CSS434: Parallel and Distributed Computing

Professor Munehiro Fukuda

Program 4: Distributed File System

Christopher Tran

# Implementation

## DFS Server

Based on the program specifications, Server portion keeps track of files and all the details on usage and function are isolated to each file. After observation of the existing program, I also decided to do a separate class that managed each file.

The file class keeps track of the file state based. States are either NONE, READ, WRITE, TODISK. The download function sends the file to the client. State will change based on the r or w mode given to the function. If another user has the file open for writing, we call writeback to tell the owner to return ownership and invalidate their write status. Clients keeps an active thread listening for the server to force a writeback.

Upload downloads the file back to the server. First it will invalidate any active copies being read. Then it will write the contents to the disk.

## DFSClient

Based on observation, there must be an infinite loop to take in file names and modes. Much like Server portion, Client also has a File class that keeps a state of the file stored locally. One of the functions will determine if the Client has the file based on conditions of state. Downloading will change the state to read or write depending on the mode given. Client will then attempt to download the file using the server interface. Uploading will also use the server interface to upload FileContents. After each attempt to download a file for reading and writing, the file will be open in emacs. Based on the file mode read or write, we will use permissions 400 for read only and 600 for read write.

# Execution Output

Screenshots In Lab need to be taken

# Discussion and Improvements

## Timeout

A heartbeat implementation could be set that would automatically remove any clients that did not respond after a period of time. When a client exits, the server still believes that the file is being read by the client.

# Source Code

## DFSClient

import java.io.\*;

import java.net.InetAddress;

import java.net.UnknownHostException;

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

/\*\*

\* Created by Christopher on 6/8/2014.

\*/

public class DFSClient extends UnicastRemoteObject implements ClientInterface {

private BufferedReader input = null;

public static final String tempFile = "/tmp/ctran206.txt";

protected ServerInterface server = null;

protected File file = null;

public DFSClient(String serverName, String port) throws RemoteException {

try {

//Connect to Server

server = (ServerInterface)Naming.lookup("rmi://" + serverName + ":" + port + "/dfsserver");

} catch (Exception e) {

e.printStackTrace();

System.exit(-1);

}

//Get new file class and connect to server

file = new File(this);

//For console input

input = new BufferedReader(new InputStreamReader(System.in));

}

public static void main(String[] args) {

//Must have two arguements for server and port

if(args.length != 2) {

System.out.println("Usage: java DFSClient serverIP Port#");

System.exit(-1);

}

try {

//make object to run as DFS Client

DFSClient dfsClient = new DFSClient(args[0],args[1]);

//Rebind for RMI

Naming.rebind("rmi://localhost:" + args[1] + "/dfsclient", dfsClient);

//Output for console verification

System.out.println("rmi://localhost:" + args[1] + "/dfsclient invoked");

dfsClient.run();

} catch (Exception e) {

e.printStackTrace();

System.exit(1);

}

}

//Infinite loop function to access files from DFSServer

private void run() {

//run indefinitely

while(true) {

//Inspired by Fukuda example program, aditional class to

WriteBackThread wbt = new WriteBackThread(this);

wbt.start();

String fileName = null;

String fileMode = null;

//Get user input to download file

System.out.print("DFSClient: Next file to open\n\tFileName or quit to exit: ");

try {

fileName = input.readLine();

} catch (IOException e) {

System.err.println("Error input for file name");

e.printStackTrace();

}

//if input is blank, try again

if(fileName.equals("")) {

wbt.kill();

continue;

}

//IF the user wants to quit, write file back if needed

if(fileName.equals("quit")) {

if(file.isWrite()) {

file.upload();

}

wbt.kill();

System.exit(0);

}

//Get file mode

System.out.print("\tHow(r\\w): ");

try {

fileMode = input.readLine();

} catch (IOException e) {

System.err.println("Error input for file mode");

e.printStackTrace();

}

//Check if anything but r and w

if(!fileMode.equals("r") && !fileMode.equals("w")) {

System.err.println("File mode not recognized");

wbt.kill();

continue;

}

wbt.kill();

boolean success = false;

//If the file is not already stored, download the file

if(!file.hasFile(fileName,fileMode)) {

//Write back any existing file

if(file.isWrite()) {

file.upload();

}

success = file.download(fileName,fileMode);

}

//If file was downloaded, open in editor

if(!file.isNullFile()) {

file.runEditor(fileMode);

} else {

System.err.println("File unable to download or was null");

continue;

}

}

}

public boolean invalidate() {

return file.invalidate();

}

public boolean writeback() {

return file.writeback();

}

class File {

//State variables

private static final int INVALID = 0;

private static final int READ = 1;

private static final int WRITE = 2;

private static final int WRITEBACK = 3;

private int state = INVALID; //start at invalid

private String fileName = "";

private boolean isOwned = false;

private byte[] fileData = null;

private String localName = null;

public File(DFSClient client) {

try {

InetAddress iNet = InetAddress.getLocalHost();

localName = iNet.getHostName();

} catch (UnknownHostException e) {

e.printStackTrace();

}

}

//State Check Function

public synchronized boolean isInvalid() {

return state == INVALID;

}

//State Check Function

public synchronized boolean isReadShared() {

return state == READ;

}

//State Check Function

public synchronized boolean isWrite() {

System.out.println("Name: " + fileName + "\tState: " + state + "\tOwnership: " + isOwned);

return state == WRITE;

}

//State Check Function

public synchronized boolean isToWriteBack() {

return state == WRITEBACK;

}

//Change state to Invalid

//Invalidate read only copy

public synchronized boolean invalidate() {

if(state == READ) {

state = INVALID;

return true;

}

return false;

}

//Writee back data

public synchronized boolean writeback() {

if(state == WRITE) {

state = WRITEBACK;

return true;

}

return false;

}

//Check if local cache has file

public synchronized boolean hasFile(String file, String mode) {

//Check if client has file

if(!fileName.equals(file)) {

System.out.println("File: " + file + " does not exist.");

return false;

}

//Check if file needs to be written to server so client doesn't have proper file

if(state == WRITEBACK) {

System.out.println("File: " + file + " must be written back.");

return false;

}

//Client has file for reading

if(state != INVALID && mode.equals("r")) {

System.out.println("File: " + file + " exist for read.");

return true;

}

//Client has file for writing

if(state == WRITE && mode.equals("w")) {

System.out.println("File: " + file + " is owned for write");

return true;

}

System.out.println("File: " + file + " accessed with " + mode);

return false;

}

//Download file to local cache

public boolean download(String file, String mode) {

System.out.println("Downloading: " + file + " with " + mode + " mode");

synchronized (this) {

//If state is invalid, set the mode to read or write depending on the input

if(mode.equals("r")) {

state = READ;

} else if(mode.equals("w")) {

state = WRITE;

} else {

//r or w not detected, return false

return false;

}

}

fileName = file;

isOwned = (state == WRITE);

try {

FileContents fc = DFSClient.this.server.download(localName,fileName,mode);

fileData = fc.get();

} catch( RemoteException e) {

e.printStackTrace();

return false;

}

return true;

}

//Upload file to server

public boolean upload() {

System.out.println("Uploading: " + fileName + " start");

synchronized (this) {

if(state == WRITE) {

state = INVALID;

} else if(state == WRITEBACK) {

state = READ;

} else {

//State is either invalid or read, return false since we shouldn't upload

return false;

}

}

FileContents fc = new FileContents(fileData);

//Upload file data

try {

DFSClient.this.server.upload(localName,fileName,fc);

} catch (RemoteException e) {

e.printStackTrace();

return false;

}

System.out.println("Upload: " + fileName + " completed");

return true;

}

//Run the Text Editor: emacs

public Boolean runEditor(String fileMode) {

//Get runtime to execute commands

Runtime runtime = Runtime.getRuntime();

//Generate String array to execute command with exec

//First command is changing temp file to OWNER:READandWRITE

String[] command = {"chmod","600",DFSClient.this.tempFile};

//Try to run command

try {

Process process = runtime.exec(command);

process.waitFor();

} catch (IOException e) {

e.printStackTrace();

return false;

} catch (InterruptedException e) {

e.printStackTrace();

return false;

}

//Output file data from File to temp file

try {

FileOutputStream fos = new FileOutputStream(DFSClient.this.tempFile);

fos.write(fileData);

fos.flush();

fos.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

return false;

} catch (IOException e) {

e.printStackTrace();

return false;

}

//If mode is read, change mode to read only

if(fileMode.equals("r")) {

//Change index 1 from 600 to 400 for OWNER:READ

command[1] = "400";

try {

Process process = runtime.exec(command);

process.waitFor();

} catch (IOException e) {

e.printStackTrace();

return false;

} catch (InterruptedException e) {

e.printStackTrace();

return false;

}

}

//Open emacs

command = new String[]{"emacs", DFSClient.this.tempFile};

try {

Process process = runtime.exec(command);

process.waitFor();

} catch (IOException e) {

e.printStackTrace();

return false;

} catch (InterruptedException e) {

e.printStackTrace();

return false;

}

//If write mode, write data back to temp file

if(fileMode.equals("w")) {

try {

FileInputStream fis = new FileInputStream(DFSClient.this.tempFile);

fileData = new byte[fis.available()];

fis.read(fileData);

fis.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

return false;

} catch (IOException e) {

e.printStackTrace();

return false;

}

}

return true;

}

//If data downloaded is null, file is probably incorrect

public boolean isNullFile() {

return fileData == null;

}

}

class WriteBackThread extends Thread {

private boolean isRunning = false;

public WriteBackThread(DFSClient client) {

isRunning = true;

}

//Writeback when state is toRead

public void run() {

while(isRunning) {

if(DFSClient.this.file.isToWriteBack()) {

DFSClient.this.file.upload();

}

}

}

//Kill thread by joining

synchronized void kill() {

isRunning = false;

try {

join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

## DFSServer

import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

import java.io.\*;

import java.util.\*;

/\*\*

\* Created by Christopher on 6/6/2014.

\*/

public class DFSServer extends UnicastRemoteObject implements ServerInterface{

private String port = null;

private Vector<File> cache = null;

public DFSServer(String args) throws RemoteException {

port = args;

cache = new Vector<File>();

}

public static void main(String[] args) {

if(args.length != 1) {

System.out.println("Usage: java DFSServer port#");

System.exit(-1);

}

try {

DFSServer server = new DFSServer(args[0]);

Naming.rebind("rmi://localhost:" + args[0] + "/dfsserver", server);

System.out.println("rmi://localhost: " + args[0] + "/dfsserver invoked");

} catch(Exception e) {

e.printStackTrace();

System.exit(1);

}

}

//send file to client file to client

public FileContents download(String client, String filename, String mode) throws RemoteException {

File file = null;

synchronized (cache) {

//Check if file already exists in the cache

for(int i = 0; i < cache.size(); i++) {

file = cache.elementAt(i);

//If file has been found, break

if(file.isFile(filename)) {

break;

}

file = null;

}

//if file is not found in cache, get a new file.

if(file == null) {

file = new File(this, filename, port);

cache.add(file);

}

}

return file.download(client,mode);

}

//get file from clients

public boolean upload(String client, String filename, FileContents contents) throws RemoteException {

File file = null;

synchronized (cache) {

//Check if file exists to be written back to

for(int i = 0; i < cache.size(); i++) {

file = cache.elementAt(i);

if(file.isFile(filename)) {

break;

}

file = null;

}

}

//If cache has file, attempt to update the contents

if(file != null) {

System.out.println(client + " upload: file " + filename);

return file.upload(client, contents);

}

return false;

}

class File {

//Condition by value

private static final int NONE = 0;

private static final int READ = 1;

private static final int WRITE = 2;

private static final int TODISK = 3;

//Class variables

private int state; //Stores current state

private String fileName; //Stores the name of the file

private byte[] fileData; //Stores the file data

private Vector<String> readers = null; //Container for all readers of file

private String writeUser = null; //user that is writing to file

private String port = null; //port

private Thread toWrite = null; //Synchronized object for readers to wait on for writing

//Constructor

public File(DFSServer dfs, String name, String port) {

state = NONE; //State begins with none

this.fileName = name; //Store file name

readers = new Vector<String>();

writeUser = null;

this.port = port;

toWrite = new Thread();

fileData = readFromFile();

}

//Read a file from the directory

private byte[] readFromFile() {

byte[] fileData = null;

//REad file into byte array and return null data at any exception

try {

FileInputStream fin = new FileInputStream(fileName);

fileData = new byte[fin.available()];

fin.read(fileData);

fin.close();

} catch(FileNotFoundException e) {

e.printStackTrace();

return null;

} catch(IOException e) {

e.printStackTrace();

return null;

}

System.out.println("File read from " + fileName + ": " + fileData.length + " bytes");

return fileData;

}

//Write to file in directory

private boolean writeToFile() {

try {

FileOutputStream fos = new FileOutputStream(fileName);

fos.write(fileData);

fos.flush();

fos.close();

} catch(FileNotFoundException e) {

e.printStackTrace();

return false;

} catch (IOException e) {

e.printStackTrace();

return false;

}

System.out.println("File written to " + fileName + ": " + fileData.length + " fileData");

return true;

}

//download will upload a file for client to download

public synchronized FileContents download(String reader, String mode) {

//If no file name, no need to do anything

if(fileName.equals("")) {

return null;

}

//Check if file is waiting to be writen back and halt download until finished

while(state == TODISK) {

synchronized (toWrite) {

try{

System.out.println(reader + " now wait on file writing");

toWrite.wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

//

int currentState = state;

boolean success = true;

switch(state) {

case NONE:

//If file isn't in a state, change to read or write

if(mode.equals("r")) {

state = READ;

readers.add(reader);

} else if(mode.equals("w")) {

state = WRITE;

if(writeUser == null) {

writeUser = reader;

} else {

success = false;

}

} else {

success = false;

}

break;

case READ:

//file is in read state and needs to check if adding to readers or writing

deleteReader(reader);

if(mode.equals("r")) {

readers.add(reader);

} else if(mode.equals("w")) {

state = WRITE;

if(writeUser == null) {

writeUser = reader;

} else {

success = false;

}

} else {

success = false;

}

break;

case WRITE:

//File is in write mode for another user

deleteReader(reader);

if(mode.equals("r")) {

readers.add(reader);

} else if (mode.equals("w")) {

state = TODISK;

//Connect to client that has the current write and force them to write back to server

ClientInterface ci = null;

try {

ci = (ClientInterface)Naming.lookup("rmi://" + writeUser + ":" + port + "/dfsclient");

} catch (Exception e) {

e.printStackTrace();

success = false;

}

//Tell current writer to write back to server

if (ci != null) {

try {

ci.writeback();

} catch (RemoteException e) {

e.printStackTrace();

success = false;

}

System.out.print("Download( " + fileName + " ): " + writeUser + "'s copy was invalidated");

}

if(success) {

try {

System.out.println("Download " + fileName + " ): " + reader + " waits for writeback");

wait();

} catch (InterruptedException e) {

e.printStackTrace();

success = false;;

}

writeUser = reader;

}

} else {

success = false;;

}

break;

}

printReport(reader, "download", mode, currentState, state);

FileContents fc = new FileContents(fileData);

if(success) {

if (currentState == TODISK) {

synchronized (toWrite) {

toWrite.notifyAll();

System.out.println(reader + " making up all threads waiting on write");

}

}

} else {

return null;

}

return fc;

}

//Get file from client

public synchronized boolean upload(String reader, FileContents source) {

boolean success = true;

//Can't upload if file is blank or in none or read state

if(fileName.equals("") || state == NONE || state == READ) {

success = false;

}

//Invalidate all readers copies

for( int j = 0; success && j < readers.size(); j++) {

String currentReader = readers.elementAt(j);

ClientInterface ci = null;

try {

ci = (ClientInterface) Naming.lookup("rmi://" + currentReader + ":" + port + "/dfsclient");

} catch (Exception e) {

e.printStackTrace();

success = false;

}

if(ci == null) {

try {

ci.invalidate();

} catch(RemoteException e) {

e.printStackTrace();

}

System.out.println("Update( " + fileName + " ): " + currentReader + "'s copy was invalidated");

}

}

//Readers are no longer valid so remove them all

readers.removeAllElements();

//Keep old state for reporting

int currentState = state;

if(success) {

//Get the data

fileData = source.get();

System.out.println("Bytes = " + fileData);

switch(state) {

case WRITE:

state = NONE;

writeUser = null;

success = writeToFile();

break;

case TODISK:

state = WRITE;

writeUser = reader;

notify();

}

}

printReport(reader, "upload", "w", currentState, state);

return success;

}

//Remove reader from active list of file readers

private boolean deleteReader(String reader) {

for(int i = 0; i < readers.size(); i++) {

if(readers.elementAt(i).equals(reader)) {

readers.remove(i);

return true;

}

}

return false;

}

//Show the users that are active on the file

private void showReaders() {

System.out.println("# Readers = " + readers.size());

for(int i = 0; i < readers.size(); i++) {

System.out.println("\tReader = " + readers.elementAt(i));

}

}

public synchronized boolean isNotShared() {

return state == NONE;

}

public synchronized boolean isReadShared() {

return state == READ;

}

public synchronized boolean isWriteShared() {

return state == WRITE;

}

public synchronized boolean isToWriteShared() {

return state == TODISK;

}

//Check if the current file exists

public synchronized boolean isFile(String target) {

return fileName.equals(target);

}

//Print out transition reports

private void printReport(String clientName, String task, String stringMode, int oldState, int newState) {

String fromState = stateToString(oldState);

String toState = stateToString(newState);

System.out.println("File(" + fileName + ") requested by " + clientName + ":" + task + "(" + stringMode + "): state( " + fromState + " --> " + toState + " )");

showReaders();

System.out.println("Owner = " + writeUser);

}

//Get string of state for transition report

private String stateToString(int inState) {

switch(inState) {

case NONE:

return "notshared";

case READ:

return "readshared";

case WRITE:

return "writeshared";

case TODISK:

return "back2writeshared";

default:

return "notrecognized";

}

}

}

}