

CSci 5105: Foundations of Modern Operating Systems – Spring 2007

Programming Assignment 2 DHT-based File System Due Fri , March 30th at 11:59pm

Overview

In this assignment you will build a DHT-based file system. You should use the same language (C/C++) as you did in Programming Assignment 1 (to anticipate a merging of assignments later).

Supported DHT, File operations:

1. Insert File - Inserts file into the DHT file system
2. Query File - Searches for the (host) location of a file in the DHT file system, then downloads that file from its location to the client.
3. Node joins DHT - Node wishes to join the DHT file system, and should be given a portion of the files in the file system (to redistribute load)
4. Node leaves DHT - Node leaves the DHT file system; files contained on this node should be transferred to other nodes in the DHT before it leaves

Algorithm, Hash Functions & Keys

For simplicity and to maintain your sanity, this system should be implemented via RPC.

We will be implementing the Chord system with finger tables as described in pp188-190 in the textbook. Do not be concerned with proximity, as this will be implemented within a local cluster of machines (our ITLabs network).

You should choose an appropriate hash function and apply it to the determine the keys associated with files. Such example hash functions include 'md5sum' and 'cksum' (available on ITLabs Unix systems).

For instance, you can take the hash value for a file, modulo MAX_NODE_ID , to get a resulting key. MAX_NODE_ID is generally expected to be the same order as the maximum number of expected nodes in the system. For example, in a 10-node system, you might set MAX_NUM_NODES to $2^6=64$.

Since there will be only a few nodes in the system, we can restrict the key space. In particular, the key space should be comparable to the max number of nodes you can expect to have.

Inserting and Querying Files

There will be a client that invokes file insertions and queries. Each operation should use `find(key)` which returns the destination node along with the path. Actual file transfers will occur directly between the client and destination node. (Hint: send files as a string-type return value over RPC.)

To generate files: a file of size X can be generated by simply writing X chars (like 'a') to a file. The

names can be randomly generated.

Nodes Joining and Leaving

For simplicity reasons, you require there to exist a single well-known super-node that exists at all times. This super-node can also be used as the initial node in the system. Then new nodes wishing to join could simply contact this (statically-configured) super-node.

You will be expected to keep appropriate finger tables, and to keep them properly updated as nodes join and leave the system. For simplicity for now, we will assume that nodes never fail; when they leave the system, they can inform other surrounding nodes that they are leaving. Also the distribution of files must be changed as nodes join or leave; this must be done appropriately to support the set of node IDs and file keys in the system.

Paths To Find/Insert Files

When querying for the location of a file OR inserting a new file, the result should not only include the final host location, but also the path taken along the algorithm to find the host. Two such paths are listed as examples in Figure 5-4 on page 189 of the textbook.

Observing Your System

You should be able to demonstrate that the finger tables are properly kept up to date. It is suggested you provide an RPC function that neatly prints out the finger table for the host.

Also, for observation purposes, you should include functionality to view the distribution of files in the system; for example, this shows if the current distribution is 'balanced'.

File System Analysis & Evaluation

As before, you will submit a report from an analysis of your system. Suggested items to cover include, but are not limited to, the items mentioned below:

- Performance of file queries over differing distributions of file queries
 - Query one particular node for all files
 - Query all nodes for one particular file
- As the rate of file queries increase, how is performance affected?
 - “Turn up the heat” on the system (increase request rate), and monitor the throughput
- As nodes join and leave the system, how well are the files redistributed?
- **EXTRA CREDIT:** Vary the file size: measure query latency and bandwidth over differing file sizes; does RPC have file transfer size limitations?

Grading

50% functionality

30% report

15% development of test cases & documentation

5% coding style

Test cases

As before, you will be required to develop your own test cases, and to provide documentation that explains how each of your test cases must be run, including the expected results.

Deliverables Checklist (Hardcopy due Tues April 3rd at the beginning of class)

- Electronic submission of your code using the SUBMIT tool, including a makefile we can use to compile your code
- We DO NOT want hard copies of your code itself – the electronic submission will suffice.
- Hard-copy: Single report with 3 sections:
 - Design description of your system (thorough) ~ 1 page
 - Test cases description/documentation
 - Report from your systematic study ~ 2-3 pages