CS 550

Sanchayan Maity
CWID A20340174
Section 03 - IITC, Bangalore

Assignment3: Manual

November 5, 2015

Please note that all commands are executed from the command line in a terminal and it is assumed that the executable program is being run from the same directory where it is present. Please refer the *Output.txt* file for example runs of the program.

There are four primary code files *dht.c*, *workqueue.c*, *error.c* and *common.c*. *dht.c* is the main code file which has the functionality as required by the assignment. *workqueue.c* has been taken from one of the existing examples on the internet and serves as the workqueue implementation required by the server. The other two code files have common code which has been reused since the first assignment. There are no explicit external library dependencies.

Compilation

For compiling run the following *make clean; make dht*

Execution

For running the binary, four compulsory arguments are to be provided from the command line. The arguments are as follows:

- *serverid* This argument needs to be 1 for the first server node, 2 for the second server node, 3 for the third server node and so on. This provides it with the information to pick up the corresponding IP and port combination from the static server configuration file.
- *server conf* This argument is the server configuration file which provides the static configuration of the nodes in the network by providing their IP and port information. Two example server configuration files are included in the source code.
- *directory to be shared* This is the directory which will be shared by the peer viz. all the files from the said directory will be registered by the peer and made available to other peers for download. Note that the shared

directory must be compulsorily provided/ended with a "/". For example, if *shared* is the directory to be shared, it must be provided as *shared/*.

- *ip to bind to* This is the IP to which the server part of the application will bind to. This is made compulsory in case we want to work with Amazon AWS public IP. EC2 instances can communicate with their private as well as public IP. However we cannot bind to public IP but only the private IP. So we can provide the private IP of the EC2 instance on the command line here to bind to while specifying the public IP in the server configuration file.

Please note that all the above arguments need to be provided in the exact same order as no error checking is carried out on any of the arguments.

Some sample command line invocations are as follows:

- *./dht 1 server.conf shared/ 127.0.0.1*
- *./dht 1 aws4.conf shared/ 172.31.4.79*
- *./dht 8 aws4.conf shared/ 172.31.44.67*

Note that before running any tests or any general operation it is assumed that all the nodes are up and running.

For generating test files for testing purposes a script is used which is provided along with source code in *file.sh*. The script takes four arguments in the exact order as follows:

- *server id* This is appended to the sample files generated. The sample files generated start with the name *sample*. The server id is appended to this and after the count will be appended. For example, if 50 files are to be generated, the first file will be *sample11.txt* and the last file will be *sample150.txt*.
- *file size* The file size needs to be specified as the second argument. If a *1KB* file is required, this needs to be specified as *1K*. For a *1MB* file, *1M* and for a *1GB* file as *1G*.
- *Number of files* This argument specifies the number of files to create. The file number along with the server id argument gets appended. See explanation for argument 1.
- *directory name* This argument needs to be the test directory which will be shared and in which the files will be generated.

For example, for generating 5 files of size *1MB* in the *shared* directory present in the current directory from where the script is being run the invocation would be as follows *./file.sh 1 1M 5 shared/*. This would create five files which would be *sample11.txt*, *sample12.txt*, *sample13.txt*, *sample14.txt* and *sample15.txt*.

Testing

Assumptions made are as follows:

- No other operations are carried out before starting the tests.
- Replication command is not given before the test or carried out during the tests.
- All nodes are assumed to up and running waiting for the first test command from the user.
- For further purposes of discussion and explanation it is assumed that the *shared* directory was shared by each node, each node is a different machine and test txt files were created in the *shared* directory using the script discussed earlier.

Please note that the application supports all kind of files. While we talk of mostly txt files in these documents for discussion purposes and that's what we generate with the script, all kinds of files like txt, binary, pdf, tar and such will work.

The register tests are carried out separately from the search and obtain tests. Once the application is started, on the command line user needs to select the option 3 for carrying out the register tests. Ten thousand iterations of the tests are carried out and at the end of the test results will be printed displaying the average time it took to register the given number of files in the shared folder. At the end of register tests, the user should exit from the application by selecting option 6 before running any other tests or carrying out general operations.

Once the application is started, on the command line user needs to select the option 4 for carrying out the search and obtain requests. However before starting the search and obtain tests, the register operation must be executed on all nodes. Without running the register operation first, the nodes will not have the information as to which file is available on which node. After carrying out the register operation, on selecting option 4, user will be asked for the file which will be used for carrying out the search and obtain tests ten thousand times. It is assumed that the file being provided for is atleast with one of the nodes. A file available with the node on which the test is about to be run should never be specified for the test. This is because the obtain file request will come to the same node and it will try to open the file and write to the same file. This condition is not handled at all and is not sensible. This is also applicable for general functioning.

For replication testing, in the directory that is shared, a folder called *replica* must be created beforehand. This is done as replication is provided as a separate feature. All the files from the shared directory would be replicated to the next node and this replicas will be stored in the *shared/replica* directory of that node. If this were not done, when the node on which replica was created does it registration, it will advertise these files as well. We decide not to do this and let replication be a complete almost automatic backend operation. Also if the files were replicated in the same directory that would be shared, this will result in copying of files recursively from one node to the next under the circumstances of the simple logic we use.

Start the application and make sure all nodes are up and running. Replication can be started by selecting option 5. At the next node, *File transfer complete* messages will be printed. The next node's *shared/replica* folder will have the files from the node where replication was invoked. Now for simulation, from any node search for a file which is known to be present at another node. For purposes of illustration it's best if no node has duplicate files. Let's say node 2 has a file sample21.txt which we want to search and then get from node 1. First from all nodes invoke registration process and complete it using option 1. From node 1, we now select option 2 to get the file. We will be asked for the file name and we provide sample21.txt on the command line as entry. The application will show the IP and port address of the peer with whom it is available which will be node 2's details. Before selecting this peer and pressing enter, on node 2 select option 6 to exit. Now node 2 is down. On node 1, now enter the number of peer (which will be 0 if as mentioned a while ago all files were uniquely available with peers) and press enter. The following message will be seen on node 1:

Peer selected is down
Using existing connection for replica
Connecting to replica on node 2
File transfer complete

Note that node 2 is actually node 3. Internally we substract one from the server id's specified on command line and have 0-7 inside the application. The sample21.txt file can be seen to be present in the directory that was shared by node 1. Note that while any directory can be shared, the shared directory must have a replica directory created inside it for the replication logic.

To simulate key value pair replication, test was carried out as follows. We use the *file.sh* script to create a file called sample11.txt. This can be done by running *./file.sh 1 1K 1 shared1/*. Here it is assumed that shared1 is the directory shared by node 1. Since we do not know to which node this file will be hashed to, we enable the debug print around line 612 in dht.c file in *put_at_server* function. It shows that this entry goes to node 1. So this key value pair would have been replicated to node 1 and 2. To simulate this, we first register from node 1 using option 1. Both nodes 2 and 3, would have shown *Connection accepted* message which denotes that the register entry got send to both these nodes. We now kill node 2. From node 4, we directly give the get file command by selecting option 2. This request should have gone to node 2 but now goes to node 3. Node 3 will also show *Connection accepted* message and node 4 shows the following message.

tcp_connect error for 127.0.0.1, 50002
Get operation successful
Available Peers: 1
Peer 0: 127.0.0.1 50001

Note that both the replication experiments above were carried out on local machine for testing using the server.conf file included and using only four nodes for demonstration purposes.