Shanghai Jiaotong University

Big Data Processing Technology

Project 3: Mini DFS

Author:
FEI Yixiao
118260910031
LI Yanhao
118260910036
LUO Dihao
118260910039
SHEN Shengyang
118260910042
YAN Shuhan
118260910050

Supervisor:
Chentao WU
Xin XIE

October 21, 2019

上海交大-巴黎高科卓越工程师学院
Ecole d'Ingénieurs SJTU-ParisTech

1 Introduction

Our distributed metadata management is based on Java and is composed with following files:

- Client
  - Client.java : the client interface where commands are accepted
- Cluster
  - Cluster.java : we use this class to group data servers and distribute data blocks
  - DataServer.java : distributed data servers which stores eventually data blocks
  - MetaData.java : store meta data information
  - DirectoryTree.java : directory tree information
  - TreeNode.java : tree node information
- Tools
  - Request.java : class to define the message
  - Response.java : class to define the message

1.1 Usage

Users can access distributed metadata management by directly running *Client.class*. Before that, we need to also launch the management services including the *Cluster.class* and at least one *DataServer.class*.

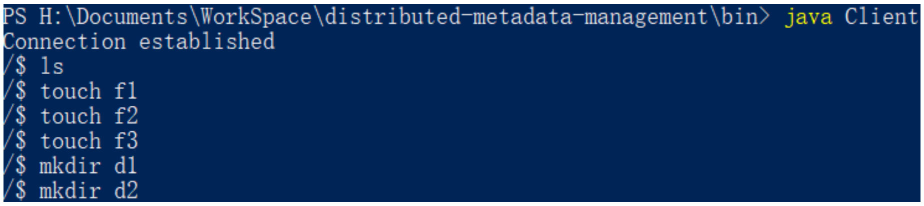
- cd dir_name : go to the directory
- ls : show list of files and directories
- touch file_name : create a file with with the name of file_name
- mkdir dir_name : create a directory with the name of dir_name
- tree : show the tree structure of the files and directories
- fulltree : show the tree structure of the files and directories with full paths
- chkdist : check distributed metadata in all data servers
- stat file_name : check the permissions of a file
- chmod file_name permissions: modify the permissions of a file

- exit : exit distributed metadata management

When a client sends a command to the interface *Client*, *Client* shares this command with *Cluster* and *DataServer* through *Request* and *Response*. *Cluster* will choose either an appropriate *DataServer* to store the data block or extract the information it needs and gives back the information to the *Client*.

2 Example

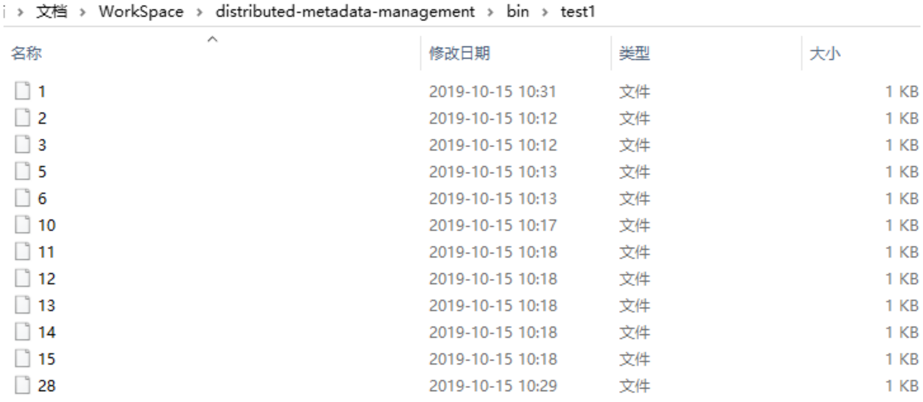
As you may discover during the experiment, our distributed metadata management accepts only several commands. When the command line does not understand the command, it gives the list of the accepted commands. In addition, each command has its own format of the input. The input number of arguments might be different, so when the format is not correct, it also gives the warning message to help the client enter the correct commands.
Now first, we can create several files in our distributed metadata management using *touch* and then we can also create some directories using *mkdir*.



```
PS H:\Documents\WorkSpace\distributed-metadata-management\bin> java Client
Connection established
/$ ls
/$ touch f1
/$ touch f2
/$ touch f3
/$ mkdir d1
/$ mkdir d2
```

Figure 1: Usage: touch & mkdir

In directory *bin/test1* and also in *bin/test2*, we can see that it creates some blocks. These two directories are generated by the *DataServer* whose names are respectively *test1* and *test2*.

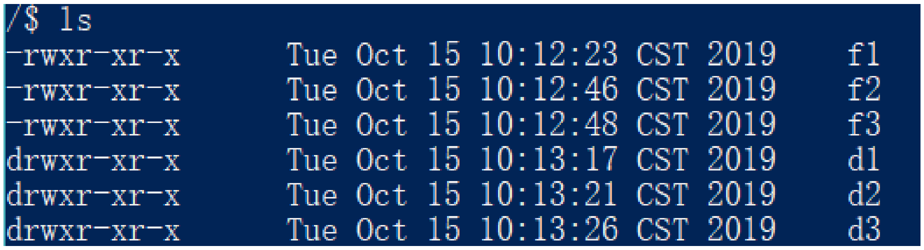


i › 文档 › WorkSpace › distributed-metadata-management › bin › test1

名称	修改日期	类型	大小
1	2019-10-15 10:31	文件	1 KB
2	2019-10-15 10:12	文件	1 KB
3	2019-10-15 10:12	文件	1 KB
5	2019-10-15 10:13	文件	1 KB
6	2019-10-15 10:13	文件	1 KB
10	2019-10-15 10:17	文件	1 KB
11	2019-10-15 10:18	文件	1 KB
12	2019-10-15 10:18	文件	1 KB
13	2019-10-15 10:18	文件	1 KB
14	2019-10-15 10:18	文件	1 KB
15	2019-10-15 10:18	文件	1 KB
28	2019-10-15 10:29	文件	1 KB

Figure 2: Usage: put & mkdir

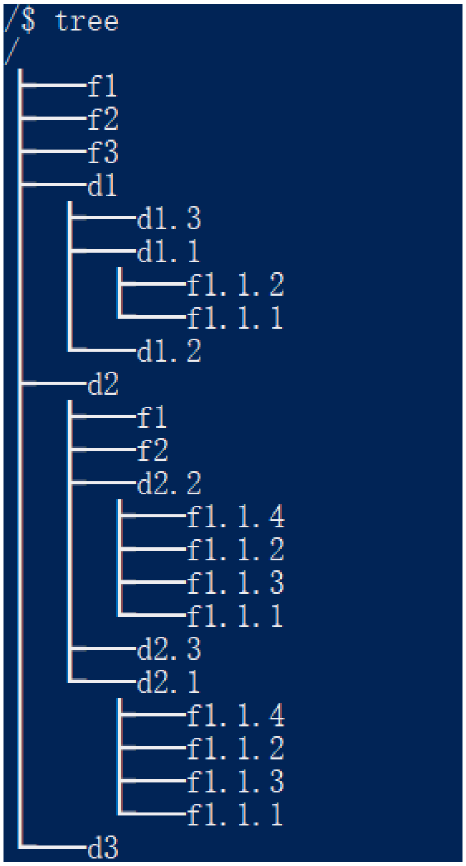
Then, we can try to list what we have created for now. As you can see in the presented picture, we have created three files and three directories with the timestamp respectively.



```
/$ ls
-rwxr-xr-x      Tue Oct 15 10:12:23 CST 2019    f1
-rwxr-xr-x      Tue Oct 15 10:12:46 CST 2019    f2
-rwxr-xr-x      Tue Oct 15 10:12:48 CST 2019    f3
drwxr-xr-x      Tue Oct 15 10:13:17 CST 2019    d1
drwxr-xr-x      Tue Oct 15 10:13:21 CST 2019    d2
drwxr-xr-x      Tue Oct 15 10:13:26 CST 2019    d3
```

Figure 3: Usage: ls

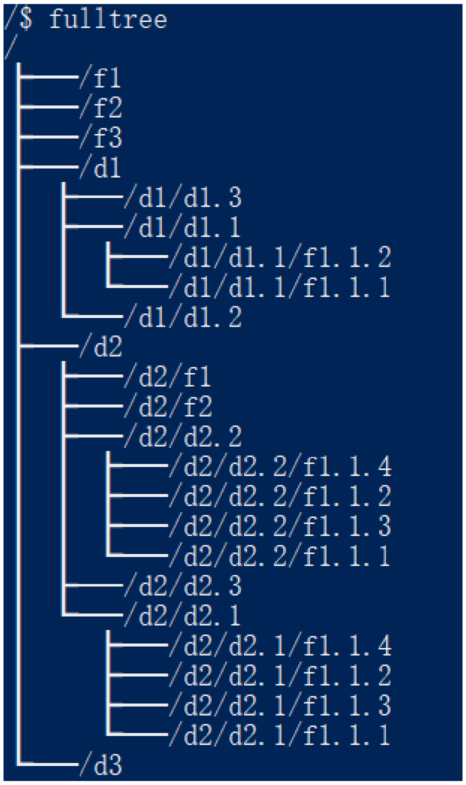
Now let us create more files and directories to complicate our metadata management and we will use *tree* to display the structure.



```
/$ tree
/
├──f1
├──f2
├──f3
├──d1
│  ├──d1.3
│  ├──d1.1
│  │  ├──f1.1.2
│  │  └──f1.1.1
│  └──d1.2
├──d2
│  ├──f1
│  ├──f2
│  ├──d2.2
│  │  ├──f1.1.4
│  │  ├──f1.1.2
│  │  ├──f1.1.3
│  │  └──f1.1.1
│  ├──d2.3
│  └──d2.1
│     ├──f1.1.4
│     ├──f1.1.2
│     ├──f1.1.3
│     └──f1.1.1
└──d3
```

Figure 4: Usage: tree

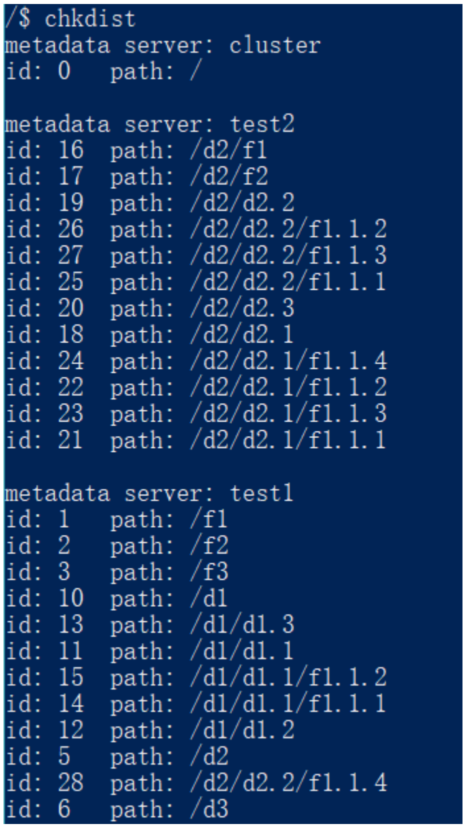
And *fulltree* will display the structure with the complete paths.



```
/$ fulltree
/
├──/f1
├──/f2
├──/f3
├──/d1
│   ├──/d1/d1.3
│   ├──/d1/d1.1
│   │   ├──/d1/d1.1/f1.1.2
│   │   └──/d1/d1.1/f1.1.1
│   └──/d1/d1.2
├──/d2
│   ├──/d2/f1
│   ├──/d2/f2
│   ├──/d2/d2.2
│   │   ├──/d2/d2.2/f1.1.4
│   │   ├──/d2/d2.2/f1.1.2
│   │   ├──/d2/d2.2/f1.1.3
│   │   └──/d2/d2.2/f1.1.1
│   ├──/d2/d2.3
│   └──/d2/d2.1
│       ├──/d2/d2.1/f1.1.4
│       ├──/d2/d2.1/f1.1.2
│       ├──/d2/d2.1/f1.1.3
│       └──/d2/d2.1/f1.1.1
└──/d3
```

Figure 5: Usage: fulltree

Since we have a relatively complicated file structure, now we can check where these data nodes are stored in our distributed metadata management using *chkdist*.



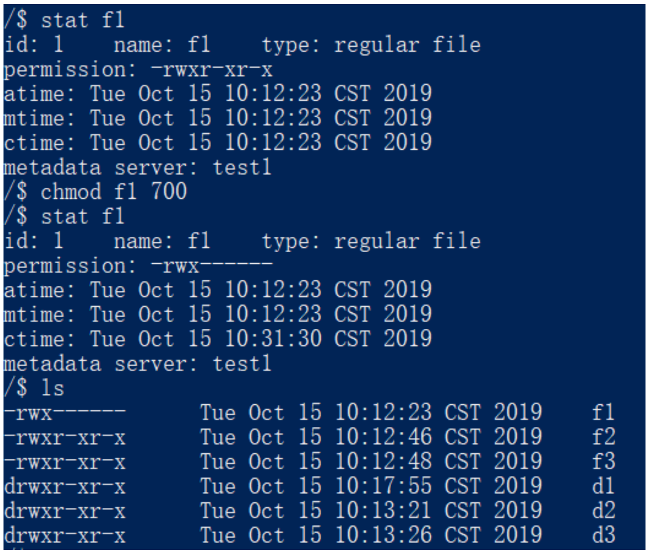
```
/$ chkdist
metadata server: cluster
id: 0   path: /

metadata server: test2
id: 16  path: /d2/f1
id: 17  path: /d2/f2
id: 19  path: /d2/d2.2
id: 26  path: /d2/d2.2/f1.1.2
id: 27  path: /d2/d2.2/f1.1.3
id: 25  path: /d2/d2.2/f1.1.1
id: 20  path: /d2/d2.3
id: 18  path: /d2/d2.1
id: 24  path: /d2/d2.1/f1.1.4
id: 22  path: /d2/d2.1/f1.1.2
id: 23  path: /d2/d2.1/f1.1.3
id: 21  path: /d2/d2.1/f1.1.1

metadata server: test1
id: 1   path: /f1
id: 2   path: /f2
id: 3   path: /f3
id: 10  path: /d1
id: 13  path: /d1/d1.3
id: 11  path: /d1/d1.1
id: 15  path: /d1/d1.1/f1.1.2
id: 14  path: /d1/d1.1/f1.1.1
id: 12  path: /d1/d1.2
id: 5   path: /d2
id: 28  path: /d2/d2.2/f1.1.4
id: 6   path: /d3
```

Figure 6: Usage: chkdist

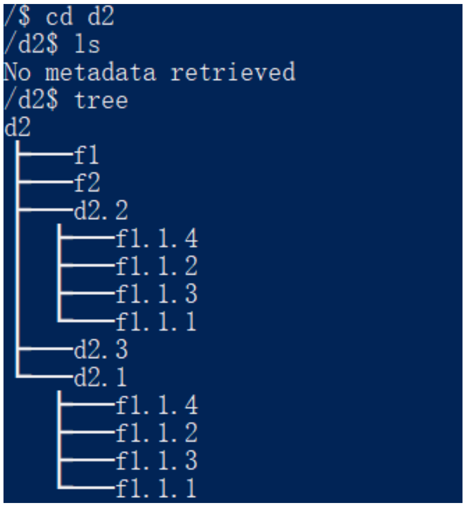
We can of course change the permissions and check whether our modification works. To do this, we first check the permissions with *stat*. Then we change the permissions with *chmod*. and finally, we check once again the permissions.



```
/$ stat f1
id: 1    name: f1    type: regular file
permission: -rwxr-xr-x
atime: Tue Oct 15 10:12:23 CST 2019
mtime: Tue Oct 15 10:12:23 CST 2019
ctime: Tue Oct 15 10:12:23 CST 2019
metadata server: test1
/$ chmod f1 700
/$ stat f1
id: 1    name: f1    type: regular file
permission: -rwx------
atime: Tue Oct 15 10:12:23 CST 2019
mtime: Tue Oct 15 10:12:23 CST 2019
ctime: Tue Oct 15 10:31:30 CST 2019
metadata server: test1
/$ ls
-rwx------      Tue Oct 15 10:12:23 CST 2019    f1
-rwxr-xr-x      Tue Oct 15 10:12:46 CST 2019    f2
-rwxr-xr-x      Tue Oct 15 10:12:48 CST 2019    f3
drwxr-xr-x      Tue Oct 15 10:17:55 CST 2019    d1
drwxr-xr-x      Tue Oct 15 10:13:21 CST 2019    d2
drwxr-xr-x      Tue Oct 15 10:13:26 CST 2019    d3
```

Figure 7: Usage: stat & chmod

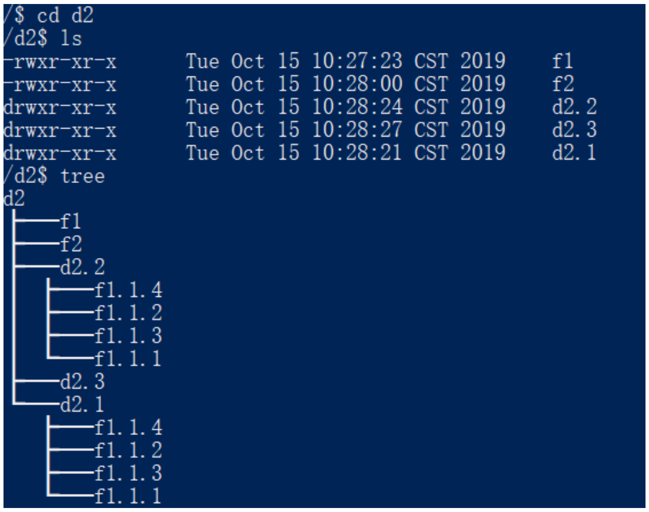
Now we wonder what if one of our *DataServer* goes offline. Here we did an experiment by disabling *DataServer test2*. And we knew in advance that all the data blocks related to files in *d2* are stored exactly in *DataServer test2*. Naturally, we will go into *d2* to see what would happen. As you might observe in the picture, this time when we demand listing the files in *d2*, it could not extract information though the tree information is still stored in the *TreeNode*.



```
/$ cd d2
/d2$ ls
No metadata retrieved
/d2$ tree
d2
├────f1
├────f2
├────d2.2
│    ├────f1.1.4
│    ├────f1.1.2
│    ├────f1.1.3
│    └────f1.1.1
├────d2.3
└────d2.1
     ├────f1.1.4
     ├────f1.1.2
     ├────f1.1.3
     └────f1.1.1
```

Figure 8: When test2 is offline

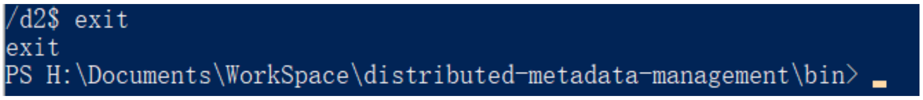
Now we can make *DataServer test2* go online once again. This time when we demand listing the files in *d2*, it could extract information, which is exactly what we want.



```
/$ cd d2
/d2$ ls
-rwxr-xr-x      Tue Oct 15 10:27:23 CST 2019    f1
-rwxr-xr-x      Tue Oct 15 10:28:00 CST 2019    f2
drwxr-xr-x      Tue Oct 15 10:28:24 CST 2019    d2.2
drwxr-xr-x      Tue Oct 15 10:28:27 CST 2019    d2.3
drwxr-xr-x      Tue Oct 15 10:28:21 CST 2019    d2.1
/d2$ tree
d2
 ├───f1
 ├───f2
 ├───d2.2
 │   ├───f1.1.4
 │   ├───f1.1.2
 │   ├───f1.1.3
 │   └───f1.1.1
 ├───d2.3
 └───d2.1
     ├───f1.1.4
     ├───f1.1.2
     ├───f1.1.3
     └───f1.1.1
```

Figure 9: When test2 is re-alive

Finally, we can safely exit distributed metadata management and this completes our examples of the experiment.



```
/d2$ exit
exit
PS H:\Documents\WorkSpace\distributed-metadata-management\bin>
```

Figure 10: Usage: quit