EEL 5737: The Implementation of Distributed File System

Han Wei
Department of Electrical & Computer Engineering
College of Engineering
University of Florida
Gainesville, Florida 32611

Cheng Li
Department of Electrical & Computer Engineering
College of Engineering
University of Florida
Gainesville, Florida 32611

***Abstract*—In previous assignments, we established a distributed file system. We stored the file data and file metadata into two servers. In this project, we will store the file data into several servers. The new file system can tolerate the server crashes and deal with data corruption. The user can write into the file, even if one file data server is not available.**

I. Introduction

II. Design and Implementation

A. Basic File Operations

The basic operations of file directory and content are similar to the original functions in previous assignments. However, we need to modify them to meet the new requirements.

B. Redundant Storage

We need to store the data on different servers. We use stateArray to record the reading status of each block. We firstly initialize them as 0. If the data can be read successfully, it is marked 1 and 0, otherwise. The same mechanism goes with the write. The old version data is modified and the status needs to be updated. If the write failed in the servers, the block that is written in will be popped.

```
def distributedWrite(self,
    path,data,offset,fh):
   metafiles = self.getmeta(path)

   startBlk = int(offset/BLOCKSIZE) #
       offset determine the first blockID
       to be read
   startOffset = offset%BLOCKSIZE

   lastBlk =
       int(math.ceil((offset+len(data))/BLOCKSIZE))
   endOffset = (offset+len(data))%BLOCKSIZE

   numblk = lastBlk - startBlk
   start_port = metafiles['startPortID']
   start_port = (start_port +
       startBlk)%self.N
   port = start_port
   state = [0] * (numblk) #initialized as
       0. 1 --- success -1 -----fail
   index_of_block = 0 #this index is to
       record the index of block.
   version = metafiles['version']
   newVersion = version
   for i in range(startBlk, lastBlk):
      if i in range(len(version)):
         newVersion[i] = version[i] + 1
      else:
         newVersion.append(0)
   checkpoint = 0 # #counter tp recort how
       much of the data have been written
   # The following part is the same logic
       as distributedRead
   #1. the old version is modified before
       and need to update
   #2. this is the very first new version
   for i in range(startBlk,lastBlk):
      if i!=startBlk and i!=lastBlk-1:
         try:
            self.dataPut(port,path,i,data[checkpoint:
                newVersion[i])
            checkpoint =
                checkpoint+BLOCKSIZE
            state[index_of_block] == 1
         except OSError, e:
            if e.args[0] ==
                errno.ECONNREFUSED:
               state[index_of_block] = -1
               print("NO connection!!!")
               break
      elif i==startBlk:
         try:
            temp = ’’
            if newVersion[i]!=0:
               temp =
                   self.dataGet(port,path,i,version[i
               temp = temp[:startOffset]
            temp = temp +
                data[:BLOCKSIZE-startOffset]
            checkpoint =
                BLOCKSIZE-startOffset
            self.dataPut(port,path,i,temp,
                newVersion[i])
            state[index_of_block] == 1
         except OSError, e:
            if e.args[0] ==
                errno.ECONNREFUSED:
               state[index_of_block] = -1
               print("NO connection")
               break
      elif i==lastBlk-1: # for the last
          block
         try:
```

```
        temp = ''
        if newVersion[i]!=0:
          temp =
             self.dataGet(port,path,i,version[i])
          temp = temp[endOffset:]
        temp = data[checkpoint:]+temp
        checkpoint = checkpoint +
           len(data[checkpoint:])
        self.dataPut(port,path,i,temp,
           newVersion[i])
        state[index_of_block] == 1
      except OSError, e:
        if e.args[0] ==
           errno.ECONNREFUSED:
          state[index_of_block] = -1
          print("No connection..")
          break
    #change the server after writing one
       block.
    port = (port+1)%self.N
    index_of_block += 1

  #when the distributedwrite failed in
     some of the servers, pop the block
     just wrote in.
  if -1 in state:
    port = metafiles['startPortID']
    for i in range(startBlk,lastBlk):
      if state[i]==1:
        self.dataPop(port,path,i,newVersion[i])
      port = (port+1)%self.N
    return False
  else:
    # update the version list
    metafiles['version'] = newVersion
    self.putmeta(path,metafiles)
    return True


def distributedRead(self, path, size,
   offset, fh):
  metafiles = self.getmeta(path)
  content = ''
  # There are two corner cases caused by
     offset and size.
  if offset >= metafiles['st_size']:
    return content
  if (offset+size) > metafiles['st_size']:
    size = metafiles['st_size']-offset

  startBlk = int(offset/BLOCKSIZE) #
     offset determine the first blockID
     to be read
  startOffset = offset%BLOCKSIZE

  # decides which server to read from
     first.
  start_port = metafiles['startPortID']
  start_port = (start_port +
     startBlk)%self.N

  lastBlk =
     int(math.ceil((offset+size)/BLOCKSIZE))
  endOffset = (offset+size)%BLOCKSIZE

  port = start_port # port, assistant
     varieble to traverse the dataservers
```

```
  state = [0] * (lastBlk-startBlk)
  #need stateArray to record the reading
     status of each block. initialized
     as 0. If success, it becomes 1
  # and it becomes 0 if fails.
  #reading each blk from different servers
  #since the reading of starting and
     ending block is affected by the
     offset, I operate them separately.
  #if it is between, I don‘t worry about
     offset ,just read them all.
  for i in range(startBlk, lastBlk):
    if i != startBlk and i != lastBlk-1:
      content = content +
         self.dataGet(port,path,i,metafiles['vers
    elif i == startBlk:
      content =
         self.dataGet(port,path,i,metafiles['vers
      content = content[startOffset:]
    else:
      rv =
         self.dataGet(port,path,i,metafiles['vers
      content = content +
         rv[:(size-len(content))]
    port = (port+1)%self.N

  return content
```

C. Crash Toleration

The file system can recover and resume the serving data from local disk when processes crash and restart. When the data on the server is lost, the file system can recover the data from the replicas on its adjacent servers.

Each server has its own data and the copy of the last server's own data. When the server restarts, the file system will retrieve the history data. If the data on the block is empty, it will get the replica from the next server. If there is still no data on the next server, it will get the replica from other servers.

In case of no history records or hard disk failure, the file system will recover previous data and the copies of the other servers kept by dataserver.

```
def __init__(self,*argv):
 # python dataserver.py 0 3333 4444 5555
    6666
 #The first argument tells which one is the
    current server. e.g., 0 means port
    3333 is the port
 #of current server.
 #In this project, there are 5 arguments,
    the first one is index and the
    following four are ports of the
 #data servers.
 current_server = argv[0]
 N = len(argv)-1
 Ports_dataServ = argv[1:]
 port = str(Ports_dataServ[current_server])
 print ('dataserver',port,'is online...')
 # Retrieve history data
 # self.data is a dict of dicts that has
    REDUNDANCY ports:data pairs
 # Actually, each server has its own data
    and the copy of last server's own data
```

```
# {}, the filename is dataserver_port

self.data =
    shelve.open('dataserver_'+port,
    writeback=True)

# connect to the nextServer for recovery
nextPort =
    str(Ports_dataServ[(current_server+1)%N])
self.nextServer =
    xmlrpclib.ServerProxy('http://localhost:'
    + nextPort)

self.ports = {}
self.servers = {}
for i in range(REDUNDANCY):
  self.ports[i] =
      str(Ports_dataServ[(current_server-i)%N])
      # ports[0] is the port of current
      dataserver
  if i!= 0:
    self.servers[self.ports[i]] =
        xmlrpclib.ServerProxy('http://localhost:'
        + self.ports[i])
#connect to the all the other dataservers
#check whether the data dic is empty. If
    it is empty, initialize them. get the
    replica from nextserver.
#Otherwise, get the replica from other
    servers.
if self.data!={}:
  print("Loaded previous data from server
      backup file")
else:
  for i in range(REDUNDANCY):
    self.data[self.ports[i]] = {}
  # 2 cases:
  # 1)No history records:
  # 2)Hard disk fails
  # Anyway, recover previous data and the
      copies of other servers kept by this
      dataserver
  for i in self.ports:
    # Communicate with the next server to
        retrieve its own data
    # 1)recover self.data[port]: talk with
        next server
    p = self.ports[i]
    if p == port:
      # Assume only one server crash at a
          time.
      # No need to retry if failed to
          connect with server
      try:
        replica =
            self.nextServer.getReplica(Binary(port))
        self.data[p] =
            pickle.loads(replica.data)
        print ('Successfully loaded previous
            data from dataserver:', nextPort)
      except socket.error:
        print ('Recovering previous data is
            passed')
        pass # Based on the assumption, this
            only happens when starting to
            run all dataservers at the
            beginning.In this case, skip
            getReplica by ingore the socket
            error
    else:
      # 2)recover other copies from the
          original dataservers
      try:
        replica =
            self.servers[p].getReplica(Binary(p))
        self.data[p] =
            pickle.loads(replica.data)
        print ("Successfully loaded previous
            copy from dataserver:", p)
      except socket.error:
        print ("Recovering previous copy is
            passed")
        pass
```

D. Corruption Protection

For test, a corrupt function is made. The function will corrupt at least one byte of any blocks of the latest file. A block and some bytes are chosen randomly. The chosen bytes will be messed intentionally.

```
def corrupt(self,path): # corrupt the latest
    data
# This function will corrupt at least one
    byte of any blocks of the file
  rv = False
  p = []
  for i in self.ports:
    if path in self.data[self.ports[i]]:
      p.append(i)
  if len(p)>0:
    s = np.random.randint(0,len(p))
    s = p[s]

    # randomly pick up a block
    IDlist =
        self.data[self.ports[s]][path].keys()
    ID = np.random.randint(0,len(IDlist))
    ID = IDlist[ID]
    # randomly pick up a byte
    version =
        len(self.data[self.ports[s]][path][ID])-1
    mes =
        self.data[self.ports[s]][path][ID][version]
    byte = np.random.randint(0,len(mes)-1)
    i = 1
    while(mes[byte]==str(i)):
      i = i+1
    mes = mes[:byte] + str(i) + mes[byte+1:]
    self.data[self.ports[s]][path][ID][version]
        = mes
    rv = True
    print('Information:')
    print ('Corrupted file:', path)
    print ('Corrupted blks: ', ID)
  else:
    print("Path is unfound.")
  return rv
```

For corruption protection, the checksum is needed. We use cyclic redundancy check (CRC), which is widely used in all types of communication. It can detect most errors and very fast to compute [1].

```
def checksum(self,data):
    crc = binascii.crc32(data) & 0xffffffff
    return "{0:x}".format(crc).rjust(8,'0') #
        take it as a 8-byte string

crc1 = message[len(message)-8:]
crc = self.checksum(data[i])
if(crc==crc1):
    server_states[i+REDUNDANCY] = 1
```

E. Dealing with Unavailable Server

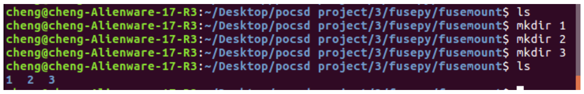
The status array can record the status of write. If the write fails because of server failure or so, the block will pop.

```
#when the distributedwrite failed in some of
    the servers, pop the block just wrote in.
if -1 in state:
    port = metafiles['startPortID']
    for i in range(startBlk,lastBlk):
        if state[i]==1:
            self.dataPop(port,path,i,newVersion[i])
                port = (port+1)%self.N
            return False
else:
    # update the version list
    metafiles['version'] = newVersion
    self.putmeta(path,metafiles)
    return True
```

III. Testing

A. Creating Directories

Creating 3 directories.

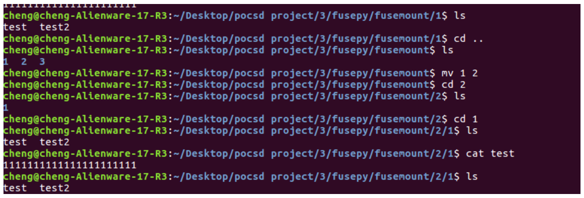


```
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ ls
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ mkdir 1
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ mkdir 2
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ mkdir 3
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ ls
1  2  3
```

Fig. 1: Creating 3 directories.

B. Creating Files

Accessing dir 1 and creating test and test2. Writing text into text.

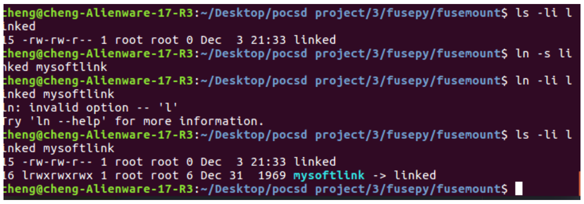


```
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ ls
test  test2
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ cd ..
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ ls
1  2  3
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ mv 1 2
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ cd 2
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/2$ ls
1
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/2$ cd 1
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/2/1$ ls
test  test2
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/2/1$ cat test
1111111111111111111111
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/2/1$ ls
test  test2
```

Fig. 2: File writing.

C. Creating Links

Creating soft links and hard links.

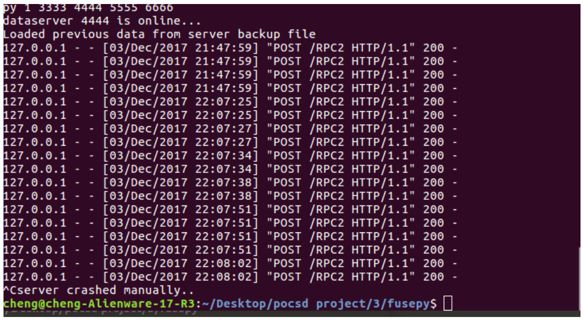


```
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ ls -li l
inked
15 -rw-rw-r-- 1 root root 0 Dec  3 21:33 linked
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ ln -s li
nked mysoftlink
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ ln -li l
inked mysoftlink
ln: invalid option -- 'l'
Try 'ln --help' for more information.
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$ ls -li l
inked mysoftlink
15 -rw-rw-r-- 1 root root 0 Dec  3 21:33 linked
16 lrwxrwxrwx 1 root root 6 Dec 31  1969 mysoftlink -> linked
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount$
```

Fig. 3: Creating links.

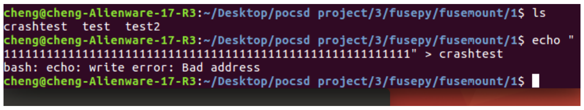
D. Crash Test

Making server 4444 crash manually, and recovering the server later.



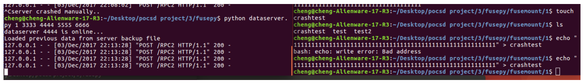
```
py 1 3333 4444 5555 6666
dataserver 4444 is online...
Loaded previous data from server backup file
127.0.0.1 - - [03/Dec/2017 21:47:59] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 21:47:59] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 21:47:59] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 21:47:59] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:25] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:25] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:27] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:27] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:34] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:34] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:38] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:38] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:51] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:51] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:51] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:07:51] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:08:02] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:08:02] "POST /RPC2 HTTP/1.1" 200 -
^Cserver crashed manually..
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy$
```

Fig. 4: The server crashes manually.



```
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ ls
crashtest  test  test2
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ echo "
111111111111111111111111111111111111111111111111111111111" > crashtest
bash: echo: write error: Bad address
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$
```

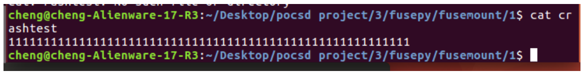
Fig. 5: Write into the crashed server.



```
^Cserver crashed manually..
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy$ python dataserver.
py 1 3333 4444 5555 6666
dataserver 4444 is online...
Loaded previous data from server backup file
127.0.0.1 - - [03/Dec/2017 22:13:28] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:13:28] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:13:28] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [03/Dec/2017 22:13:28] "POST /RPC2 HTTP/1.1" 200 -
```

```
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ touch
crashtest
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ ls
crashtest  test  test2
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ echo "
111111111111111111111111111111111111111111111111111111111" > crashtest
bash: echo: write error: Bad address
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ echo "
111111111111111111111111111111111111111111111111111111111" > crashtest
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$
```

Fig. 6: Recover the server.



```
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$ cat cr
ashtest
111111111111111111111111111111111111111111111111111111111
cheng@cheng-Alienware-17-R3:~/Desktop/pocsd project/3/fusepy/fusemount/1$
```

Fig. 7: Recover the server.

IV. Evaluation

V. Conclusion

VI. Contribution

VII. Appendix

distributedFS.py
dataserver.py
metaserver.py

References

[1] On-line crc calculation and free library. [Online]. Available: https://www.lammertbies.nl/comm/info/crc-calculation.html