Group Project - Content Searching in a Distributed Overlay Network

Due Dates

- Design Document - Sep 29, 2017
- Part 2 Implementation - Oct 25, 2017
- Part 3 Implementation and Demo - Nov 9 and 10, 2017
- Final Report and Personal Statement - Nov 17, 2017

Note

- Each project group consists of 4 students. You are free to pick your group members.
- Use Yammer group for project related discussions.

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 project, you will have developed a solution to search for contents in a distributed system. You will be able to:

- design, develop, and debug overlay-based applications such as a simple search engine to find contents in a distributed system
- use RPCs or web services to develop distributed systems
- measure an analyze the performance of a distributed system

2. Challenge

The challenge consists of 4 phases. The 1st phase of the assignment is to generate the network topology and the contents of each node. The network will consist of 10+ nodes sharing 20 files among them, with each node contributing 3-5 files. Some files may be present in multiple nodes. Form the network and initialize the node contents as follows:

1. A new node that comes up gets connected to 2 randomly selected nodes existing in the distributed system. A Bootstrap Server (BS) is provided to facilitate this step.
    - Following explains how to join and maintain the connectivity in the distributed system. See Section 4 for specific message formats used to talk to nodes and BS.
        1. Each new node added to the system will register at the given BS by providing node's IP address, port number, and user name.
            - A unique username/key is essential to keep one student's nodes separate from another.

        - BS will respond only to the messages specified in Section 4.1. Thus, it should be used only to find nodes currently in the system. It will not respond to any network formation or query messages.
    2. The 1st node will receive only an acknowledgement (ACK) from the BS. The 2nd node will receive an ACK and the details of the first node. 3rd node will receive details of the first two nodes. 4th node will receive details of first three registered nodes, and so on (all nodes are sent from BS only to simplify debugging). However, from the 4th node on wards you should join only to 2 randomly selected nodes from the list of nodes you received.
    3. The new node joins the network via the 2 nodes learned from BS using the *JOIN* message, syntax of which is specified in Section 4.2. *JOIN* messages tell the contacted nodes that there is a new node in the system. UDP sockets are to be used to facilitate this communication.
        - If your solution uses a routing/neighbor table, each node should either display it or should be able to do so upon request (when a command is issued).
    4. If your program crashes, you will get a 9998 from BS when you try to register *(REG)* the same node 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 the node with a different IP or port no, you have to issue an unregister request *(UNREG)* for the previous entry. You may do this through Telnet (only for testing and debugging) by manually issuing the command. You may issue *'PRINT'* command through Telnet to see all the valid entries in the BS.
2. A list of file names is provided, and each node should be initialized with 3 to 5 randomly selected files from this list.
    - Each node should either display the file names that it selected or should be able to do so upon request (when a command is issued).

The second phase is to design and develop a socket-based solution to find the files requested by different nodes. Query process is as follows:

1. Nodes generate requests for random file names.
2. Each request results in a query that is propagated in the network to find a node (Y) containing the file.
3. Node Y responds to the querying node with its address, which may be used to download the file.

Your solution for Phase 2 should satisfy the following requirements:

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

The third phase is to extend your solution using RPCs or web services. Your modified solution in Phase 3 should satisfy the following requirements:

- Implementation may be based on either RPCs or web services. If you have already written web services, you are recommended to extend the solution using RPC. Otherwise, it is recommended to use web services.
- Update the Phase 2 design to reflect the RPCs or web services based query resolution. No need to modify the communication with the BS or neighbors while setting up the network. You may change the message format, if required.
- Modify program in Phase 2 to reflect the RPCs or web services based implementation.

The final phase is to analyze the performance of the solutions developed in 2nd and 3rd phases.

3. Steps

1. Prepare a design document describing how you will implement this solution.
    - Report should at least include proposed topology to connect nodes, how to communicate among nodes, format of routing table(s), performance parameters, how to capture them, and pseudo codes (when possible).
    - Use a layered design. Then you will be able to reuse part of this code in Phase 3.
    - Check with the lecturer/instructor before using any high-level libraries or reusing any 3rd-party code.
    - Get approval for your design from the lecturer/instructor before committing lots of time for coding.
    - Actively participate to discussions on Yammer and talk to lecturer whenever you have concerns.
2. Develop Phase 2 of the solution.
3. Develop Phase 3 of the solution.
4. Conduct a performance analysis using solutions for both Phase 2 and Phase 3.
    - Ensure that at least 10 nodes are in the system.
    - You are expected to demonstrate the operation of the system as described above. In addition, prepare a report with the following results:
        1. Pick 3 nodes randomly and issue list of queries in the given file, one after the other (it is not essential to issue parallel queries from the same node). Each query should attempt (best effort) to find at least 1 node with the given file name, if such a node exists. You should be able to search for the entire file name as well as parts of it.
            - e.g.: If query ask for "Lord", "Lord rings", or "Lord of the rings", "lord of the rings" should be considered as a match. 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.
        2. Find number of application-level hops and latency required to resolve each query. After resolving all the queries, find the number of query messages received, forwarded, and answered by all 10+ nodes. Also find their routing table sizes and any routing related overhead/messages that may be involved (if any).
        3. Remove 2 random 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). We will consider only graceful departure of nodes.

4. Repeat Steps 1 and 2, and collect all the statistics.
5. Find min, max, average, and standard deviation of hops, latency, messages per node, and node degree. Also find per query cost and per node cost. Plot distribution (CDF) of hops, latency, messages per node, and node degree.
6. Prepare a report by including your findings and critically evaluating them. Also discuss how your solution will behave, if number of queries (Q) are much larger than number of nodes (N) (Q >> N) and vice versa (N >> Q). Comment on how to improve the query resolution while reducing messages, hops, and latency.
7. Write a personal reflections statement that explains your contribution to project, other group members contributions, what you learned from the project, what you like and dislike, etc. This is a private document that's not shared with other team members.

4. Protocol

We will use a character-based protocol to make it easy to debug. You may issue commands through Telnet 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*.

4.1 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 and 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
    - 0 – request is successful, no nodes in the system
    - 1 or 2 – request is successful, 1 or 2 nodes' contacts will be returned
    - 9999 – failed, there is some error in the command
    - 9998 – failed, already registered to you, unregister first
    - 9997 – failed, registered to another user, try a different IP and port
    - 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).
- To simplify the debugging BS will actually return all the nodes known to it. However, you should use only the first 2 nodes to connect.

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., 0012 UNROK 0*
- *length* – Length of the entire message including 4 characters used to indicate the length. In xxxx format.
- *UNROK* – Unregister response.
- *value* – Indicate success or failure.
    - 0 – successful
    - 9999 – error while unregistering. IP and port may not be in the registry or command is incorrect.

For any message BS can't understand it will send an error of the format in Section 4.5.

Testing

You can use netcat to test communication with the bootstrap server. Try the following from a unix/linux terminal.

```
$ nc -u node1.cse.mrt.ac.lk 5000
```

Then issue register and unregister commands. e.g.:

```
0033 REG 192.248.230.150 57000 vwb
```

4.2 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
  - 0 – successful
  - 9999 – error while adding new node to routing table

4.3 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., 0014 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

4.4 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 – Response 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. The node that sends this message is the one that actually stored the (*key*, *value*) pair, i.e., node that index the file information.
- *no_files* – Number of results returned
    - ≥ 1 – Successful
    - 0 – no matching results. Searched key is not in key table
    - 9999 – failure due to node unreachable
    - 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.

4.5 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 files/keys this should be send to the initiator of the message.

5. Demo

- You need to run at least 10 nodes.
- Need to show that your nodes get registered with the BS.
- Need to show what file names are in your node.
- Should work even if few random nodes are removed.
- Plan for a 10 min demo and 5 min viva.
- All group members must be present.

6. What to Submit

- Design document (must not exceed 3 pages) - By Deadline 1.
- 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 nodes - By Deadline 2 and 3.
- Report analyzing performance (must not exceed 5 pages) - By Deadline 4.
- Personal reflections statement - By Deadline 4.
- Please do NOT submit executables. Submit all files as a single compressed file (.zip or .tar.gz is preferred). Applies both the Deadline 2 and 3.
- Submit files to Moodle by respective deadlines.

7. Grading

- Design - 25%. Do a good design while considering the given protocol specifications.
- Implementation and Demonstration - 40%, Your code will be tested for a combination of inputs.
- Final Report - 25%. Proper presentation and analysis of the results.
- Personal Reflection Statement - 10%

8. Resources

File names, Queries, and Bootstrap Server code given on Moodle course page.

Credits

This project is derived from Dilum Bandara and Anura P. Jayasumana, "Lab 4 – Searching Contents in a Distributed Application-Layer Network," ECE 658 – Internet Engineering class at Colorado State University, 2012.