CS 550 Programming Assignment #2

A Simple Distributed Hash Table

Instructions:

- ***Due date: 11:59PM on Monday, 10/12/15***
- ***Maximum Points: 100 points***
- ***Maximum Extra Credit Points: 10 points***
- *This is an individual assignment, although you may work in groups to brainstorm on possible solutions; your code implementation, report, and evaluation must be your own work.*
- *Please post your questions to the Piazza forum.*
- *Only a softcopy submission is required; it must be submitted to "Digital Drop Box" on Blackboard.*
- *For all programming assignments, please submit just the softcopy; please zip all files (report, source code, compilation scripts, and documentation) and submit it to BB.*
- *Name your file as this rule: "PROG#_LASTNAME_FIRSTNAME.{zip|tar|pdf}". E.g. "Prog2_Raicu_Ioan.tar".*
- *Late submission will be penalized at 10% per day (beyond the 7-day late pass).*

1 The problem

In this programming assignment you are going to implement a distributed hash table, such as ZHT (http://datasys.cs.iit.edu/projects/ZHT/). You can be creative with this project. You are free to use any programming languages (C, C++, Java, etc) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. Also, you are free to use any computer as long as it runs Linux. Your assignment will be graded in a Linux environment, and you will loose many points if the TAs cannot compile and run your assignments.

If you are not familiar with what Hash Tables are, see the Wikipedia article for more details:

- https://en.wikipedia.org/wiki/Hash_table

For more information on Distributed Hash Tables (also known as NoSQL distributed storage systems), see the following documents:

- Wikipedia Article: https://en.wikipedia.org/wiki/Distributed_hash_table
- Paper: http://datasys.cs.iit.edu/projects/ZHT/ZHT-CRC-PID2666213-Final.pdf
- Slides: http://datasys.cs.iit.edu/projects/ZHT/001-Public--ZHT-05-22-2013-IPDPS.pptx
- Source code: https://github.com/mierl/ZHT

Pay more attention to its architecture and design goals rather than its protocol details, since we are not strictly going to be using all the features of the ZHT distributed hash table.

In this project, you need to design a distributed hash table. Like in programming assignment 1, each peer should be both a server and a client. As a client, it provides interfaces through which users can issue queries and view search results. As a server, it accepts queries from other peers, checks for matches against its local hash table, and responds with corresponding results. In addition, since there's no central indexing server, search is done through consistent hashing.

Below is a list of what you need to implement:

1. First of all, we are not implementing the dynamic initialization of the network. To keep things simple, assume that the structure of the network and servers is static. Your network and servers will be initialized statically using a config file that is read by each server at startup time. You are free to use any format for your config file. The only requirement is that it provides a list of all servers in the system.
2. You must implement the following two operations:
    a. boolean=put(key,value)
        i. the put operation should return success or failure
        ii. key should be of type string
        iii. value should be of type string
    b. value=get(key)
        i. the get operation should return the value or null
        ii. key should be of type string
        iii. value should be of type string
    c. boolean=del(key)
        i. the del operation should return success or failure
        ii. key should be of type string
3. Put, Get, and Del operations are sent to the correct server by performing a hash on the key.

Other requirements:

- Use threads so your peer can serve multiple requests concurrently.
- Messages can be fixed size at 1024 bytes
    - Keys can be limited to be 20 bytes long
    - Values can be limited to 1000 bytes long
    - An optional header can be used that is 4 bytes long; if no header is used, keys can be increased to 24 bytes long
- No GUI is required.

Extra credit (10%):

- Implement resilience by allowing system wide replication of key/value pairs; make sure to test your fault tolerance by killing one of your servers and ensuring that your client can still locate a replica.

2 Evaluation and Measurement

Deploy 8 servers. They can be setup on the same machine (different directories) or different machines, but your code must work in a multi-machine environment (no reliance on shared file systems or pipes). Perform an evaluation on running 100K put operations per client (you can name the keys 100000…199999 for client 1, 200000…299999 for client 2, etc), then 100K get operations per client, and finally 100K del operations. Your first experiment should have 1 client doing this experiment. Then 2 clients concurrently. Then 4 clients. And finally 8 concurrent clients. If the experiment is too short to get all 8 clients running concurrently, you can increase the number of operations to 1M operations. Compute the average response time per client query request by measuring the average response time seen by a client.

3 What you will submit & Grading

When you have finished implementing the complete assignment as described above, you should submit your solution to 'digital drop box' on blackboard. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Source Code (30 points):** You must hand in all your source code, including with in-line documentation.

2. **Makefile/Ant (5 points):** You must use Makefiles or Ant to automate your programming assignment compilation
3. **Manual (10 points):** A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step.
4. **Compiles Correctly (10 points):** Your code must compile in a Linux environment.
5. **Output file (10 points):** A sample output of you running your program (e.g. 10 operations of each command is sufficient). Describe any cases for which your program is known not to work correctly
6. **Design Doc (15 points):** You must write about how your program was designed, what tradeoffs you made, etc. Also describe possible improvements and extensions to your program (and sketch how they might be made).
7. **Performance Evaluation (20 points):** You are to conduct a basic performance evaluation
8. **Extra Credit (10 points):** Add resilience for the key/value pairs.

Please put all of the above into one .zip or .tar file, and upload it to 'digital drop box' on blackboard'. The name of .zip or .tar should follow this format: "PROG#_LASTNAME_FIRSTNAME.{zip|tar|pdf}".

Please do NOT email your files to the professor and TA!!

Grades for late programs will be lowered 10% per day points per day late (beyond the 7-day late pass).