Programming Assignment #1

COEN 296 Introduction to Cloud Computing
Department of Computer Engineering
Santa Clara University

Dr. Ming-Hwa Wang
Phone: (408) 525-2564
Course website:
Office Hours:
Summer Quarter 2013
Email address: mwang2@cse.scu.edu
http://www.cse.scu.edu/~mwang2/sen961
Monday 9:15-9:45pm

Due date: July 7, 2013

Content Addressable Network (CAN) (150 points)

Please implement content addressable network (CAN) for cloud computing using p2p with network proximity.

Nodes can join or leave the network at will, and are organized in a virtual *d*-dimensional Cartesian coordinate space on a *d*-torus. The coordinate space is dynamically split into *n* disjoint zones and each node owns one. Node *i* splits (in the order of 1st dimension, then the 2nd, and so on) its zone into two equal parts and hands over one subzone to the incoming newly-joined node. Nodes store their neighbors' information as their routing table. Item keys are mapped to coordinate space using uniform hash functions listed below. Your program should be able to find any item at any remote computer in a CAN. Each key is associated with a *d*-dimension vector, which represents its position in the coordinate space. If a key's position in the coordinate space falls in the zone owned by a node *i*, node *i* stores the pointer to the object related to the key.

The input to the hash function is a text string, and the output of the hash function is a *d*-dimensional Cartesian coordinate (a byte for each dimension.) The input string is divided into *d*-byte chucks (fill with zeros if the last chunk is not full), reverse the bits for all odd numbered chunks, and return the exclusive OR of all those chunks.

For example, the input is "Listen to the music", which is divided into 5 chunks in a 4D dimension: "List", "en t", "o th", "e mu", and "sic". Those 5 chunks are represented in hex as 0x4C697374, 0x656E2074, 0x6F207468, 0x65206D75, and 0x73696300. Then reverse the bits of all odd chunks and return the exclusive OR of all chunks: 0x2ECE9632 ^ 0x656E2074 ^ 0x162E04F6 ^ 0x65206D75 ^ 0x00C696CE.

The syntax of the commands are:

dimension <d>
addNode(<nodeCoordinates>)
removeNode(<nodeCoordinates>)
failedNode(<nodeCoordinates>)
insertItem(<nodeCoordinates>, <itemName>)
deleteItem(<nodeCoordinates>, <itemName>)
find(<nodeCoordinates>, <itemName>)

To make Autotest working, your program reads in commands (any of the above function calls) and executes the commands one after another. For addNode, removeNode, failedNode, insertItem, and deleteItem, display success for fail; and for find, display the optimal sequence of the nodes it traversed. If the nodes in the path are failed, then it should display alternate route.

Student Name:

SSN/ID:

Score:

Correctness and boundary condition (52%):

Whitespace and free format compliance (4%):

Compiling without warning/error (2%):

Error Handling (4%):

Alternate routing if case of failed nodes (10%);

Modular design, file/directory organizing, showing input, documentation, coding standards (24%):

Automation (4%):

Subtotal:

Late penalty (20% per day):

Special service penalty (4%):

Total score: