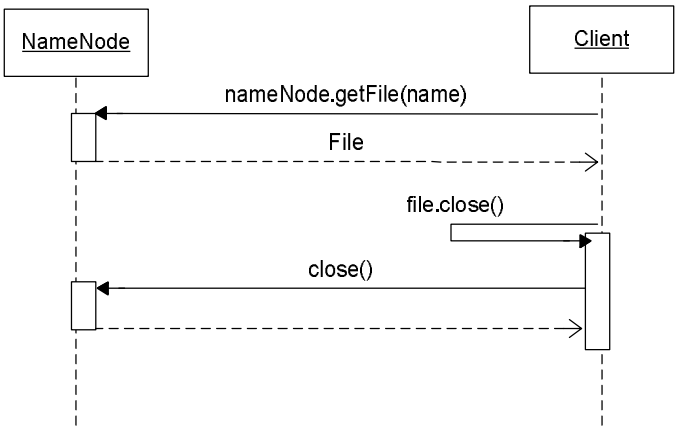


Figure 10: Sequence Diagram to Close a file

7. Query NameNode: A client can query Name Node to find the size of block of a file, whether the NameNode is in safe mode, statistics of the cluster etc.

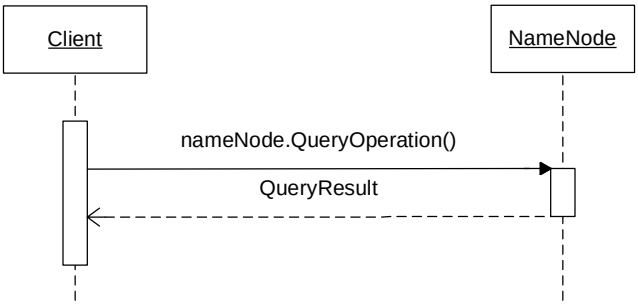


Figure 11: Sequence Diagram to Query Name Node

8. Meta data Operations: A client can request Name Node for metadata operations like:

- Creating a Directory
- Moving a File to New Location/Renaming a file
- Changing permission of a file
- Changing ownership of a file
- Deleting a File:

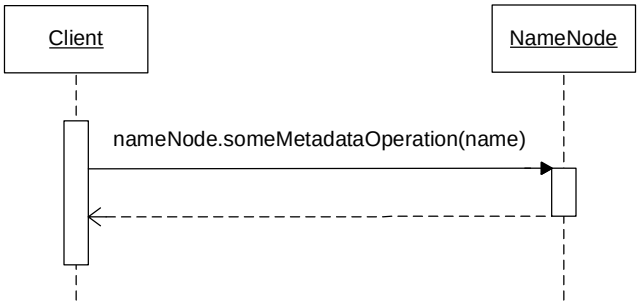


Figure 12: Sequence Diagram Metadata Operation

Data Node to Name Node Communication

A Data Node communicates with the Name Node using the Data Node Name Node Protocol. Following are the operations that are part of this protocol:

1. Registration – When a data node want to be part of YAHAS system, it need to register itself with Name node. Name node registers data node and allocates id to it. While registering the Data Node sends information like hostname, IP and MAC address and it's UUID.
   The sequence diagram for registration is shown below:

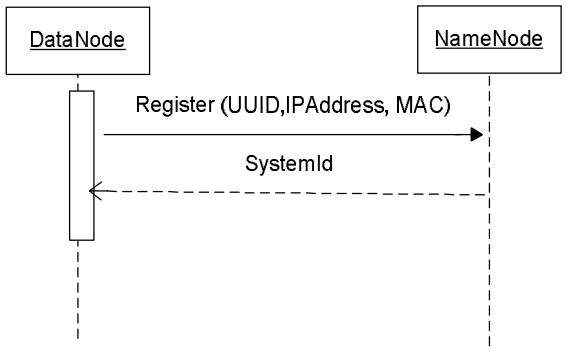


Figure 13: Data Node Registration

2. Block Report- A Data Node sends a list of Block that it has. Essentially, the Data Node only sends the list of non corrupted blocks that it has. Based on the receipt of Block report a Name Node can check the replication factor of a file and issue a replication command. The replication command is issued as per the NameNode DataNode Protocol.

   The sequence diagram for sending block report is provided below:

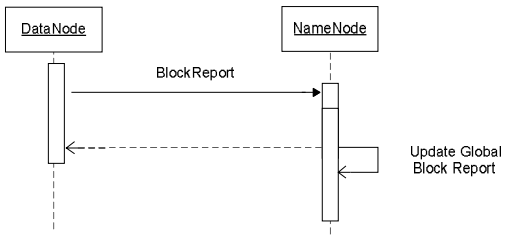


Figure 14: Sequence Diagram Block Report

3. Heart Beat- A DataNode sends heart beat to Name Node, as per a predefined interval. The receipt of the heart beat enables the Name Node to maintain a list of active Data Nodes at instance of time.

   The sequence diagram for sending heart beat is provided below:

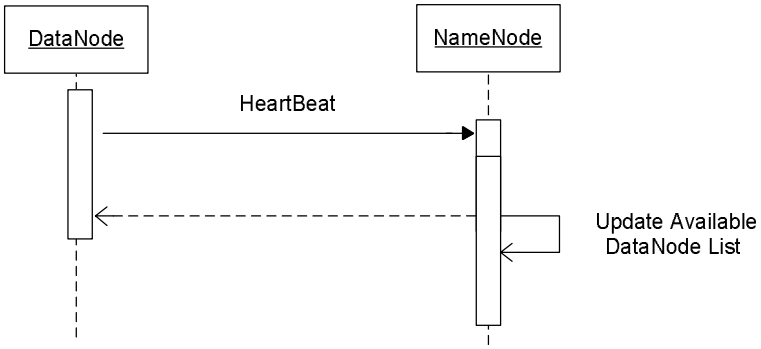


Figure 15: Sequence Diagram for Heartbeat

4. Block Received: Whenever a block written to a DataNode is completed, it notifies the NameNode that a block has been received. Further, the DataNode also notifies the NameNode when blocks are replicated from one data node to another.

   The sequence diagram for sending block report is shown below:

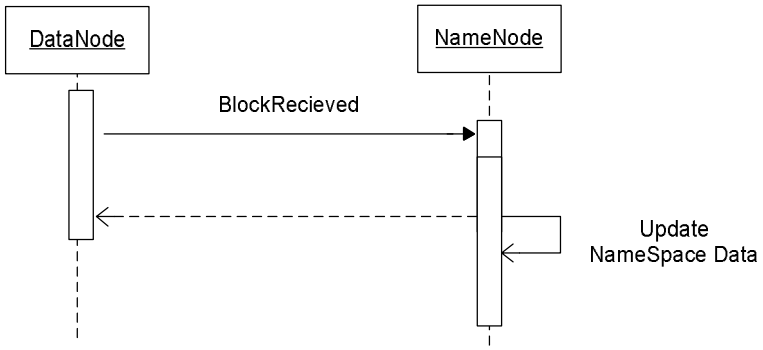


Figure 16:Sequence Diagram of Block Received

2.1.2 Data Node

Data Nodes stores the actual data in the YAHAS system. Data Node implements Inter Data Node Protocol to offer services to other Data Node. It also implements a ClientDataNode Protocol to expose it's services to Client Nodes. Data Nodes are also responsible for maintaining the integrity of the data. Therefore, the DataNodes maintains the checksum of blocks that it has. The DataNode does not have a file level view of the data stored in it. The file level view is only with the NameNode. The DataNode communicates to the NameNode to send Block Report, Heartbeat and notifies the Name Node about the block level commits that it does.

The major components of the DataNode are shown in Figure below. The DataNode RMI Server implement the InterDataNode and ClientDataNode Protocol. The Integrity Manager component is responsible for calculation CRC and/or hash of the block and storing it. The Heartbeat generator generates heart beat and sends it to the NameNode. The BlockReport generator, generates the BlockReports which are then send to the NameNode. The Replication Manager manager inter data node replication. In case a DataNode is not available during replication, the replication manager skips that particular data node and moves onto the next. The registration manager handles registration of a DataNode with the NameNode. It gathers information like IP address, MAC address and generates UUID for a particular DataNode. Thereafter, it sends this information to the NameNode for registering itself.

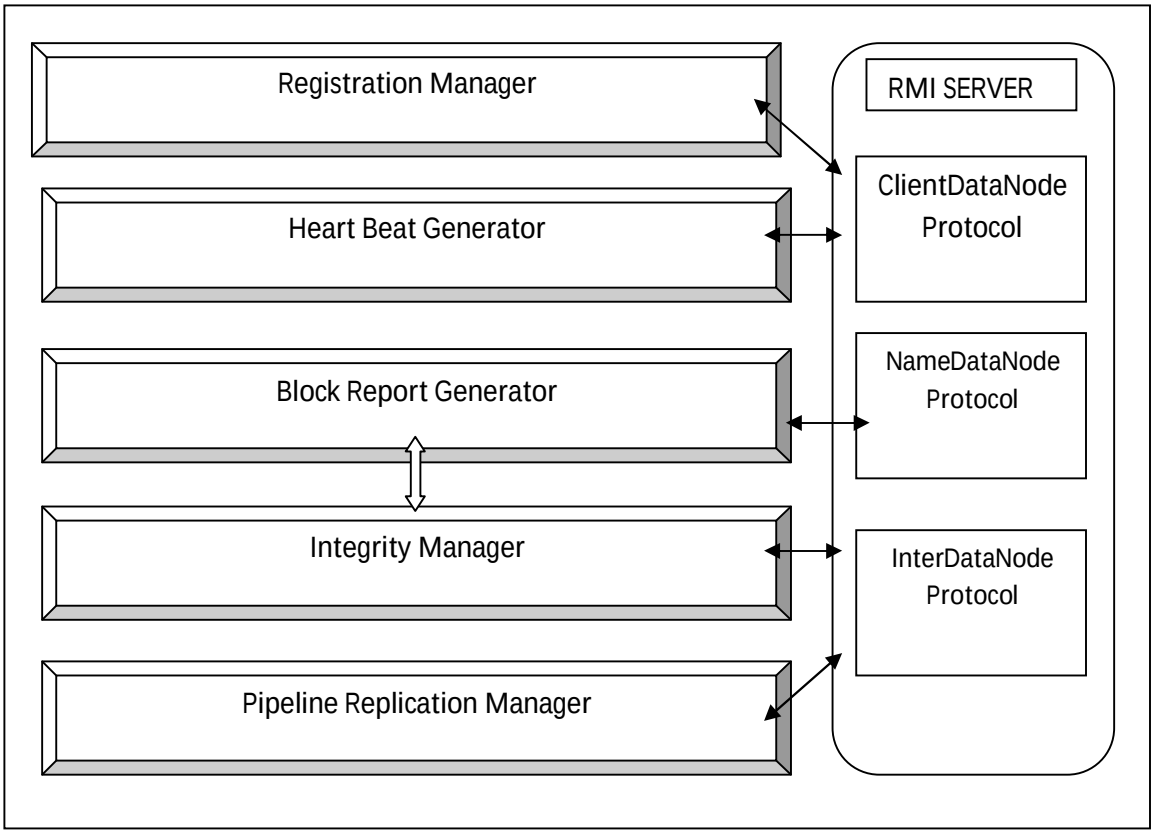


Figure 17: Sub Components of Data Node

2.1.2.1 Data Node Startup

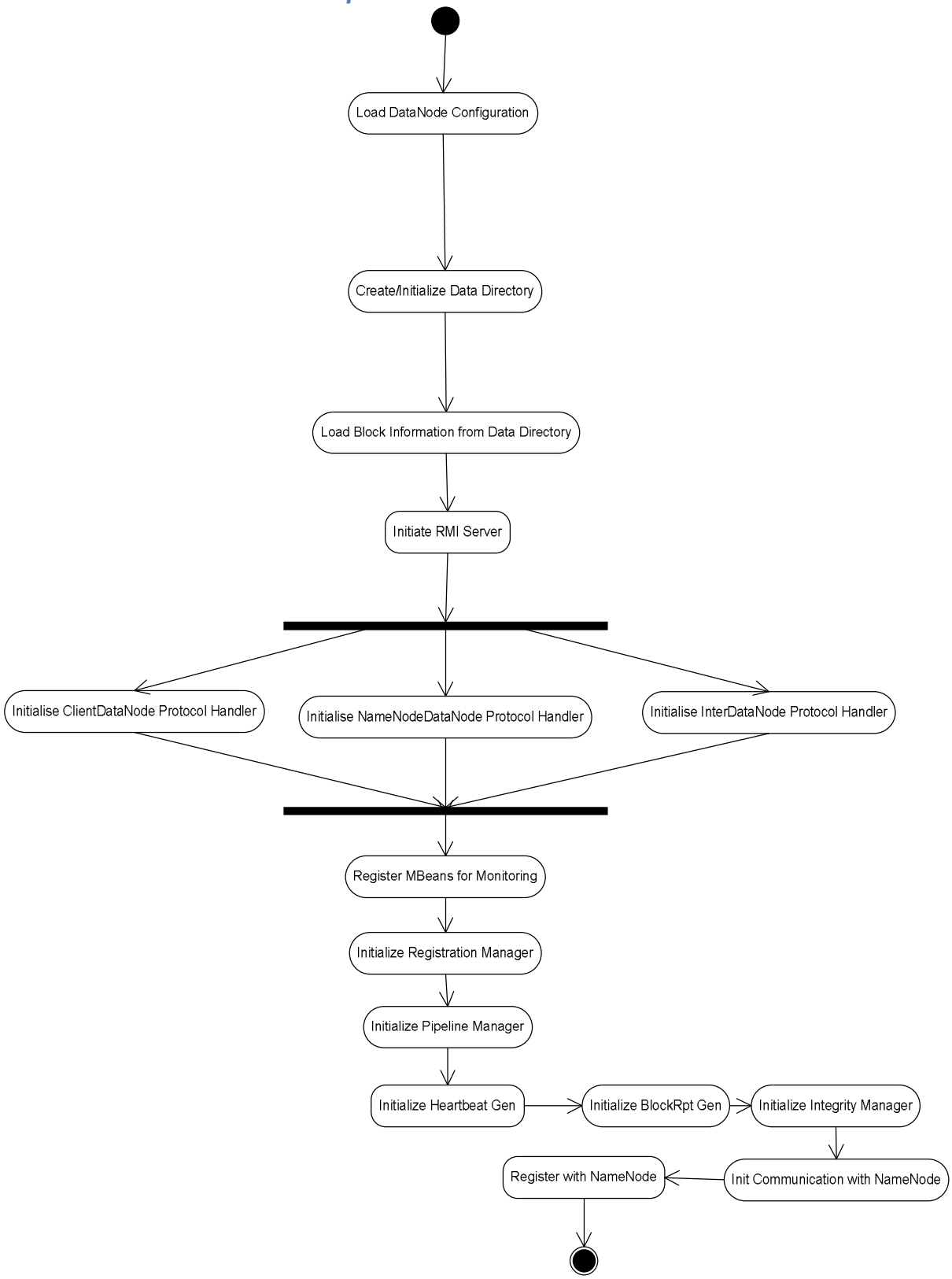


Figure 18: Activity Diagram DataNode Startup

Figure 6, presents the Activity Diagram pertaining to the starting up of the DataNode. The DataNode initially reads a configuration file for the initial configuration parameters like the hostname, IP address of the NameNode, path for the data directory etc. It then goes on to initialize the Data Directory (i.e, the directory in which it stores the block). Thereafter, it initialized MBean for gathering stats of its performance and usage. It then loads Block Information of the Block directory. Further, it initializes the RMI server, which in turn initializes the protocol handlers for Client Communication, Name Node Communication and communication with other DataNode. Once the RMI server is successfully initialized other components of the system like Registration Manger, Integrity Manager, Heartbeat Generator, Blockreport Generator. Once the subsystems are initialized the DataNode registers itself with the NameNode.

Data Node Operations:

Following are the operations supported by the Data Block:

- Read Block: The Read Block command is used to read block information which is with the dataNode. The DataNode maintains a directory wherein it stores the blocks. The Datanode therefore finds the Blocks from this directory.
- Write Block: This command is used to initiate a write request to the Data Node. This request can either be from the client or from another DataNode.
- Read Metadata.: This commands returns a metadata of the block
- Checksum Block: This command is used to create a checksum for the block.
- Heart Beat: The Data Node is responsible for sending heart beat information to the Name Node. The Heart Beat Is send in predefined interval of time. If a NameNode does not receive it assumes that a the Data Node is dead and updates the list of available DataNode.Within the name node, the Heart Beat generator is responsible for generating Heart Beat.

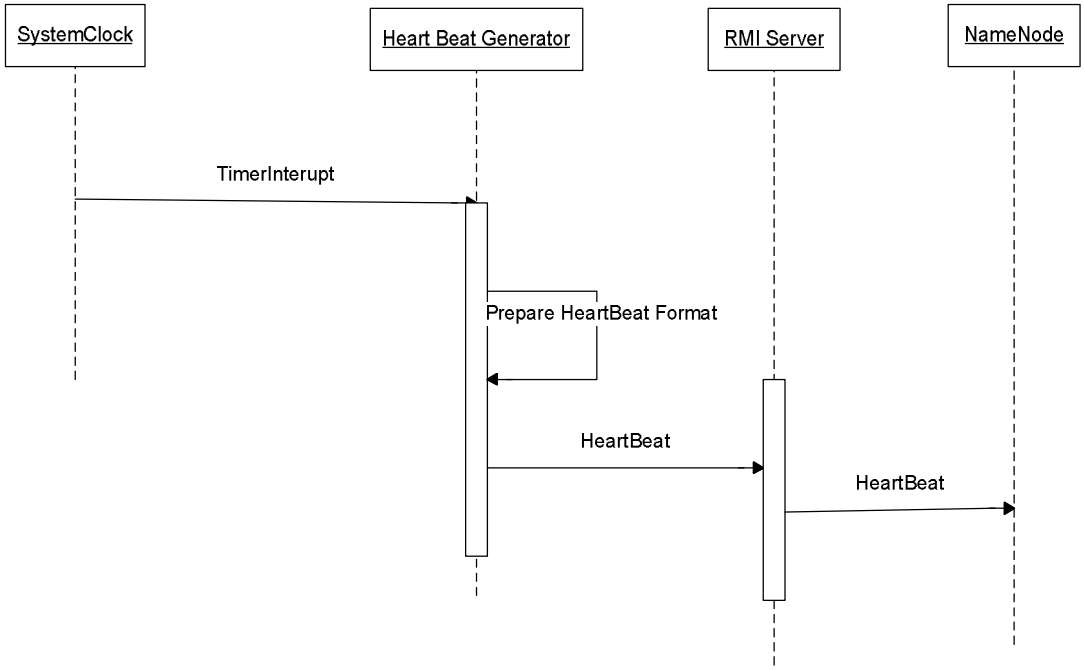


Figure 19: Sequence Diagram for Heart Beat Generation

- Block Report: The DataNode is also responsible for creating Block report and sending the same to the NameNode. The NameNode a list of available blocks on the DataNodes and issues replication commands is a file is not replicated enough.

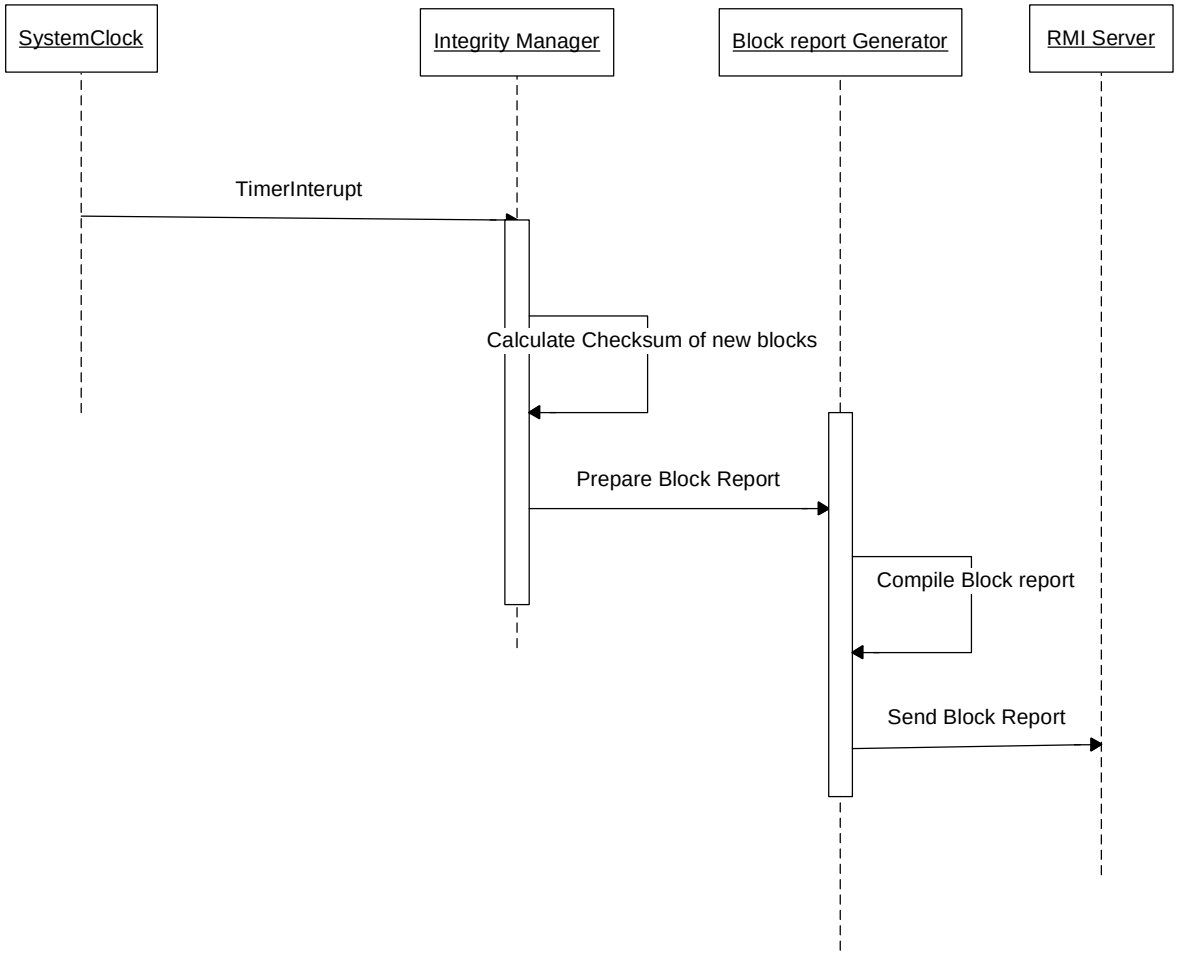


Figure 20: Sequence Diagram to Send Block report

2.1.3 Client Node

The Client Node provides a command line interface to YAHAS. The list of commands provided by the client and operation is provided in the table below:

Table 1: List of Commands

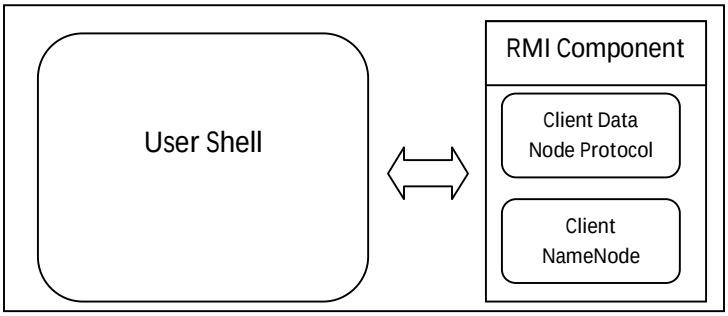
create	Command to create a new file.
	1. The user of the system, uses the YAHAS Client program to the following: • The files and/or directories he wants to upload. • Which remote directory to upload them to. • Replication rate 2. The client sends the above information to the NameNode: 3. The NameNode sends the client a list of DataNode to which the client can write the files. 4. The client sends data pertaining to one block size along with the DataNode list to the first Data Node. 5. The first Data Node receives the data from the Client and further, streams the data to the Next DataNode in the list. This process continues till all the DataNodes in the list are covered. 6. The DataNode then informs the NameNode of the Data Block that they have written. 7. A step 3-5 continues till no more blocks of data is left for the client. 8. The Client closes the file. 9. NameNode updates the file's Metadata information.
cat	To list the content of the file.
	1. The user specifies in the client program: • The files and/or directories he wants to download. • Which local directory to download them to. 2. The client program forwards this information to the Name Node.

	3. The NameNode returns a list of DataNodes having the block pertaining to the file. 4. The client can then directly read blocks from the above DataNodes. 5. The client then assembles the file after it receives all the blocks. The client notifies the user after the download is complete.
mv	Move/rename files and/or directories
	1. The user specifies: • The source files and/or directories, to move. • The destination directory to move it too. 2. The client forwards this information to the NameNode. 3. The NameNode will update the Metadata information of the File and return to the client. 4. The client presents the updated information to the user.
rm	Delete files and/or directories
	1. The user specifies the files/directories to delete. 2. The client forwards this information to NameNode. 3. The NameNode marks these files as deleted, however the actual blocks are not deleted. 4. The client is notified about the changes made in the file metadata. 5. The Pruner component of the NameNode then, issues command to the Data Node to physically deletes these directories.
ls	List the contents of a directory
	1. The user specifies the directory to be listed to the client. 2. The client forwards this to the NameNode. 3. The NameNode returns the metadata information of the Directory to the client The client displays the information to the User

chrep	Change replication rate of files and/or directories
	• The user specifies The files and/or directories to change replication rate for • The client forwards this information about this to the NameNode. • The NameNode updates it's metadata information and incorporates the changed replication factor. The NameNode issues replication orders to DataNode when it receives the next heart beat from it.
chmod	Change the permissions of files and/or directories
	1. The user specifies to the client program the following: • The remote files he wants to update the permissions for • The new permission mask. 2. The client forwards this requests NameNode. 3. The NameNode updates the Metadata Information and updates the client The client informs the user.
chown	Change ownership of the file
	1. The user specifies to the client program the following: • The remote files he wants to update the permissions for • The new permission mask. 2. The client forwards this requests NameNode. 3. The NameNode updates the Metadata Information and updates the client.
Chgrp	Change the group of a file.
	1. The user specifies to the client program the following:

	• The remote files he wants to update the permissions for • The new permission mask. 2. The client forwards this requests NameNode. The NameNode updates the Metadata Information and updates the client
Useradd	Create a User
	1. The user specifies to the client program the following: • The username • The group to which the user belongs 2. The client forwards this requests NameNode. The NameNode updates the Metadata Information and updates the client
Rmuser	Removes a user.
	1. The user specifies to the client program the following: • The username 2. The client forwards this requests NameNode. The NameNode updates the Metadata Information and updates the client

The subcomponents of the Client Module are shown below. The user interacts with the User Shell to type command for various operations as listed above. The Shell interacts with the RMI component to interact with DataNodes and NameNode using their respective protocol.



UML Class Diagrams

This section presents the Object Oriented design of the YAHAS subsystems. YAHAS subsystem is divided into the following packages:

- **Client**: The package contains code that contains the client component of YAHAS . The class diagram of the classes in this package is shown below:

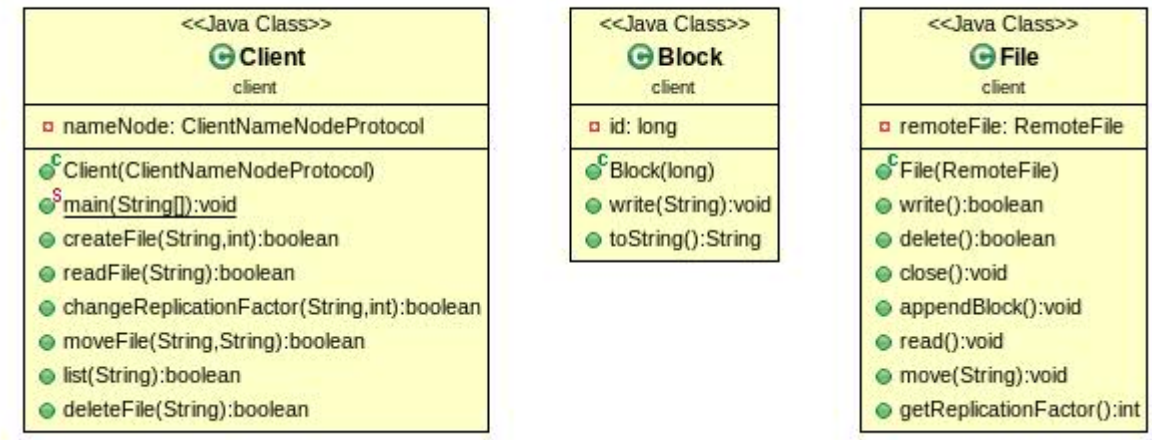


Figure 21: Class Diagram of Client Code

Class Client Details

The Class represents the Client of YAHAS. It presents a command line interface to interact with YAHAS. Through this, a User can manage files in YAHAS System.

Method Summary

`boolean`	**`changeReplicationFactor`**`(java.lang.String path, int newReplicationFactor)` Function to change replication factor of a file.
`boolean`	**`createFile`**`(java.lang.String path, int replicationFactor)` Function to create a File
`boolean`	**`deleteFile`**`(java.lang.String src)` Function to delete a file
`boolean`	**`list`**`(java.lang.String src)` Function prints list of contents of a Directory
`static void`	**`main`**`(java.lang.String[] args)` Invokes the Shell.
`boolean`	**`moveFile`**`(java.lang.String src, java.lang.String dest)` Function to move a file
`boolean`	**`readFile`**`(java.lang.String path)` Function to read a File

Constructor Detail

`public Client(ClientNameNodeProtocol nameNode)`
Parameters: `nameNode` - The implementation of the NameNode Protocol

Method Details

createFile

```
public boolean createFile(java.lang.String path,
                          int replicationFactor)
```

Function to create a File
Parameters:
`path` - Path of the file
`replicationFactor` - Indicates how many replicas of the block needs to be made
Returns:
true indicates success and false indicates failure in executing command

readFile

`public boolean readFile(java.lang.String path)`
Function to read a File
Parameters:
`path` - Path of the file to be read
Returns:
true indicates success and false indicates failure

changeReplicationFactor

```
public boolean changeReplicationFactor(java.lang.String path,
                                       int newReplicationFactor)
```

Function to change replication factor of a file.

Parameters:

`path` - Path of the File

`newReplicationFactor` - New replication Factor

Returns:

: true indicates success , false indicates failure in command execution

moveFile

```
public boolean moveFile(java.lang.String src,
                        java.lang.String dest)
```

Function to move a file

Parameters:

`src` - Source File path

`dest` - Destination File Path

list

```
ublic boolean list(java.lang.String src)
```

Function prints list of contents of a Directory

Parameters:

`src` - path of the directory

Returns:

true indicates success, false indicates failure in command execution

deleteFile

```
public boolean deleteFile(java.lang.String src)
```

Function to delete a file

Parameters:

`src` - path of file to be deleted

Returns:

true indicates success, false indicates failure in command execution

Details Class File

```
public class File
extends java.lang.Object
```

```
The class represents a File. The class acts a proxy to the remote file. A
client can get an instance of this class and perform file operations on it.
These file operations are in turn done on the RemoteFile using RMI.
```

Constructor and Description

File(RemoteFile remoteFile)

Method Summary

void	**appendBlock**() Function to append a Block
void	**close**() Function called to close a File.
boolean	**delete**() Function to delete a file
int	**getReplicationFactor**() Function to get the replication factor of a file
void	**move**(java.lang.String destPath) Function to move a file
void	**read**() Function to read a File.
boolean	**write**() Function to write data to a file

Method Details

write

```
public boolean write()
```

Function to write data to a file

Parameters:
data - : Represents a Block of data

delete

```
public boolean delete()
                throws java.rmi.RemoteException
```

Function to delete a file

Parameters:
fileName - : Path of the File to delete
Returns:
Throws:
java.rmi.RemoteException

close

```
public void close()
           throws java.rmi.RemoteException
```

Function called to close a File. In YAHAS, after blocks are added this function needs to be called

for creating a nameSpace entry in the NameNode

Throws:
`java.rmi.RemoteException`

appendBlock

```
public void appendBlock()
                  throws java.rmi.RemoteException
```

Function to append a Block

Throws:
`java.rmi.RemoteException`

read

```
public void read()
                throws java.rmi.RemoteException
```

Function to read a File.

Throws:
`java.rmi.RemoteException`

move

- `public void move(java.lang.String destPath)`
  `throws java.rmi.RemoteException`

Function to move a file

Parameters:
`destPath` - destination path of the new File
Throws:
`java.rmi.RemoteException`

getReplicationFactor

```
public int getReplicationFactor()
                          throws java.rmi.RemoteException
```

Function to get the replication factor of a file

Returns:
Replication Factor of the file
Throws:
`java.rmi.RemoteException`

- **DataNode**: The package contains the code that pertains to the Data Node. The class diagram of the classes in the package is shown below:

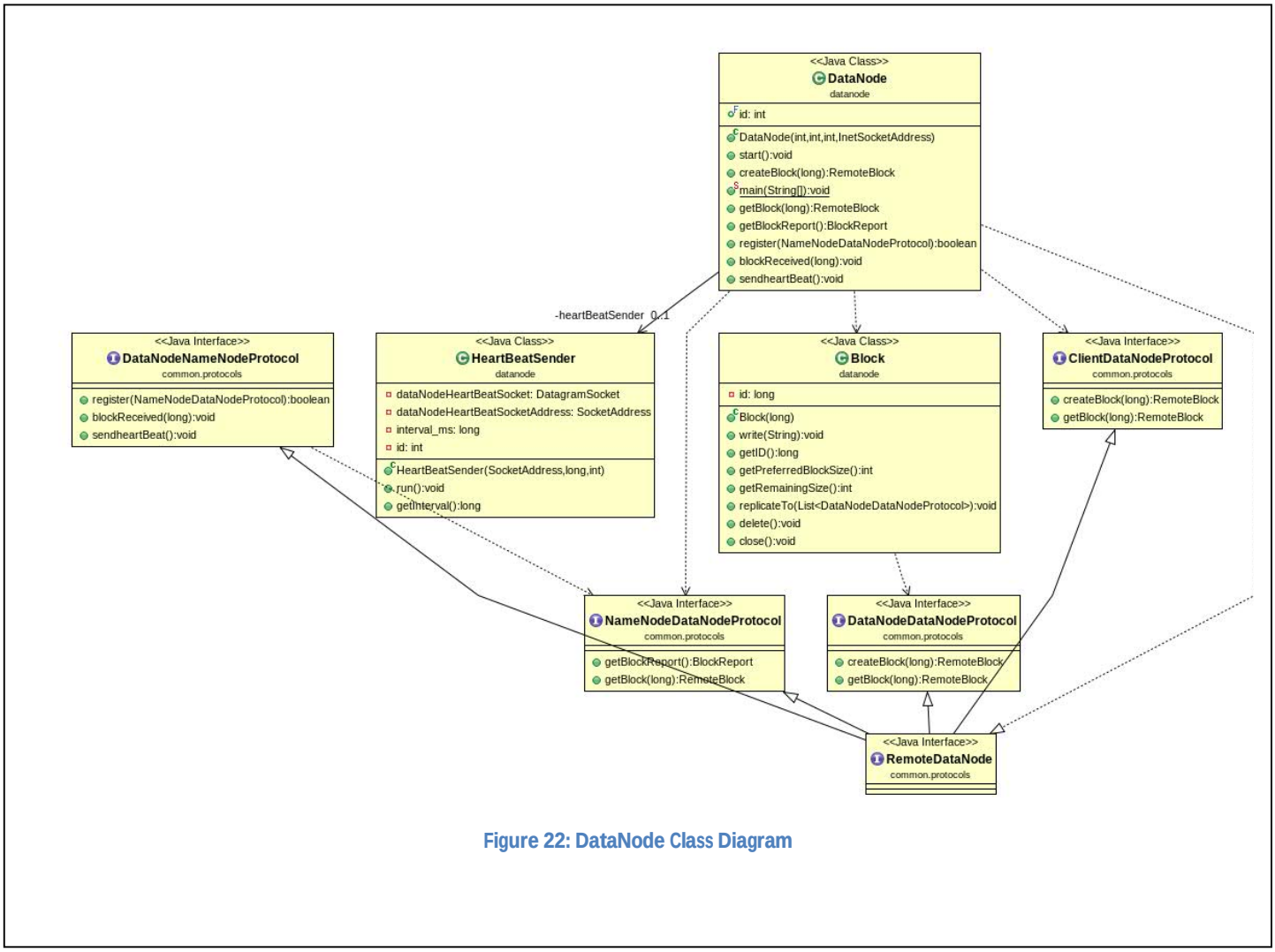


Figure 22: DataNode Class Diagram

Details of Class Block

Block is a specific implementation of a remote block. A Block resides in the data node.

Method Summary

void	**close**() Close the Block
void	**delete**() Delete the Block
long	**getID**() Gets the Id of the RemoteBlock
int	**getPreferredBlockSize**() Gets the Preferred Block Size
int	**getRemainingSize**() Gets the remaining size
void	**replicateTo**(java.util.List<DataNodeDataNodeProtocol> dataNodes) Replicate Block to the specified DataNodes
void	**write**(java.lang.String s) Write content specified by s into the block

Method Details

write

```
public void write(java.lang.String s)
           throws java.rmi.RemoteException,
                  java.io.IOException
```

Description copied from interface: RemoteBlock

Write content specified by s into the block

Specified by:
write in interface RemoteBlock
Throws:
java.rmi.RemoteException
java.io.IOException

getID

```
public long getID()
           throws java.rmi.RemoteException
```

Description copied from interface: RemoteBlock

Gets the Id of the RemoteBlock

Specified by:
getID in interface RemoteBlock
Returns:

Throws:
java.rmi.RemoteException

getPreferredBlockSize

```
public int getPreferredBlockSize()
                           throws java.rmi.RemoteException
```

Description copied from interface: RemoteBlock

Gets the Preferred Block Size

Specified by:
getPreferredBlockSize in interface RemoteBlock
Returns:
Throws:
java.rmi.RemoteException

getRemainingSize

```
public int getRemainingSize()
                     throws java.rmi.RemoteException
```

Description copied from interface: RemoteBlock

Gets the remaining size

Specified by:
getRemainingSize in interface RemoteBlock
Returns:
Throws:
java.rmi.RemoteException

replicateTo

- `public void replicateTo(java.util.List<DataNodeDataNodeProtocol> dataNodes)`
  `throws java.rmi.RemoteException`

Description copied from interface: RemoteBlock

Replicate Block to the specified DataNodes

Specified by:
replicateTo in interface RemoteBlock
Parameters:
dataNodes - A List of Data Nodes to which the blocks needs to be replicated.
Throws:
java.rmi.RemoteException

delete

```
public void delete()
```

```
                throws java.rmi.RemoteException
```

Description copied from interface: RemoteBlock

Delete the Block

Specified by:
delete in interface RemoteBlock
Throws:
java.rmi.RemoteException

close

```
public void close()
           throws java.rmi.RemoteException
```

Description copied from interface: RemoteBlock

Close the Block

Specified by:
close in interface RemoteBlock
Throws:
java.rmi.RemoteException

Class Details of Data Node

The class represents a DataNode.

Method Summary

void	**blockReceived**(long blockId) Function to indicate a Block recieved.
RemoteBlock	**createBlock**(long id) Function to create a Block
RemoteBlock	**getBlock**(long blockID) Function to get a Block specified by BlockId
BlockReport	**getBlockReport**() Function to get a Block report from Data Node
static void	**main**(java.lang.String[] args)
boolean	**register**(NameNodeDataNodeProtocol dataNode) Registers a DataNode
void	**sendheartBeat**() Function used to send a Heart beat
void	**start**() Function to start the NameNode

Details of Class HeartBeatSender

HeartBeat Sender is a sub component of the DataNode that sends Heart Beats to the Name Node.

Method Summary

ong	**getInterval**()
void	**run**()

- **NameNode**: The package contains the code that pertains to the Name Node.

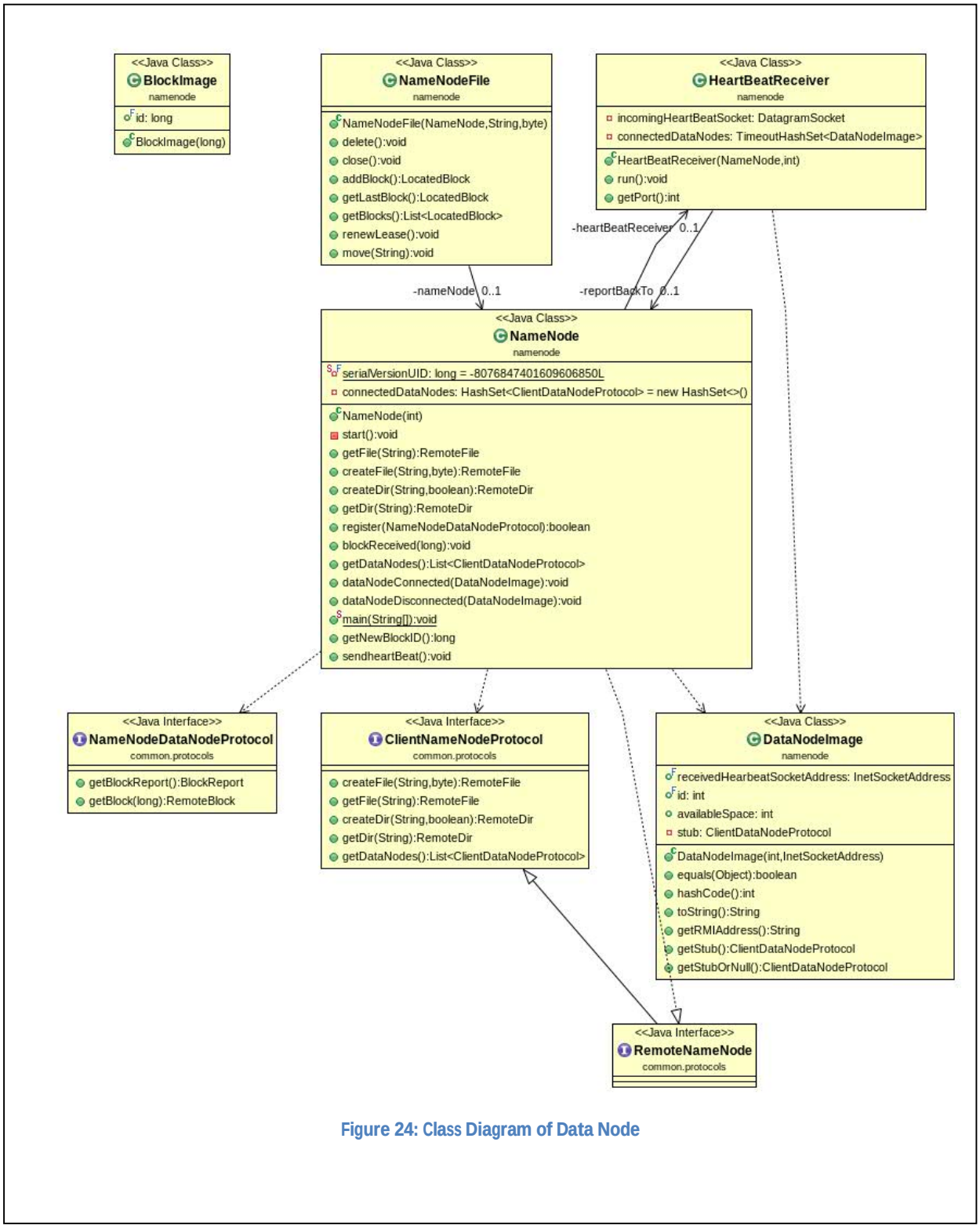


Figure 24: Class Diagram of Data Node

Class Details of Name Node

Interfaces Implemented: ClientNameNodeProtocol, DataNodeNameNodeProtocol, RemoteNameNode, java.io.Serializable, java.rmi.Remote

A specific implementation of the NameNode

Method Summary

void	**blockReceived**(long blockId) Function to indicate a Block recieved.
RemoteDir	**createDir**(java.lang.String name, boolean createParentsAsNeeded) Function to create a RemoteDirectory.
RemoteFile	**createFile**(java.lang.String name, byte replicationFactor) Create a New File
void	**dataNodeConnected**(DataNodeImage dataNodeImage)
void	**dataNodeDisconnected**(DataNodeImage dataNodeImage)
java.util.List<ClientDataNodeProtocol>	**getDataNodes**() Return a List of DataNodes
RemoteDir	**getDir**(java.lang.String name) Function to get a handle for the directory
RemoteFile	**getFile**(java.lang.String name) Returns a handle to the RemoteFile
long	**getNewBlockID**()
static void	**main**(java.lang.String[] args)
boolean	**register**(NameNodeDataNodeProtocol dataNode) Registers a DataNode
void	**sendheartBeat**() Function used to send a Heart beat

Class Details of NameNodeFile

Represents the Implementation of a file. It is noted that only the NameNode has the Notion of File.

Method Summary

LocatedBlock	**addBlock**() Function to add a block to a file.
void	**close**() Function called to close the File.
void	**delete**() Function to Delete a File
java.util.List<LocatedBlock>	**getBlocks**() Gets the Blocks associated with this File.
LocatedBlock	**getLastBlock**() Function to get the Last Block
void	**move**(java.lang.String filePathAndName) Function to move the file to a new location
void	**renewLease**() Function to renew the lease on the file

- **Protocols**

  Communication between various sub-systems of YAHAS is through specific protocols. From an implementation point of view, these protocols are implemented as Java Interfaces and specifies components implements these interfaces.

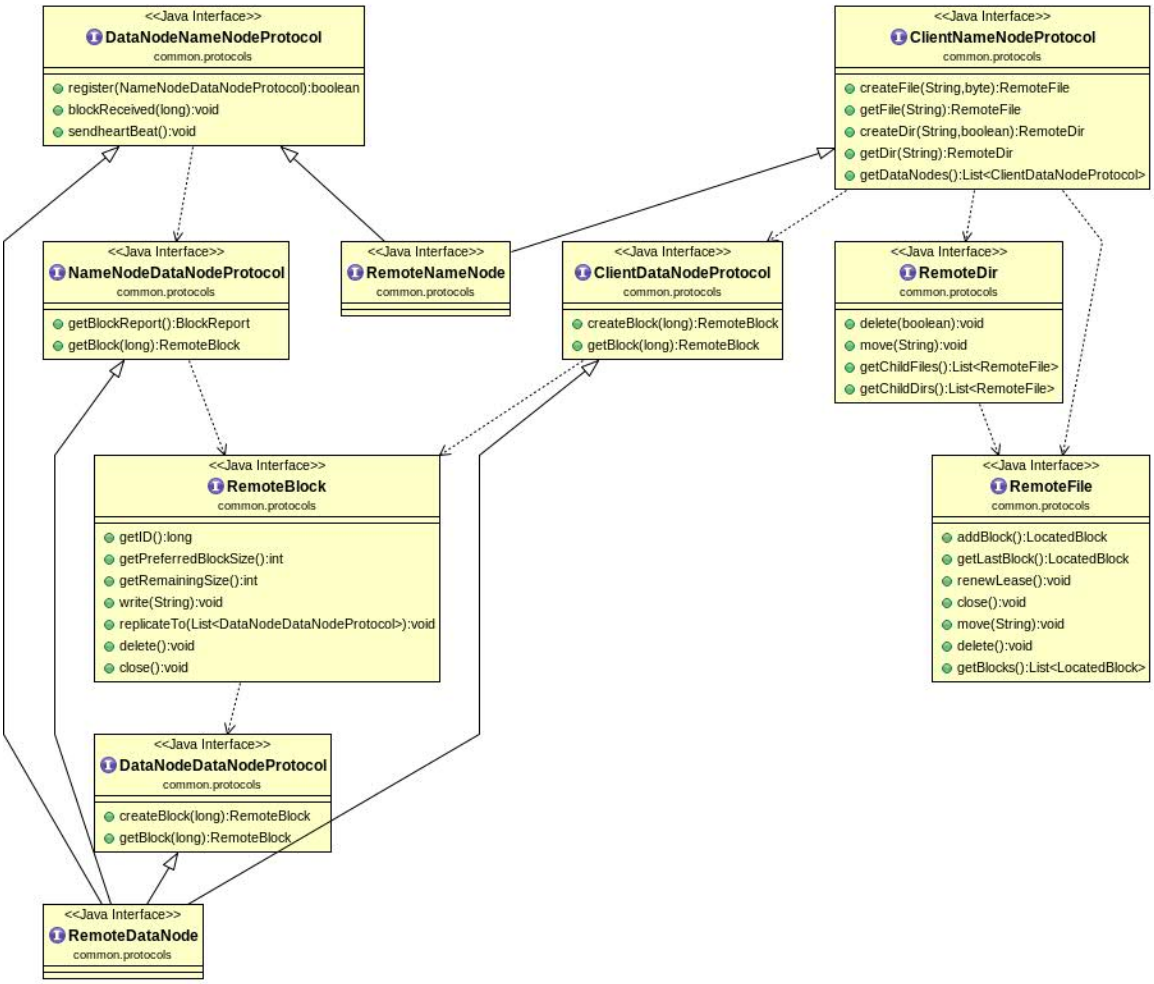


Figure 26: Class Diagram for Protocols

Details of Interface ClientDataNode Protocol	
Interface for the ClientDataNodeProtcol representing the protocol used between the client and the data node	
Method Summary	
RemoteBlock	**createBlock**(long blockID) Function to create a block
RemoteBlock	**getBlock**(long blockID) Function to get a Block specified by BlockId

Details of Interface ClientNameNodeProtocol	
Interface for the protocol between Client and NameNode. The Protocol is used to a get a file handle for RemoteFile. Once a file handle for remoteFile is obtained all operation on file can be performed	
Method Summary	
RemoteDir	**createDir**(java.lang.String name, boolean createParentsAsNeeded) Function to create a RemoteDirectory.
RemoteFile	**createFile**(java.lang.String name, byte replicationFactor) Create a New File
java.util.List<ClientDataNodeProtocol>	**getDataNodes**() Return a List of DataNodes
RemoteDir	**getDir**(java.lang.String name) Function to get a handle for the directory
RemoteFile	**getFile**(java.lang.String name) Returns a handle to the RemoteFile

Details of Interface DataNodeDataNode Protocol	
Interface for Data Node Data Node Protocol. Using this protocol, two data nodes can communicate with each other.	
Method Summary	
RemoteBlock	**createBlock**(long blockID) Create a New Block.
RemoteBlock	**getBlock**(long blockID) Returns the Handle of a Block

Details of Interface DataNodeNameNode Protocol

Representation of the protocol used between Data Node and Name Node

Method Summary

`void`	**`blockReceived`**`(long blockId)` Function to indicate a Block recieved.
`boolean`	**`register`**`(NameNodeDataNodeProtocol dataNode)` Registers a DataNode
`void`	**`sendheartBeat`**`()` Function used to send a Heart beat

Details of Interface NameNodeDataNode Protocol

Represents a Protocol for Name Node Data Node Communication.

Method Summary

`RemoteBlock`	**`getBlock`**`(long blockID)` Function to get a Block
`BlockReport`	**`getBlockReport`**`()` Function to get a Block report from Data Node

Details of Interface RemoteBlock

Represents a Block present in the DataNode. A client program gets a handle to this remote block and manipulates the same using RMI

Method Summary

`void`	**`close`**`()` Close the Block
`void`	**`delete`**`()` Delete the Block
`long`	**`getID`**`()` Gets the Id of the RemoteBlock
`int`	**`getPreferredBlockSize`**`()` Gets the Preferred Block Size
`int`	**`getRemainingSize`**`()` Gets the remaining size
`void`	**`replicateTo`**`(java.util.List<DataNodeDataNodeProtocol> dataNodes)` Replicate Block to the specified DataNodes
`void`	**`write`**`(java.lang.String s)` Write content specified by s into the block