ECE/CS 658 – Internet Engineering

Lab 4 – Searching Contents in a Distributed Application-Layer Network

Due Dates:

- Design report : March 2nd
- Source code and report : March 13th

1. Goals

Develop a simple overlay-based solution that allows a set of nodes to share contents (e.g., music files) among each other.
Consider a set of nodes connected via some overlay topology. Each of the nodes has a set of files that it is willing to share with other nodes. A node in the system (X) that is looking for a particular file issues a query to identify a node (Y) containing that particular file. Once the node is identified, the file can be exchanged between X and Y.
After completing this lab, you will have developed a solution to search for contents in a distributed system. You will be able to design and develop useful overlay-based applications such as simple search engines to find contents in a distributed system.
Note that the main goals are to generate the network topology and to design the search algorithm to locate interesting items. Real file transferring is not implemented but you may try in the future.

You can design your network topology to have a super peer structure or an unstructured topology. Centralised design is not permitted. Your search algorithm should decentralised.

2. Challenge

Phase I:
The first phase of the assignment is to generate the network topology and the contents of each node. The network will consist of 20 to 80 PlanetLab nodes sharing 20-80 files among them, with each node contributing 3-5 files. Some files may be present in multiple nodes. The network can be scaled to contain 20,40 and 80 PlanetLab nodes.

Form the network and initialize the node contents as follows: A new node that comes up gets connected to 2 randomly selected nodes existing in the distributed system. A bootstrap server is provided to facilitate this step (See Section 3). A list of file names is provided, and each node is initialized with 3 to 5 randomly selected files from this list.

Phase II:
The second phase is to design and develop a solution to find the files requested by different nodes. The nodes generate requests for random file names, each of which results in a query that is propagated in the network to find a node (Y) containing the file. The node Y responds to the querying node with its address, which may be used to download the file. Your solutions should satisfy the following requirements:

1. Nodes will communicate using UDP and will follow the message format given in Section 5.
2. The system should continue to operate, albeit with degraded performance, even when some nodes fail.
3. Use the given list of file names & queries to demonstrate your solution.
4. No need to implement file transfer between nodes.

3. Steps

Here we explain joining and maintaining the connectivity in the distributed system. See Section 5 for specific message formats used to talk to nodes and bootstrap server.

1. Each node should take the arguments as follows: ***(Self_port_number, BootStrap_IP_Address, Bootstrap_Port_number)***
2. Each new node added to the system will register at the given Bootstrap Server (BS) by providing node's IP address, port number, and user name. BS will be maintained by the TA.
    - A unique username is essential to keep one students nodes separate from another. Username for student should be **firstname_lastname**
    - BS will respond only to the messages specified in Sec 5.1. Thus, it should be used only to find nodes currently in the system. It will not respond to any network formation or query messages.
3. The first node will receive only an acknowledgement from the BS. The Second node will receive an acknowledgement and the details of the first node. Third node will receive details of the first two nodes. Fourth node onwards will receive details of two randomly selected previously registered nodes.
4. The new node joins the network via the 2 nodes learned from BS using the *JOIN* message, syntax of which is specified in Section 5.2. *JOIN* messages tell the contacted nodes that there is a new node in the system.
5. Each new node will pick 3-5 file names randomly from the given list of file names (filenames.txt). Each node should display the file names that it selected upon request (when a command is issued). Make sure to select different files.
6. When you are running 20 nodes you should read the first 20 file names in filenames.txt and query for first 20 files in queries.txt. When you are running 40 nodes you should read the first 40 file names in filenames.txt and query for first 40 files in queries.txt. When you are running 80 nodes you should read the all 80 file names in filenames.txt and query for all 80 files in queries.txt.
7. You should take care of the case when the file is not found.
8. Ensure that at least 20 nodes are in the system.
9. You are expected to demonstrate the operation of the system as described above. In addition, prepare a report with the following results:

a. Pick 5 nodes randomly and issue list of queries in the given file, one after the other (no parallel queries). All the five nodes should query the file names given in queries.txt according to the number of nodes as mentioned in Step 6, by reading the file line by line. Each query should be able to find at least 1 node with the given file name if such a node exists. You should be able to search for both the entire file name and parts of it. You should write the results in a file called **results.txt**. The naming of the result files should be **results_20.txt for 20 nodes, result_40.txt for 40 nodes and so on**. For the report if you want to transfer the result files into your local machine, you may add the local IP and port in the file name so as to not overwrite all the files in your local machine. Put appropriate spaces and new lines to show your results.
    - e.g.: If query ask for "Lord", "Lord rings", or "Lord of the rings" consider "lord of the rings" as a match if the node has a file with that name. Consider only complete words, e.g., if you search for "Lord", file with "Lo Game" is not a match similarly if you search "Lo", "Lord of the ring" is not a match.
b. Find number of application-level hops and latency required to resolve each query. After resolving all the queries, find number of query messages received, forwarded, and answered by all nodes. Also find their routing table sizes and any routing related overhead/messages that may be involved (if any). This should be diplayed on the terminal after each query for all nodes.
c. Remove nodes form the distributed system one at a time. Before leaving, a node must inform all the nodes in its routing table that it is leaving using *LEAVE* message. It must also tell the BS that it is leaving (using *UNREG* message).
d. Repeat Steps 1-6.c 5 times and collect all the statistics. Every time pick a different node as the 1st node.
e. Repeat Steps 1-6.d for networks of three different sizes corresponding to 20,40 and 80 PlanetLab nodes.
f. Find min, max, average, and standard deviation of hops required , latency, messages per node, and node degree for the three different sizes of networks. Also find per query cost and per node cost. Plot the cumulative distribution (CDF) of hops, latency, messages per node, and node degree in different scales of networks.
g. Submit a final report which includes your findings from Step 6f. Also discuss how your solution will behave in networks of different scales, if number of queries (Q) is much larger than number of nodes (N) (Q >> N) and vice versa (N >> Q). The network that you create has a special property. Comment on it (hint: use node degree). Comment on how to improve the query resolution while reducing messages, hops, and the latency. Also comment on how the network scaling affects messages, hops, and latency.

4. Things to Note

1. Prepare a design document describing how you will implement this solution. Get approval for your design from TA before coding. Design report should at least include expected topology, how to communicate among nodes, format of routing table, performance parameters, how to capture them, and pseudo codes.
2. Use a layered design as you will be able to reuse part of this code in upcoming labs.
3. If your program crashes, you will get a 9998 from BS when you try to register a node (REG) again (unless you use a different IP or port). To simplify and maintain a consistent view of the distributed system at the BS (if this happens), your node needs to unregister before attempting to register again. If you are going to rerun a node with a different IP, you have to issue an unregister request (*UNREG*) for the previous entry. You may do this through netcat (only for testing and debugging) by manually issuing the command.
4. Use string *tokenize* to break a command and extract different elements of it.
5. You should display all the commands mentioned in Section 5 that are being sent and received on the terminal. You should specify whether is being sent or received. You should also specify the IP adress and Port Numbers to which you are either sending or receiving the command. Put proper spacing between each command on the terminal. Also print the display elements mentioned in Section 6b only after each query.
6. You should be able to display the following on the node terminal upon typing the request on the console.
    1. details: Print Self_IP, Self_Port
    2. neighbours: Print IP and Port numbers of neighbours
    3. files : Print the files that the self node contains
    4. search: This command should start reading the queries.txt and start searching for files
7. The results of each query should be displayed on the quering node console. (As well as be written in results.txt)
8. You can do the lab individually or as a group of two. If done individually, you will get a maximum of 10 extra credits.
9. Opening separate terminals to connect to each PlanetLab node could be tedious. Instead, use "pssh" command to "ssh" remote control programs in parallel. Here is a useful tutorial telling you how to use "pssh". You can save the IPs of all PlanetLab nodes in a "ips.txt" file and "pssh" will look up the IPs in this file for parallel remote control. Use "pssh" command to run programs on remote host machines and "pnuke" command to stop the programs in parallel. Also you can use "pscp" command to transfer the files to a bunch of remote host machines. You should create your own folder in each PlanetLab node identified by a unique name. When using "pssh" commands, enter the directory in command line correctly to avoid transferring files to the wrong folder. Note that "pssh" is not installed in ENGR linux servers, you may need to download and install it in your own linux or virtual machine.
10. Follow proper naming conventions and arguments order.
11. You can also use Shell Scripting to ssh and scp into the PlanetLab Nodes. Here is an example script :exampleScript.sh
12. Log into your PlanetLab account to check the complete list of nodes in cnrl slice for ip addresses. You may add nodes to cnrl slice if necessary
13. Actively participate in discussion to clarify any thought on Canvas and talk to TA whenever you have concerns...
14. Start the lab early...

5. Protocol

We will use a character-based protocol to make it easy to debug. You may issues commands through netact to the BS and other nodes to check whether commands are correctly responded. Each message starts with a *command* (in uppercase characters) that can be up to *n* characters long. Rest of the message will depend on the command. Each element in the command is separated by a *white space*.

5.1 Register/Unregister With Bootstrap Server

***Register Request message* – used to register with the BS**

```
length REG IP_address port_no username
```

- *e.g., 0036 REG 129.82.123.45 5001 1234abcd*
- *length* – Length of the entire message including 4 characters used to indicate the length. Always give length in xxxx format to make it easy to determine the length of the message.
- *REG* – Registration request.
- *IP_address* – IP address in xxx.xxx.xxx.xxx format. This is the IP address other nodes will use to reach you. Indicated with up to 15 characters.
- *port_no* – Port number. This is the port number that other nodes will connect to. Up to 5 characters.
- *Username* – A string with characters & numbers.

***Register Response message* – BS will send the following message**

```
length REGOK no_nodes IP_1 port_1 IP_2 port_2
```

- *e.g., 0051 REGOK 2 129.82.123.45 5001 64.12.123.190 34001*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *REGOK* – Registration response.
- *no_ nodes* – Number of node entries that are going to be returned by the registry
  - If = 0, request is successful no nodes in the system
  - If 1 or 2, request is successful, list of nodes contacts will be returned
  - If = 9999, failed, there is some error in the command
  - If = 9998, failed, already registered to you, unregister first
  - If = 9997, failed, registered to another user, try a different IP and port
  - If = 9996, failed, can't register. BS full.
- *IP_1* – IP address of the 1st node (if available).
- *port_1* – Port number of the 1st node (if available).
- *IP_2* – IP address of the 2nd node (if available).
- *port_2* – Port number of the 2nd node (if available).

***Unregister Request message* – used to unregister from the BS**

```
length UNREG IP_address port_no username
```

- *e.g., 0028 UNREG 64.12.123.190 432*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *UNREG* – Unregister request.
- *IP_address* – IP address in xxx.xxx.xxx.xxx format. This is the IP address other nodes will use to reach you. Indicated with up to 15 characters.
- *port_no* – Port number. This is the port number that other nodes will connect to. Up to 5 characters.
- *username* – A string with characters & numbers. Should be the same username used to register the node.

***Unregister Response message* – BS will send the following message**

```
length UNROK value
```

- *e.g., 0015 UNREGOK 0*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *UNREGOK* – Unregister response.
- *value* – Indicate success or failure
  - If 0 – successful,
  - 9998 -- IP and Port combination not registered for this username
  - 9997 -- This username is not present
  - 9999 – error while unregistering

For any message BS cannot understand it will send an error of the format in Section 5.5

5.2 Join Distributed System

***Request message* – used to indicate presence of new node to other nodes that is found from BS**

```
length JOIN IP_address port_no
```

- *e.g., 0027 JOIN 64.12.123.190 432*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *JOIN* – Join request.

- *IP_address* – IP address in xxx.xxx.xxx.xxx format. This is the IP address other nodes will use to reach you. Indicated with up to 15 characters.
- *port_no* – Port number. This is the port number that other nodes will connect to. Up to 5 characters.

Response message

```
length JOINOK value
```

- *e.g., 0014 JOINOK 0*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *JOINOK* – Join response.
- *value* – Indicate success or failure
  - If 0 – successful, if 9999 – error while adding new node to routing table

5.3 Leave Distributed System

***Request message* – used to indicate this node is leaving the distributed system**

```
length LEAVE IP_address port_no
```

- *e.g., 0028 LEAVE 64.12.123.190 432*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *LEAVE* – Leave request.
- *IP_address* – IP address in xxx.xxx.xxx.xxx format. This is the IP address other nodes will use to reach you. Indicated with up to 15 characters.
- *port_no* – Port number. This is the port number that other nodes will connect to. Up to 5 characters.

Response message

```
length LEAVEOK value
```

- *e.g., 0015 LEAVEOK 0*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *LEAVEOK* – Leave response.
- *value* – Indicate success or failure
  - If 0 – successful, if 9999 – error while adding new node to routing table

5.4 Search for a File Name

***Request message* – Used to locate a key in the network**

```
length SER IP port file_name hops
```

- e.g., Suppose we are searching for *Lord of the rings*, *0047 SER 129.82.62.142 5070 "Lord of the rings"*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *SER* - Locate a file with this name.
- *IP* - IP address of the node that is searching for the file. May be useful depending your design.
- *port* - port number of the node that is searching for the file. May be useful depending your design.
- *file_name* - File name being searched.
- *hops* - A hop count. May be of use for cost calculations (optional).

***Response message* – Responce to query originator when a file is found.**

```
length SEROK no_files IP port hops filename1 filename2 ... ...
```

- e.g., Suppose we are searching for string *baby*. So it will return, *0114 SEROK 3 129.82.128.1 2301 baby_go_home.mp3 baby_come_back.mp3 baby.mpeg*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *SEROK* - Sends the result for search.
- *no_files* - Number of results returned
  - If ≥ 1, Successful
  - If = 0 no matching results. Searched key is not in key table
  - If = 9999, failure due to node unreachable
  - If = 9998, some other error.
- *IP* - IP address of the node having (stored) the file.
- *port* - Port number of the node having (stored) the file.
- *hops* - Hops required to find the file(s).
- *filename* - Actual name of the file.

5.5 Error Message

```
length ERROR
```

- *0010 ERROR*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *ERROR* - Generic error message, to indicate that a given command is not understood. For storing and searching keys this should be send to the initialter of the message.

6. Testing with Bootstrap:

- The bootstrap is running on the following nodes locally (IP_Address Port). **Please use these nodes to do your testing of codes before you test on PlanetLab.**
  - denver.cs.colostate.edu 5000
  - turnip.cs.colostate.edu 5000
  - wasabi.cs.colostate.edu 5000
- On PlanetLab, the Bootstrap is running on the following nodes :
  - planetlab-coffee.ait.ie 5000
  - pli1-pa-4.hpl.hp.com 5000
- **The executable files for the BootStrap Server are uploaded on Canvas.** you can access them by going to Files->Lab4. Download the entire BootstrapCode folder. Run the server by running "java BootstrapServer port_number" once you are in this folder. The server will run on the localhost where you are running code on the specified port_nmber.
- Make sure to use the specified username format
- For Testing, follow the below steps;
  - On your terminal, run "nc -u *Bootstrap_IP Bootstrap_port*"
  - The following commands will give you the output witht the BootStrap Server:
    - PRINT user_name : All the registered nodes for this user_name
    - Using register and Unregister commands given in Section 5.1 you can manually register and Unrgister with the BootStrap.
  - If your program crashes, you will need to unregister your node manually.
  - The BS will unregister the node only if you give the correct IP, Port and Username you want to unregister.
  - Use PRINT to find out registered nodes for your username.
  - The BS will answer and give error outputs as mentioned in Section 5.1. Please follow the same for error codes.
  - Here is a screenshot of how you can use the above steps:
  - Bootstrap Server.pdf

7. Demo

- You need to run at least 20 nodes in PlanetLab.
- **You should download your submitted file on Canvas and transfer the files to the PlanetLab nodes at the time of demo. Make use of pscp or bash Scripting.**
- Should work even if few random nodes are removed.

8. What to Submit

- Design document (must not exceed 3 pages). (Should not be included in the final submission. Should be mailed to the TA by design due date)
- Report analyzing performance (must not exceed 10 pages). Your report should include the following:
  - Frontpage
  - A short summary of the design.
  - Results
  - Statistics and discussion (See section 6d- 6g)
- You need to submit your code and the makefile (if any). However, do not forget to submit readme.txt file with instructions for compilation and how to run your node
- Please do NOT submit executables. Submit all files as a single compressed file (.zip or .tar.gz is preferred). Your uncompressed

and compressed folder should be named **Lastname_Firstname_Lab4**
- Upload your submission on Canvas by the due date and time.
- **Those who will be doing the lab in groups, should both upload their submissions on Canvas**
- **Your most recent submission on Canvas will be considered as your final submission.** Whatever is the time and date of your most recent submission, will be considered. No exceptions. You will have to download the code of your final submission from Canvas for demo.

9. Late Penalty

- There will a 5% late penalty per day.

10. Grading

- Design - 15%. Do a good design while thinking about the given protocol specification.
- Demo - 50% . You will also face a couple of questions during the demo that should be answered for full score.
  - 35% - Correct working of lab. All files are search correctly.
  - 10% - Display of all required details asked to be displayed on the terminal
  - 5% Clarity and neatness in display of results. Give appropriate spaces, tabs and new lines.
- Report - 25%
  - 20% - Content
  - 5% - Presentation
- Coding style - 10%. Use layered design. Make use of generic functions. Comment your code.
- Extra credits - 10%.

Prepared by Dilum Bandara and Anura P. Jayasumana
Last updated 02/20/2015 by Divyanka Bose