EEL 5737 POCSD: Homework #4: ***PartB*** Dt. 11/4/2016
Homework Agenda: Distributed Filesystem

A) Team Members:

Sunal Mittal UFID: 0889 1404
Rohan Naik UFID: 0216 7483

B) Submitted Files:

distributedFS.py

simpleHTmeta.py

simpleHTdata.py

C) Design Implementation:

The existing filesystem code (hierarchicalFS.py) was a multilevel hierarchical filesystem for local storage of files/ directories in a hierarchical structure. The files were stored as an array of blocks of 512 bytes each.

The problem was to divide the attributes of files and directories and the data of files in two parts, namely :

1. metaserver and 2. Dataservers.

Metaserver to store the meta data and Dataservers to store the data blocks in a round robin fashion.

Major modifications in __init__ (self):

Defined metaserver port using argv(2) and dataservers using argv(3) to argv(N+2) where N is number of dataservers.

Meta server is directly assigned using xmlrpclib.ServerProxy

Data servers are created by iterations for number of data servers opened

#Meta ports

Self.rpc = xmlrpclib.ServerProxy(http://localhost: + str(port[i]) + '/')

#Data ports

self.dserver = len(port1)

self.rpc1 = list()

for i in range(0,len(port1))

self.rpc1.append(xmlrpclib.ServerProxy(http://localhost: + str(port1[i] + '/'))

1. **Design implementation of Metaserver:**

Modifications in distributedFS.py:

- Self.files in the existing code contains the metadata of an object. Assigned a new variable (ex: inm) to the dictionary:
  dict(st_mode=(S_IFDIR | 0o755), st_ctime=now, st_mtime=now, st_atime=now, st_nlink=2, files={})
- Dump this variable carrying metadata to the metaserver (SimpleHT.py) with the combination of
  SendData = pickle.dumps(data) - for serialization, and
  rpc.put(Binary(path), Binary(SendData)) -for sending the data over to server.
- The data sent to the server depends on what object has been pickled and the path.
- Depending on the requirement for further functions, the data is again fetched from the server with a combination of:
  value = rpc.get(Binary(path)) for receiving and,
  RequiredData = pickle.loads(value.data)

As a major modification in the code, all functions have been modified for re-defining self.files. Thereby all the self.files instances have been removed (commented) from the code and the same is now accessed from the metaserver on port number **'port', defined in main() – DistributedFS.py**. Hence can be accessed by command :

`python metaserver.py --port 2222`, where 2222 is the port number/ port address

Modifications in traverseparent(self, path) function.

The Parent path in existing code defined as path was renamed as pathx and was also returned through traverseparent return.

- Pathx is consequently used in rename, symlink, unlink, mkdir, rmdir, create etc

Also, when executing rpc.get it is checked for len(pathx) to avoid errors while accessing data from metaserver, in mkdir function.

- The Pickle.loads value of meta data dictionary needs to be checked for length in particular in mkdir function. i.e. if the length is 0, it appends '/' to access the root directory directly.

Modifications in simpleHTmeta.py :

- One modification made in metaserver.py is in the function: get(self, key): (line 45 function) Where rv = {} is modified to rv = False to avoid errors pertaining to blank dictionaries (an error faced while starting and accessing empty dictionary)
- A new function *remove* was introduced to remove values i.e. associated metadata when a file is deleted from the server.

2. Design implementation of dataserver:

- From a broader viewpoint, just like the self.files instance was modified for metadata, the self.data instance will be central entity to work upon for distributed data on data servers.

The Hash Function:

- We initially implemented the round robin assignment pattern beginning from first data server.
- But we realized that this would result in a corner case where all hundreds of files of the size of just one block (512 Bytes). In that case, all these files will be stored on the first server only and it would result in an inefficient and unfair design. Thus the need for a sophisticated hash function was realized.
- This case was addressed by using a unique hashing function wherein the ASCII code of each element of the item in path of a particular file, is computed and summed up.
- Modulo operation is performed on this summation of ASCII code (modulo %N, where N is number or data servers)to randomly assign an index i.e. a data server index from N servers.

Added the hash function :

```
For i in str(path):
    var = var + ord(i)          #ord(i) gives ASCII code of each i
    var%self.dserver            #returns a random data server out of N servers
```

- Major modifications made in functions write, read, rename, getattr, rmdir, mkdir etc wherein all the instances of self.data were replaced by pickled lists, just as in case of self.files
- A new function was introduced to remove values i.e. associated data of files when a file is deleted from the server.

The following operations were checked after all modifications were made in the code:

- Creating directories, subdirectories and files in hierarchical order
- Creating subdirectories within directories with same names.
- Renaming directories, subdirectories and files
- Deleted directories, subdirectories and files
- Operations in a file for reads, writes,
- Operations in a file for overwrite and append
- Fair assignment of initial data server as per the hashing function developed for distributed storage on server
- Moving and Copying files using mv and cp commands at different levels of filesystem