Project 1: P2P Architecture – Building Content-Addressable Network (CAN)

Due: October 2nd, 2015 23:59:59pm

1 Overview

A Distributed Hash Table (DHT) helps searching for a file efficiently with a keyword in peer-to-peer (P2P) networks. Those DHT-based P2P networks are referred to as structured P2P networks. On the contrary, unstructured P2P networks use flooding or a central server for search. Structured P2P networks offer more scalable, efficient, and robust search and management. CAN, Pastry, and Chord are a few examples of the structured P2P networks. You will implement a CAN P2P system in this project. For implementation, you are free to use one of Java, C, C++, or Python, and any networking techniques including sockets, RPC and RMI.

2 Background

The paper below describes the details of the CAN protocol. You should first read the paper VERY carefully, and understand it completely. Feel free to read any related documents (except its implemented program). You can find the paper on Blackboard, or search for it on Google. You can also download the paper by clicking on the title below.

S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, **A Scalable Content-Addressable Network**. In Proceedings of ACM SIGCOMM, August 2001.

After reading the paper, you should be able to answer the following questions:

1. How does a node insert and retrieve a file with a keyword?
2. How is a search query routed from source to destination?
3. How does a new peer join CAN? Specifically, how does CAN compute the zone of the new peer? Who are the neighbors of the new peer?
4. How does a peer leave CAN? Specifically, how is CAN re-organized after node departures?

You need to focus on Section 2 and Figures 1-3 to answer these questions. For simplicity, you assume that nodes and networks do not fail; hence, you do not need to implement the CAN recovery mechanism.

3 What to do

Your program is to support the four main features:

1. the file insert and retrieve algorithm
2. the routing mechanism

3. the node join and leave protocol

4. displaying peer information

Of course, you are always welcome to add more functionalities either explained in the paper or your own ideas. In such a case, be sure to specify clearly in your README. Additionally, one peer needs to be designated as the bootstrapping server that helps a new node join the CAN network successfully. As discussed in the paper, the server randomly chooses the coordinate of the new node and routes a join request to this target peer. A new node can use any active peer as the bootstrapping server.

You will use a two-dimensional coordinate space (like a square) for this project. The side-length of this square is 10, and the number of peers in the experiment is no greater than 10. To determine a mapping point in the coordinate space, you will use a simple modulo-based hash function. Specifically, the x-coordinate of a keyword (a string) is computed as *CharAtOdd mod 10*, where *CharAtOdd* is the addition of the character values at odd positions. The y-coordinate of the keyword is computed similarly, which is *CharAtEven mod 10* where *CharAtEven* is the addition of the character values at even positions. Note that a floating-point number can be used for the coordinates for accurate zone splitting. Assume that keywords have at least 5 characters.

When a node joins, if its zone is a square, then split the zone vertically (this will give the node a rectangle zone). If the zone is a rectangle with height greater than width, split the zone horizontally (this will give the node a square zone). When a node leaves, search its neighbors and merge its zone with a neighbor zone if that merge creates a rectangle or square zone. Otherwise, the neighbor with the smallest zone owns the zone of the departing node temporarily.

Once your CAN has started correctly, your program is to accept the commands listed below:

Command	**Description**
insert *keyword*	Insert a file with *keyword* starting from *peer*, where *peer* is a node identifier, not an IP address. After a successful insertion, display which peer stores the file and the route at the IP layer from peer to the destination peer. If the insertion fails, display "Failure".
search *keyword*	Search for a file with *keyword* starting from *peer*, where *peer* is a node identifier, not an IP address. After a successful search, display which peer stores the file and the route at the IP layer from *peer* to the destination peer. If the search fails, display "Failure".
view	Display the information of a specified peer *peer* where *peer* is a node identifier, not an IP address. The information includes the node identifier, the IP address, the coordinate, a list of neighbors, and the data items currently stored at the peer. If no *peer* is given, display the information of all currently active peers.
join	A new node *peer* joins the CAN network. After a successful join, display the peer information (see view *peer*). If the join fails, display "Failure". If no *peer* is given, all the hosts join the CAN one after another.
leave	A node *peer* leaves the CAN network. After a successful leave, display the information at affected peers (neighbors). If the leave fails, display "Failure".

All the error cases and exceptions should be handled properly. This means your system should not crash abruptly. As long as the minimum requirements (the four main features) are met, the project specifications can be flexible. New ideas, possible extensions, and other suggestions are always welcome.

4 How to submit

To submit the project, you should first create a directory whose name is "your BU email ID"`-project1`. For example, if your email ID is `jdoe@binghamton.edu`, you should create a directory called `jdoe-project1`. You should put the following files into this directory:

1. Your report about your design and implementation choices.

2. Your source code.

3. A `README` file describing the programming language(s) you are using, how to compile and run your code, how to test your program, and sample input/output.

Compress the directory (e.g., `tar czvf jdoe-project1.tgz jdoe-project1`) and submit the tarball (in `tar.gz` or `tgz` format) to Blackboard. Again, if your email ID is `jdoe@binghamton.edu`, you should name your submission: `jdoe-project1.tar.gz` or `jdoe-project1.tgz`. Failure to use the right naming scheme will result in a 5% point deduction.

Your project will be graded on the CS Department computers `remote.cs.binghamton.edu`. If you use external libraries in your code, you should also include them in your directory and correctly set the environment variables using absolute path in the Makefile. If your code does not compile on or cannot correctly run on the CS computers, you will receive no points.