Shanghai Jiaotong University

Big Data Processing

Distributed File System Design

Author:
WANG Lei
118260910044
WU Tiance
117260910057
GU Yuanzhe
118260910034
LIU Qingmin
118260910037

Supervisor:
WU Chentao

November 30, 2019

1 Problem Description

- Design a Mini Distributed File System (Mini-DFS), which contains
  - A client
  - A name server
  - Four data servers
- Mini-DFS is running through a process. In this process, the name server and data servers are different threads
- Basic functions of Mini-DFS
  - Read/write a file
    - Upload a file: upload success and return the ID of the file
    - Read the location of a file based on the file ID and the offset
  - File striping
    - Slicing a file into several chunks
    - Each chunk is 2MB
    - Uniform distribution of these chunks among four data servers
  - Replication
    - Each chunk has three replications
    - Replicas are distributed in different data servers
- Name Server
  - List the relationships between file and chunks
  - List the relationships between replicas and data servers
  - Data server management
- Data Server
  - Read/Write a local chunk
  - Write a chunk via a local directory path
- Client

    - Provide read/write interfaces of a file

- Mini-DFS can show
    - Read a file (more than 7MB)
        - Via input the file and directory
    - Write a file (more than 3MB)
        - Each data server should contain appropriate number of chunks
        - Using MD5 checksum for a chunk in different data servers, the results should be the same
        - Check a file in (or not in) Mini-DFS via inputting a given directory
        - By inputting a file and a random offset, output the content

2 Structure and Method

In our project, we design a mini distributed file system. We can store files into this MiniDFS. We totally have 5 options: put, read, fetch, ls, and quit.

The meaning of each option is as following:

- put: Write a local file into MiniDFS.
- fetch: Download a file in MiniDFS to a local path.
- read: Read a special part of a file in MiniDFS.
- ls: Get the file list in MiniDFS.
- quit: Exit the system.

There are two python files:

- **const_share.py** It defines several meta-parameters.
- **main.py** It defines classes and functions.

We will introduce them respectively.

- **const_share.py**

  In this file we define following meta-parameters:

  - BLOCK_SIZE: It is the size of each chunk and it is 2MB.
  - NUM_DATA_SERVER: It is the number of data servers and it is 4.
  - NUM_REPLICATION: It is the number of replication of each chunk and it is 3.
  - BLOCK_PATTERN: It is the name format of each chunk. For example, the second chunk of file 3 will be named as "3-part-2".

- **main.py**

  In this file we mainly have two classes: NameNode and DataNode.

  - NameNode In this class, we define multiple functions:
    - *__init__*: Define the map projection between file id and chunks, map projection between file id and file name, map projection between chunks and data servers, last file id, and last data server id that has been used.
    - *run*: Override the *run* function of threading. It receives commands and deal with 4 commands: put, fetch, read, and ls.
    - *load_meta*: Load name meta data, including "id_block_map", "id_len_map", "block_server_map", "last_file_id", and "last_data_server_id".
    - *update_meta*: Update above name meta data each time after "put" action.
    - *list_dfs_files*: List out the files stored in our MiniDFS.
    - *generate_split*: Split the input file into several chunks and distribute them into different blocks.
    - *assign_read_work*: If the command is "read", this function will be called. It gives the "read" command to the specific data nodes.
    - *assign_fetch_work*: If the command is "fetch", this function will be called. It gives the "fetch" command to the specific data nodes.
  - DataNode In this class, we define multiple functions:

- *__init__*: Define the server id of corresponding data server.
- *run*: Override the *run* function of threading. It only deals with two specific missions: "put" and "read". For each command, it calls corresponding function in this class.
- *save_file*: Data node save files according to the distribution outcomes of Name node.
- *read_file*: Data node read file according to the offset and counting number.

3 Usage

There are 2 files in this project. And there are 5 options for users to use MiniDFS.

- put: "put LOCAL_PATH".

  LOCAL_PATH: the local path of target file that we want to upload into MiniDFS.

- fetch: "fetch FILE_ID SAVE_PATH".

  FILE_ID: the id of target file in MiniDFS. It can be checked by option "ls".

  SAVE_PATH: the local path where we want to store this file.

- read: "read FILE_ID OFFSET COUNT".

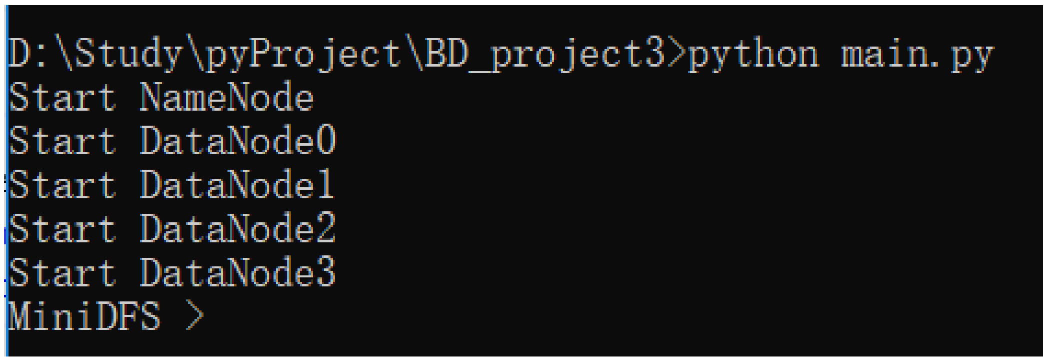
  FILE_ID: the id of target file in MiniDFS. It can be checked by option "ls".

  OFFSET: the offset from the very beginning of a file to the start position for reading.

  COUNT: the length for reading.

- ls: "ls".

- quit: "quit".



```
D:\Study\pyProject\BD_project3>python main.py
Start NameNode
Start DataNode0
Start DataNode1
Start DataNode2
Start DataNode3
MiniDFS >
```

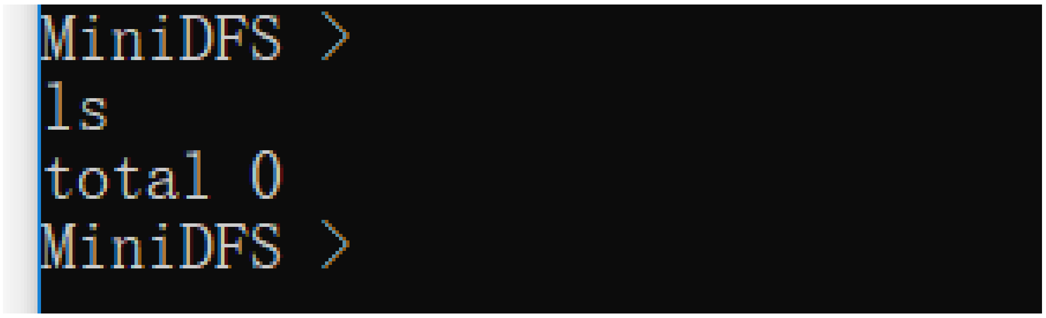
Figure 1: Start MiniDFS

4 Experiment and Result

We use following command to start our MiniDFS:

As we can see in figure 1, we start 1 name server and 4 data servers.

If we use ”ls” command to query files in MiniDFS, we will get the result shown in figure 2



```
MiniDFS >
ls
total 0
MiniDFS >
```

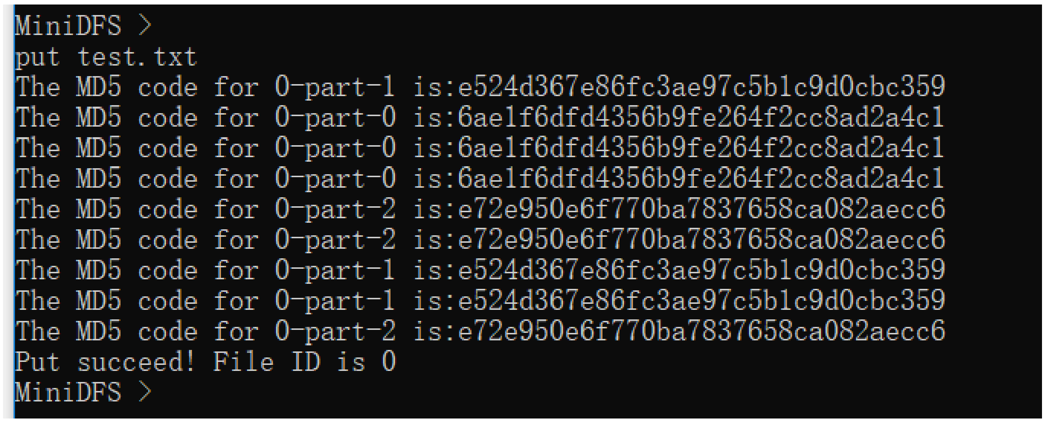
Figure 2: ”ls” after start MiniDFS

As we can see, the total file number is 0. There is no file in our file system.

Then we put a file into our file system: ”test.txt”. The result is shown in figure 3.

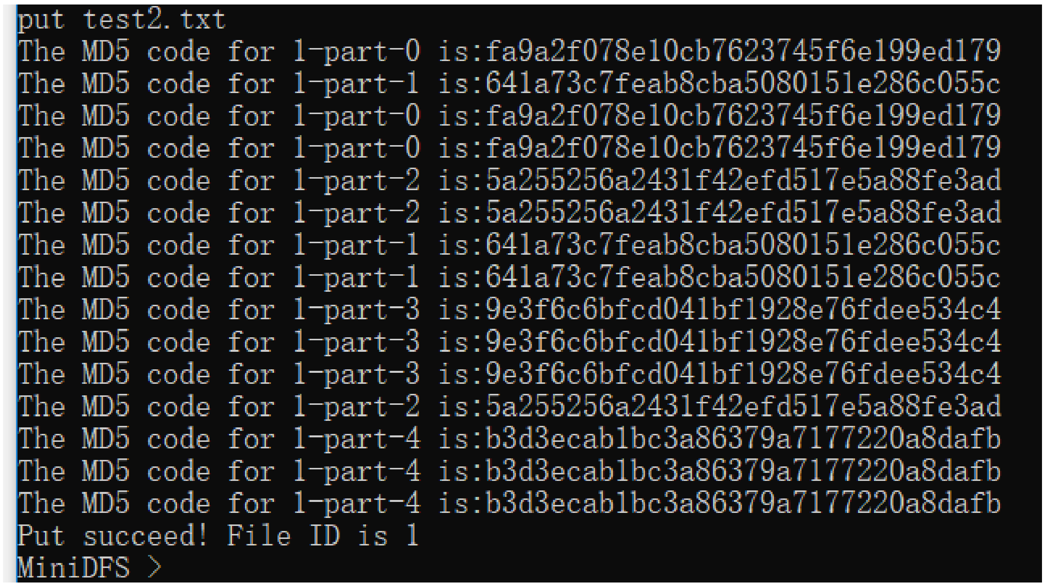
And then put another file into file system: ”test2.txt”.

The file ”test.txt” is splited into 3 chunks and each chunk is replicated 3 times. Similarly, the file ”test2.txt” is splited into 5 chunks. That’s because in our settings, the maximum size of each chunk is 2MB. We alse use MD5



```
MiniDFS >
put test.txt
The MD5 code for 0-part-1 is:e524d367e86fc3ae97c5b1c9d0cbc359
The MD5 code for 0-part-0 is:6ae1f6dfd4356b9fe264f2cc8ad2a4c1
The MD5 code for 0-part-0 is:6ae1f6dfd4356b9fe264f2cc8ad2a4c1
The MD5 code for 0-part-0 is:6ae1f6dfd4356b9fe264f2cc8ad2a4c1
The MD5 code for 0-part-2 is:e72e950e6f770ba7837658ca082aecc6
The MD5 code for 0-part-2 is:e72e950e6f770ba7837658ca082aecc6
The MD5 code for 0-part-1 is:e524d367e86fc3ae97c5b1c9d0cbc359
The MD5 code for 0-part-1 is:e524d367e86fc3ae97c5b1c9d0cbc359
The MD5 code for 0-part-2 is:e72e950e6f770ba7837658ca082aecc6
Put succeed! File ID is 0
MiniDFS >
```

Figure 3: Put ”test.txt” into MiniDFS

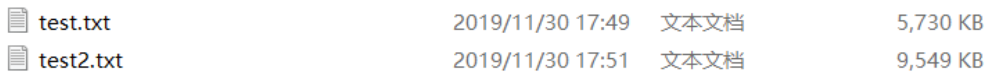


```
put test2.txt
The MD5 code for 1-part-0 is:fa9a2f078e10cb7623745f6e199ed179
The MD5 code for 1-part-1 is:641a73c7feab8cba5080151e286c055c
The MD5 code for 1-part-0 is:fa9a2f078e10cb7623745f6e199ed179
The MD5 code for 1-part-0 is:fa9a2f078e10cb7623745f6e199ed179
The MD5 code for 1-part-2 is:5a255256a2431f42efd517e5a88fe3ad
The MD5 code for 1-part-2 is:5a255256a2431f42efd517e5a88fe3ad
The MD5 code for 1-part-1 is:641a73c7feab8cba5080151e286c055c
The MD5 code for 1-part-1 is:641a73c7feab8cba5080151e286c055c
The MD5 code for 1-part-3 is:9e3f6c6bfcd041bf1928e76fdee534c4
The MD5 code for 1-part-3 is:9e3f6c6bfcd041bf1928e76fdee534c4
The MD5 code for 1-part-3 is:9e3f6c6bfcd041bf1928e76fdee534c4
The MD5 code for 1-part-2 is:5a255256a2431f42efd517e5a88fe3ad
The MD5 code for 1-part-4 is:b3d3ecab1bc3a86379a7177220a8dafb
The MD5 code for 1-part-4 is:b3d3ecab1bc3a86379a7177220a8dafb
The MD5 code for 1-part-4 is:b3d3ecab1bc3a86379a7177220a8dafb
Put succeed! File ID is 1
MiniDFS >
```

Figure 4: Put ”test2.txt” into MiniDFS

code for checking each chunk. As shown in figure 3 and figure4, the MD5 code for the same chunk is the same.

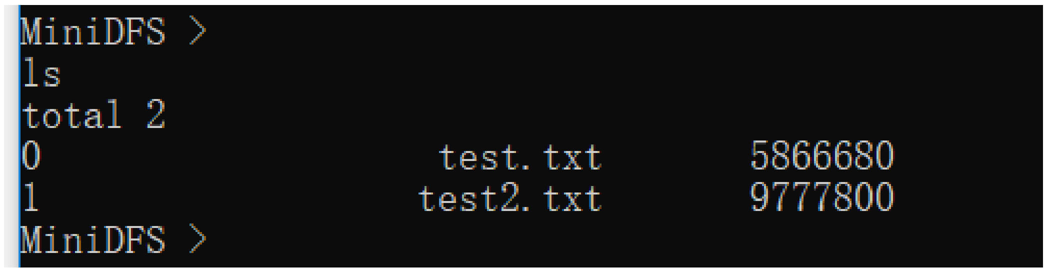
And the sizes of "test.txt" and "test2.txt" is shown in figure 5



test.txt	2019/11/30 17:49	文本文档	5,730 KB
test2.txt	2019/11/30 17:51	文本文档	9,549 KB

Figure 5: Size of "test.txt" and "test2.txt"

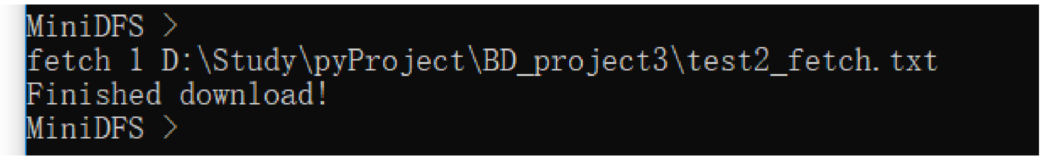
After uploading two files, let's do "ls" again and we will get following result in figure 6:



```
MiniDFS >
ls
total 2
0                    test.txt         5866680
1                   test2.txt         9777800
MiniDFS >
```

Figure 6: Do "ls" after uploading 2 files

Now we can try "fetch".



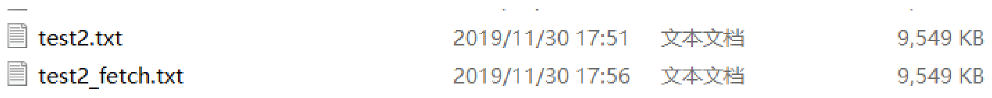
```
MiniDFS >
fetch 1 D:\Study\pyProject\BD_project3\test2_fetch.txt
Finished download!
MiniDFS >
```

Figure 7: Fetch "test2.txt"

After fetch, we get a new local file "test2_fetch.txt". It is the same as file "test2.txt" and their size are also the same as shown in figure 8.

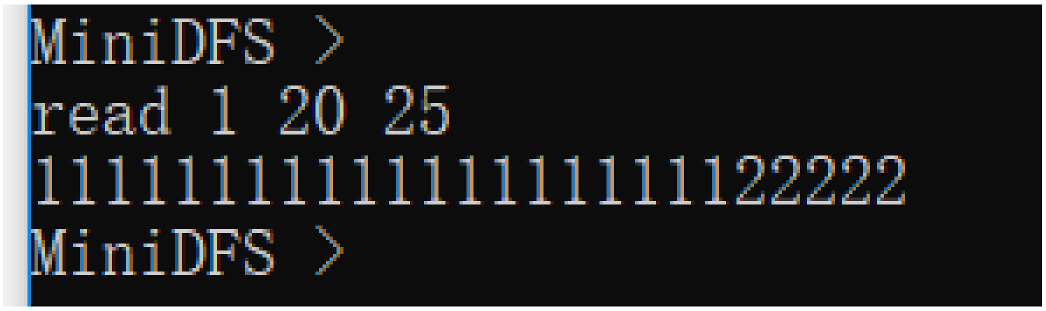
What's more, we can read files on MiniDFS as shown in figure 9 and figure 10 repectively:

At last, we can exit our MiniDFS through "quit" as shown in figure 11:



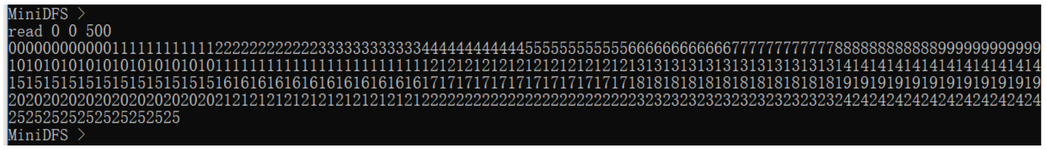
```
test2.txt          2019/11/30 17:51   文本文档   9,549 KB
test2_fetch.txt    2019/11/30 17:56   文本文档   9,549 KB
```

Figure 8: The size of two files are the same



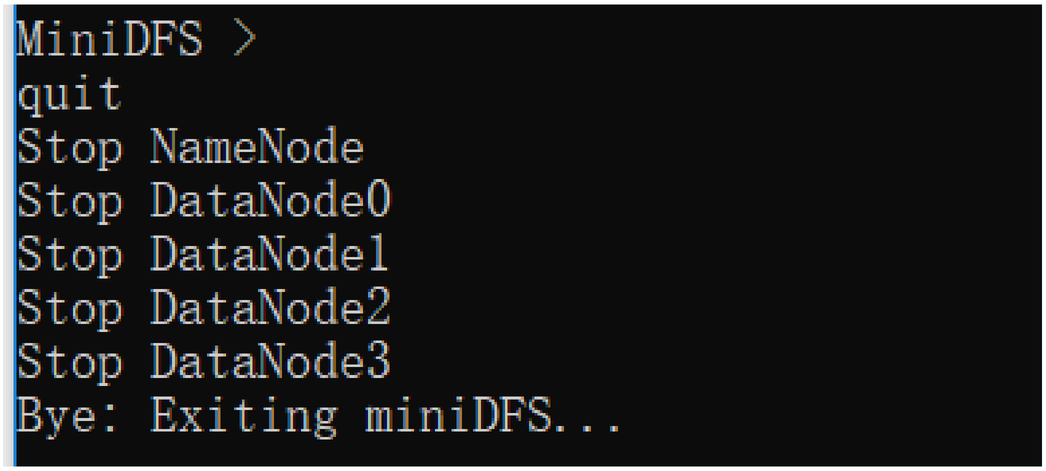
```
MiniDFS >
read 1 20 25
1111111111111111111122222
MiniDFS >
```

Figure 9: Read from files in MiniDFS



```
MiniDFS >
read 0 0 500
000000000000111111111111222222222222333333333333444444444444555555555555666666666666777777777777888888888888999999999999
101010101010101010101010111111111111111111111111121212121212121212121212131313131313131313131313141414141414141414141414
151515151515151515151515161616161616161616161616171717171717171717171717181818181818181818181818191919191919191919191919
202020202020202020202020212121212121212121212121222222222222222222222222232323232323232323232323242424242424242424242424
25252525252525252525
MiniDFS >
```

Figure 10: Read from files in MiniDFS



```
MiniDFS >
quit
Stop NameNode
Stop DataNode0
Stop DataNode1
Stop DataNode2
Stop DataNode3
Bye: Exiting miniDFS...
```

Figure 11: Exit file system

5 Future and Development

In our project, we design a distributed file system. We can do ”put”, ”fetch”, ”read”, ”ls”, and ”quit” commands on it. And each file has been splited into chunks that no more than 2MB and each chunk is replicated 3 times in different data servers. What’s more, we can use MD5 code to check chunk. In the future, we will try to realize the recovery function according to replications in different data servers.