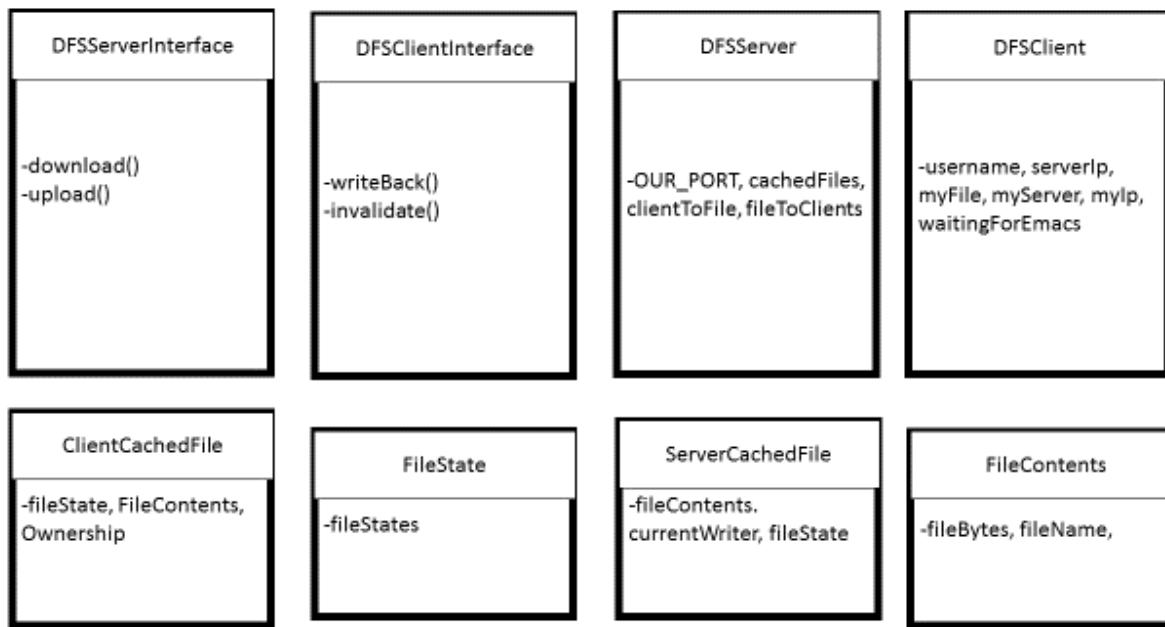


Documentation

Figure 1: Classes



Server

Brief

The server follows the logic described by the project description; ensuring that all the clients' file-states adhere to logic described in the state transition diagrams. It's how we factored the code that makes our project stand out.

The server relies on the setting of its currently cached file's file-state, represented in the enum FileState.

Flow

Instantiating a server is easy. We use a default port so you only need to pass the server its' own IP as the first argument.

The server is factored out into a collection of functions that worked to simplify debugging of the program and manage the collection of cached files. There're are three main data structures: two hash-maps; a map of client to file, and a map of file to clients; and one vector of cached files.

The cached files are stored as instances of ServerCachedFile which contains an instance of the provided FileContents class as well as a FileState and a string with the current client with write permissions.

Whenever the methods upload or download are called, the state of the ServerCachedFile is updated as well as the hash-maps with references to what client has what file cached. These updates take place after guaranteeing all other clients currently caching that file now have to correct FileState. In the case of ownership change of a file, the server changes the FileState of a cached file to Ownership_Change and blocks. The download() function remains blocked

until the current owner of a file responds to a writeBack() call and uploads a new version of the file.

[Client](#)

[Brief](#)

Client maintains the program flow dictated by the state transition diagram shown in the project documentation. By factoring out the code into many discrete functions for ease of debugging.

[Flow](#)

Opening a client requires two arguments, its' own IP, and the server IP. The port is a constant the client sources from the server class. The client then runs in two discrete stages, a startup, and a main loop.

The startup is executed with the main function. Where here error checking takes place and the client creates an rmi connection to the server as well as adding itself to the rmi.

The second part of the client works in three parts.

First the client gets a file from the server, then writes the file into its' own cache. The client's file state is guaranteed by the server. The client will always download a new version of the file from the server, unless the file state is not invalid and the ownership of the file has not been altered. Before writing reading a file. The client checks if it needs to upload its currently cached file.

Next the client runs emacs.

Finally the client checks if it has a file-state of Release_Ownership, in this case it uploads its current file and changes its file state to Read_Shared.

[Other Classes](#)

The two interfaces DFSServerInterface and DFSClientInterface are there to allow the use of the java rmi package.

ServerCachedFile and ClientCachedFile wrap FileContents and hold additional information about a file particular to either the server or the client.

FileContents was provided.

FileStates is an enum with the a collection of file-states used by both the server and the clients.

Output

1. Compilation

The image displays four terminal windows arranged in a 2x2 grid, showing the output of Java commands related to the DFS system.

- Top Left Terminal:** Shows the compilation of Java files. The user runs `javac *.java`, followed by `nano /tmp/demoA` and `nano /tmp/demoB`. Then, they run `rmic DFSServer` and `java DFSServer`.
- Top Right Terminal:** Shows the client side. The user runs `rmic DFSClient`, followed by `java DFSClient uw1-320-02 uw1-320-12`. The client prompts "File Client: Next file to open" and "File name: []".
- Bottom Left Terminal:** Shows the client side. The user runs `rmic DFSClient`, followed by `java DFSClient uw1-320-15 uw1-320-12`. The client prompts "File Client: Next file to open" and "File name: []".
- Bottom Right Terminal:** Shows the client side. The user runs `rmic DFSClient`, followed by `java DFSClient uw1-320-03 uw1-320-12`. The client prompts "File Client: Next file to open" and "File name: []".

2. File read test

brandenlivermore — lbranden@uw1-320-19:~/434/hw4 — ssh lbranden@uw1-...

```
[lbranden@uw1-320-12:~/434/hw4$ javac *.java
[lbranden@uw1-320-12:~/434/hw4$ nano /tmp/demoA
[lbranden@uw1-320-12:~/434/hw4$ nano /tmp/demoB
[lbranden@uw1-320-12:~/434/hw4$ rmic DFSServer
[lbranden@uw1-320-12:~/434/hw4$ java DFSServer
File name: demoA not found. Attempting to load from disk.
About to return file: demoA
```

brandenlivermore — lbranden@uw1-320-21:~/434/hw4 — ssh lbranden@uw1-...

File Edit Options Buffers Tools Help

```
-UU-:%%--F1 lbranden.txt All L1 (Text) -----
Note: file is write protected
```

3. File write test

The image displays four terminal windows arranged in a 2x2 grid, illustrating the interaction between a Java DFSClient and a Java DFSServer.

- Top Left Terminal:** Shows the compilation and execution of the Java DFSClient. It includes commands like javac, nano, rmic, and java, along with log messages indicating the client is attempting to load from disk and requesting a write back.
- Top Right Terminal:** Shows the Java DFSServer running. It receives a write request for file "demoA" and logs the file contents ("File is not cached!", "Got file contents!"), then attempts to exec emacs.
- Bottom Left Terminal:** Shows a nano editor session for file "lbranden.txt". The buffer contains "xyz".
- Bottom Right Terminal:** Shows the Java DFSClient again, this time attempting to open the file "lbranden.txt" on the server.

4. File replacement test

The screenshot shows a terminal session with four panes. The top-left pane shows the server-side log of a Java application (lbranden.txt) being uploaded. The log includes commands like javac, nano, rmic, and java, along with file transfer details. The top-right pane shows the client-side Emacs editor displaying the file 'lbranden.txt' with content 'xyz'. The bottom-left pane shows the client-side log where the user attempts to open 'lbranden.txt' and upload a new file 'randen.txt'. The bottom-right pane shows the client-side Emacs editor displaying the file 'lbranden.txt' with content 'xyz', indicating the file was write-protected.

```
lbranden@uw1-320-12:~/434/hw4$ javac *.java
lbranden@uw1-320-12:~/434/hw4$ nano /tmp/demoA
lbranden@uw1-320-12:~/434/hw4$ nano /tmp/demoB
lbranden@uw1-320-12:~/434/hw4$ rmic DFFServer
lbranden@uw1-320-12:~/434/hw4$ java DFFServer
File name: demoA not found. Attempting to load from disk.
About to return file: demoA
About to return file: demoA
Requesting write back!
Requesting writeback for uw1-320-02
Received file! Name: demoA
Writing file back to disk!
Write back request completed.
Write back completed.
About to return file: demoA
Received file! Name: demoA
Writing file back to disk!
File name: demoB not found. Attempting to load from disk.
About to return file: demoB
About to return file: demoA
About to return file: demoA
File Client: Next file to open
File name:demoB
How(r/w)
w
File is not cached!
Uploading file!randen.txt  All L1  (Text) -----
Got file contents!
Username is lbranden
Trying to exec emacs
File Client: Next file to open
File name:[]
```

```
File Edit Options Buffers Tools Help
xyz
```

```
-UU-:%%--F1 lbranden.txt All L1  (Text) -----
Note: file is write protected
```

```
File Edit Options Buffers Tools Help
xyz
```

```
-UU-:%%--F1 lbranden.txt All L1  (Text) -----
Note: file is write protected
```

5. File writeback test

The screenshot displays four terminal windows illustrating a file writeback test between a DFS Server and a DFS Client.

- Terminal 1 (DFS Server):** Shows the server handling a request for file "demoA". It logs: "File name: demoA not found. Attempting to load from disk.", "About to return file: demoA", "Requesting write back!", "Requesting writeback for uw1-320-03", "Received file! Name: demoA", "Writing file back to disk!", "Write back request completed.", "About to return file: demoA", and "About to return file: demoA".
- Terminal 2 (DFS Client):** Shows the client opening file "lbranden.txt" in Emacs. The buffer content is "xyz?!abc". The status bar indicates: "UU-----F1 lbranden.txt All L1 (Text)".
- Terminal 3 (DFS Client):** Shows the client performing a read operation ("r") on file "demoA". The log includes: "File Client: Next file to open", "File name:demoA", "How(r/w)", "File is not cached!", "Got file contents!", "Username is lbranden", "Trying to exec emacs", "File Client: Next file to open", "File name:Invalidate!", "demoA", "How(r/w)", "File is not cached!", "Got file contents!", "Username is lbranden", "Trying to exec emacs", "File Client: Next file to open", "File name:demoA", "How(r/w)", "File is not cached!", "Got file contents!", "Username is lbranden", "Trying to exec emacs", "File Client: Next file to open", "File name:demoA", "How(r/w)", and "File name:demoA".
- Terminal 4 (DFS Client):** Shows the client performing a write operation ("w") on file "demoA". The log includes: "File Client: Next file to open", "File name:demoA", "How(r/w)", "File is not cached!", "Got file contents!", "Username is lbranden", "Trying to exec emacs", "File Client: Next file to open", "File name:Received write back request!", "demoA", "How(r/w)", "File is not cached!", "Got file contents!", "Username is lbranden", "Trying to exec emacs", "File Client: Next file to open", "File name:demoA", "How(r/w)", and "File name:demoA".

6. Session semantics read test

The screenshot displays four terminal windows arranged in a 2x2 grid, illustrating the interaction between a DFSClient and a DFSServer.

- Top Left Terminal:** Shows the DFSServer process (lbranden@uw1-320-12:~/434/hw4\$ java DFSServer). It logs that file "demoA" was not found and attempts to load from disk. It also logs receiving a writeback request for "demoA" and writing it back to disk.
- Top Right Terminal:** Shows the DFSClient (lbranden@uw1-320-02:~/434/hw4\$ java DFSClient uw1-320-02 uw1-320-12) attempting to open file "demoA". It logs the file is not cached, getting file contents, and trying to exec emacs.
- Bottom Left Terminal:** Shows the DFSServer process (lbranden@uw1-320-21:~/434/hw4\$ ssh lbranden@uw1-...). It logs receiving a writeback request for "demoA" and writing it back to disk.
- Bottom Right Terminal:** Shows the DFSClient (lbranden@uw1-320-22:~/434/hw4\$ ssh lbranden@uw1-...) attempting to open file "demoA". It logs the file is not cached, getting file contents, and trying to exec emacs.

Below the terminals, two Emacs buffers are visible:

- Left Buffer:** Shows the file "lbranden.txt" with content "xyz!". The status bar indicates it's a write-protected file.
- Right Buffer:** Shows the file "lbranden.txt" with content "xyz!". The status bar indicates it's a write-protected file.

7. Multiple write test

We were a bit confused about this one. At the end of the test, it says to make sure demoB is equal to '123' but after writing to the file and receiving a writeback request, the next client will receive those changes. In this test, all clients but the last one to edit the file would receive the writeback request, so I would expect to see all but the last change in the file. In any case, I included both results where I saved the files (the result is different than what we were told is expected) and results where I did not save the files after modifying (and we get the expected results). The results where we saved the changes come first.

The image shows four terminal windows side-by-side, each representing a different client or node in the distributed file system. The windows are arranged horizontally and show the following content:

- Top Left Terminal:** Shows a loop of "Still waiting." messages.
- Top Right Terminal:** Shows the file "lbranden.txt" being edited with the contents "123pqr".
- Bottom Left Terminal:** Shows a series of system log messages related to package installations and Java command-line arguments.
- Bottom Right Terminal:** Shows a sequence of file operations between a client and a server, including opening files, reading contents, and performing write-back requests.

Alex s. Ducken, Branden Livermore: CSS 434 Homework 4 Distributed File System

The image shows four terminal windows arranged in a 2x2 grid. The top row contains two windows, and the bottom row contains two windows.

- Top Left Window:** Shows the command `lbranden@uw1-320-01:~\$` followed by a long sequence of "Still waiting." messages.
- Top Right Window:** Shows the command `lbranden@uw1-320-02:~/434/hw4\$` followed by a series of file operations: opening 'demoB' for write ('How(r/w)'), invalidating it, and then opening 'demoB' again for write ('How(r/w)'). It also shows a session on host 01.
- Bottom Left Window:** Shows the command `lbranden@uw1-320-06:~\$` followed by a long sequence of "Still waiting." messages.
- Bottom Right Window:** Shows the command `lbranden@uw1-320-07:~\$` followed by a series of file operations: opening 'demoB' for write ('How(r/w)'), invalidating it, and then opening 'demoB' again for write ('How(r/w)'). It also shows sessions on hosts 01 and 02.

The image shows four terminal windows arranged in a 2x2 grid, similar to the one above but with different content.

- Top Left Window:** Shows the command `lbranden@uw1-320-01:~\$` followed by a long sequence of "Still waiting." messages. It also shows a message about receiving a file named 'demoB' and writing it back to disk.
- Top Right Window:** Shows the command `lbranden@uw1-320-02:~/434/hw4\$` followed by a series of file operations: opening 'demoB' for write ('How(r/w)'), invalidating it, and then opening 'demoB' again for write ('How(r/w)'). It also shows sessions on hosts 01 and 06.
- Bottom Left Window:** Shows the command `lbranden@uw1-320-06:~\$` followed by a long sequence of "Still waiting." messages. It also shows a message about receiving a file named 'demoB' and writing it back to disk.
- Bottom Right Window:** Shows the command `lbranden@uw1-320-07:~\$` followed by a series of file operations: opening 'demoB' for write ('How(r/w)'), invalidating it, and then opening 'demoB' again for write ('How(r/w)'). It also shows sessions on hosts 01, 02, and 06.

Alex s. Ducken, Branden Livermore: CSS 434 Homework 4 Distributed File System

```
[lbranden@uw1-320-01:~/434/hw4$ cat /tmp/demoA
xyz?!
[lbranden@uw1-320-01:~/434/hw4$ cat /tmp/demoB
123pqrs456
[lbranden@uw1-320-01:~/434/hw4$ ]
```

```
[lbranden@uw1-320-02:~/434/hw4$ java DFSClient uw1-320-02 uw1-320-01
File Client: Next file to open
File name:C\lbranden@uw1-320-02:~/434/hw4$ java DFSClient uw1-320-02 uw1-320-01
File Client: Next file to open
File name:demoB
How(r/w)
w
File is not cached!
Got file contents!
Username is lbranden
Trying to exec emacs
File Client: Next file to open
File name:Invalidating!
demoB
How(r/w)
w
File is not cached!
Invalidating!
Got file contents!
Username is lbranden
Trying to exec emacs
File Client: Next file to open
File name:Invalidating!
^C\lbranden@uw1-320-02:~/434/hw4$ ]
```

```
[lbranden@uw1-320-06:~/434/hw4$ java DFSClient uw1-320-06 uw1-320-01
File Client: Next file to open
File name:demoB
How(r/w)
w
File is not cached!
Got file contents!
Received write back request!
Username is lbranden
Trying to exec emacs
File Client: Next file to open
File name:Invalidating!
demoB
How(r/w)
w
File is not cached!
Invalidating!
Got file contents!
Received write back request!
Username is lbranden
Trying to exec emacs
File Client: Next file to open
File name:C\lbranden@uw1-320-06:~/434/hw4$ ]
```

```
[lbranden@uw1-320-07:~/434/hw4$ java DFSClient uw1-320-07 uw1-320-06
File Client: Next file to open
File name:demoB
How(r/w)
r
File is not cached!
Uploading file!
Got file contents!
Username is lbranden
Trying to exec emacs
File Client: Next file to open
File name:demoB
How(r/w)
w
File is not cached!
Got file contents!
Username is lbranden
Trying to exec emacs
File Client: Next file to open
File name:C\lbranden@uw1-320-07:~/434/hw4$ ]
```

Results where the modifications were not saved:

The screenshot shows a terminal window with four tabs open, each representing a different session or command-line interface.

- Session 1:** A continuous loop of the message "Still waiting." followed by a small square icon.
- Session 2:** The title bar reads "brandenlivermore — ssh lbranden@uw1-320-01.uwb.edu — 80x24". The content shows a file named "lbranden.txt" containing the text "123pqr".
- Session 3:** The title bar reads "brandenlivermore — ssh lbranden@uw1-320-06.uwb.edu — 80x24". It displays a message asking for assistance with CSS Technical Issues, with the email address "UWBSTITL@UW.EDU". Below this, it shows disk quota information for user "lbranden" on the "metis.uwb.edu:/usr/apps" filesystem.

Filesystem	space	quota	limit	grace	files	quota	limit	grace
metis.uwb.edu:/usr/apps	3380K	4000M	4098M		271	0	0	

- Session 4:** The title bar reads "brandenlivermore — ssh lbranden@uw1-320-07.uwb.edu — 80x24". It shows a message about assistance with CSS Technical Issues, the last login time (Tue Jun 7 12:37:26 2016), and disk quota information for user "lbranden" on the "metis.uwb.edu:/usr/apps" filesystem.

Filesystem	space	quota	limit	grace	files	quota	limit	grace
metis.uwb.edu:/usr/apps	3380K	4000M	4098M		271	0	0	

Alex s. Ducken, Branden Livermore: CSS 434 Homework 4 Distributed File System

The top-left terminal window shows a client session with the command `java DFSClient uw1-320-02 uw1-320-01`. It displays a continuous loop of "Still waiting." messages.

The top-right terminal window shows a storage node session with the command `java DFSStorageNode uw1-320-01`. It shows disk quota information and a file registry table.

The bottom-left terminal window shows a client session with the command `java DFSClient uw1-320-02 uw1-320-01`. It shows a file transfer process, including receiving a file named `demoB` and writing it back to disk.

The bottom-right terminal window shows a storage node session with the command `java DFSStorageNode uw1-320-01`. It shows disk quota information and a file registry table.

The top-left terminal window shows a client session with the command `java DFSClient uw1-320-02 uw1-320-01`. It displays a continuous loop of "Still waiting." messages.

The top-right terminal window shows a storage node session with the command `java DFSStorageNode uw1-320-01`. It shows disk quota information and a file registry table.

The bottom-left terminal window shows a client session with the command `java DFSClient uw1-320-02 uw1-320-01`. It shows a file transfer process, including receiving a file named `demoB` and writing it back to disk.

The bottom-right terminal window shows a storage node session with the command `java DFSStorageNode uw1-320-01`. It shows disk quota information and a file registry table.

Alex s. Ducken, Branden Livermore: CSS 434 Homework 4 Distributed File System

The image displays four terminal windows, each showing the interaction of a DFSClient with a distributed file system across four hosts: uw1-320-01, uw1-320-02, uw1-320-06, and uw1-320-07.

- Terminal 1 (Host 01):** Shows a user reading file "demoA" (containing "xyz?") and writing file "demoB" (containing "123").
- Terminal 2 (Host 02):** Shows a user reading file "demoB" (containing "123") and writing file "demoA" (containing "xyz?").
- Terminal 3 (Host 06):** Shows a user reading file "demoA" (containing "xyz?") and writing file "demoB" (containing "123").
- Terminal 4 (Host 07):** Shows a user reading file "demoB" (containing "123") and writing file "demoA" (containing "xyz?").

In all cases, the clients are communicating with each other to handle the file operations, demonstrating the distributed nature of the file system.

Discussion

Functional Improvements

Improving our program would start with exception handling, currently we handle exceptions only generally and usually terminate to process. A more robust design would handle individual exceptions and in general not terminate the process. We would factor out the enums into two different groups, one for the server and one for the client. Our ServerCachedFile is not properly factored and does not use getters and setters. Writing our comments in a standard for that can auto generate documentation would also be a considerable improvement.

Performance Improvements

Our program preforms well and completes the test. Making use of multiple threads for the server could allow for handling more clients. Where one thread handles downloads, and another handles uploads. Using semaphores and interrupts instead of spin locking the on ownership_change would likely yield considerable improvements.

Working with emacs proved difficult and while a solution was found, there is likely a more efficient way to handle spinning up emacs.