EEL-4736/EEL-5737 Principles of Computer Systems Design
Homework #4 - Assigned: 10/13/2016

Part A
To be done individually - due by 5pm on 10/25/2016

1) Textbook exercise 6.5
2) Textbook exercise 6.6

3) Consider a magnetic hard disk system as follows.
   a) Seeking: the disk arm takes 0.1ms to move between adjacent tracks, and there are a total of 1000 tracks. The average seek latency is 8ms.
   b) Tracks and sectors: there are a total of 1000 tracks, each track storing 1.5MB. Sectors are 4KB in size
   c) The rotation speed is 7200 RPM.
   d) The disk is capable of transferring data out of disk media at 120 tracks per second. The disk is connected to memory through a 3GB/s Serial ATA bus

i) [3] What is the average *latency* of a random read of a *single* sector?

ii) [2] What is the average *throughput* of a random read of a *single* sector?

iii) [3] What is the average latency for a random read of an *entire* track?

iv) [2] What is the avg. throughput of a random read of an *entire* track?

Assume you model the disk as a single-server FIFO queue with exponentially distributed service times, as discussed in class. Suppose each request coming into the queue is for a random sector read. Assume a memory-less request arrival process.

v) [5] At what arrival rate would the disk reach average 80% utilization?

vi) [5] What would be the average length of the queue at 80% utilization?

Submission guidelines:

Submit through Canvas a PDF file named ‘hw4partA.pdf’ with answers to the above questions

Part B - Distributed Remote file system

To be done individually or in groups of 2 - due by 5pm on 11/3/2016

In this homework, you will build a remote file system which stores its data in multiple remote servers. The aim is to distribute the data among multiple servers to reduce their load. The goal of this assignment is to focus on distributing the load across multiple servers - you do not need to worry about fault tolerance at this point. Use the code that you have designed in the previous assignment as a starting point to divide the data into blocks that are stored across multiple servers.

In your design, you will need two kinds of servers. For storing the metadata use a single server (called meta-server); for storing file data, use N servers (called data-servers). You should use an XMLRPC server based on simpleht.py (code provided). You may modify the basic functionality as required. The client (modified memory.py) will act as the FUSE handler as well as an XMLRPC client which connects to multiple servers.

The meta-server will hold the keys (which represent paths) and the values are the metadata information for each file/directory. This means that the hierarchical data-structure constructed previously will have to be broken down into a hashtable. Instead of embedding actual pointers inside your data-structure like last time, use an implicit pointer that will help you traverse the file-system without having to query all the keys in the server. For example, listing the files inside '/dir1' will only require the entry of '/dir1' (and the entries of files/dir inside /dir1 if you're using ls). Having to send/receive and store a dictionary of metadata requires you to serialize/marshal it. Use the Python library called pickle for this purpose.

The N data-servers will hold the data in a manner that will try to distribute the load evenly for all servers. For any file, start storing the Block 0 at a particular server (e.g. determined by a hash function). The subsequent blocks can be stored in a round-robin fashion across all the servers using modulo arithmetic. For example, if `hash(path) = x`, then the blocks can be stored in this manner [x % N, (x+1) % N, (x+2) % N, (x+3) % N ...] where each item in this list represent the server number at which the particular block of data is stored. The selection of hash function is left for you. You can use an existing method or use your own custom/simple hash mechanism that serves the purpose.

Segmenting the files and storing it in a round-robin fashion means that there can be collisions. Say for N = 4, Block 0 and Block 4 will be stored in the same server. So make sure you include in your design, a method to distinguish those keys in the same data server.

Your implementation should be such that any arbitrary N >= 1 should be supported (i.e. at least one meta-server and one data-server). The parameter N is configured and made known to the client program by means of the command-line arguments.

Program arguments and guidelines:

The servers (metaserver.py and dataserver.py) should take the port argument in the format specified below. The client program takes N+2 arguments in the format <fuse_dir>, <metaserver port>, <dataserver1 port>, <dataserver2 port> .. <dataserverN port>.

<u>Example (N=2):</u>

```
python metaserver.py --port 2222
python dataserver.py --port 3333
python dataserver.py --port 4444
python distributedFS.py fusemount 2222 3333 4444
```

The meta-server and the data-server may or may not share the same code.

Testing:

The programs you submit will be tested for all filesystem operations like previous assignments. It will be tested for N <= 5. The rename function must at least perform renaming and moving (mv) of files along with their data.

Submission guidelines:

Submit through Canvas, four files (attached individually) named as follows:

1. distributedFS.py - Client that implements distributed remote filesystem with FUSE and XMLRPC
2. metaserver.py - Modified simpleht.py that hosts your meta-server
3. dataserver.py - Modified simpleht.py that hosts your data-server
4. hw4design.pdf - PDF file describing in detail, the design of your implementation and the tests you conducted to verify functionality