COP 701 (Summer 2015)

ASSIGNMENT 2

Distributed Hash table
Pastry

You are required to Implement a **Pastry DHT in C++** which should be operational over various nodes in a network. Your code should be able to store data pertaining to a peer to peer network in an efficient manner. There are more or less two basic parts in a DHT. One is WHERE to store data and another is HOW to retrieve stored data. This should be done all in peer to peer fashion.
(Do not have any master/server node or “0” marks will be awarded).

Your Pastry DHT must support a simple command-line user interface. Each instance (running on different machines or on the same machine on different ports) will run an RPC server and act as an RPC client. Although, you may skip RPC and do it with socket programming (no compulsion of using RPC server/client).

You are free to add as many commands as you want, but the ones mentioned in each phase are the bare minimum, and very importantly all output needs to be **readable** and ‘**elegantly formatted**’.

Also for this assignment, take special care of invalid inputs, **don’t let your system crash/hang on a bad/invalid input. Nobody likes that.**

The assignment has three phases. Remember your phase 3 entirely depends on how stable your implementations in phase 1 and 2 are, so work accordingly. And did we mention already? Phase 3 is very important! *Hint*: Identify the methods that are required to make phase 3 operational and work hard at stabilizing them early on.

Also , with each phase, you have to submit a **doxygen** report of your code. Do include a make.sh for this entire project. Phase wise marking will be done as always, that said, if you complete the entire assignment but Phase 1 doesn’t work fine, your marks will be deducted. Bitbucket business will be same as last time. Bitbucket commits will be checked

during demo for each phase. Give read access to following IDS:
csz148382@cse.iitd.ac.in, mcs142114@iitd.ac.in,
mcs142123@iitd.ac.in, mcs142800@cse.iitd.ac.in .

Phase 1 [Deadline: Sept 19 2015 23:55 PM]

In this phase you should have a basic pastry network operational with essentially the *node joining* scenario.

Deliverables for this phase are :

help : Provides a list of commands you have implemented and their usage details.
port <x>: Listen on this port for other instances of the program over different nodes.
create : creates the pastry henceforth will be known by this node's address and decided port.
join <x:p>: Join the pastry with x address and port p.
put <key> <value>: insert the given <key,value> pair in the pastry.
get <key>: returns the value corresponding to the key, if one was previously inserted in the node.
lset: Prints the leafset of current node
routetable: Prints the routing table of current node.
nset: Prints the neighbourhood set of current node.
dump: display all information pertaining to current node.

Configuration parameters: **l and b**. **l=4 and b=4 and m=4.**

Also, in papers and online tutorials, you will find that "ipaddress" is being hashed (SHA1, MD5) to create a node ID in the pastry. But, here you need to hash "**ipaddress:port**" as we may run several instances in one system only with different port numbers. The only hash we are allowing in this assignment is CRC-32 bit. You have to send and receive these hashes across the network in this phase and in the next phase; so it is strongly advised to first get your hands dirty in the hash domain/strings/ chars as soon as possible.

It is advised to use **pthreads** for all multithreading related work.

Phase 2 [Deadline: Sept 27 2015 23:55 PM]

An abstruse task in any DHT is to maintain the distributed data even when a node fails. That means if a node fails the data on that node shouldn't get lost. All DHTs have special ways of handling such a situation. We will assume a node goes down by informing others.

In this phase, you are required to model the *node leaving* scenario for your Pastry. The keys of the leaving node get distributed over the network.

Deliverables for this phase are :

quit: Shuts down this node, not the pastry, distributing the data.
shutdown: shuts down the entire pastry, no node should have any keys or pastry data, the programs at all the terminals should get closed on the notification.
lset <x>: Prints the leafset of node at address x
routetable <x>: Prints the routing table of node at address x.
nset <x>: Prints the neighbourhood set of node at address x.
dump <x>: display all information pertaining to node at address x.

This phase will be tough as you are required to pass a lot of info among nodes in the network. The commands we will love to see working are 1, 2 else you get nothing(not even partial marks) from this phase as far as the marks are concerned. Rest of the commands are mandatory too, but we are warning beforehand that they will consume a lot of your brain and time.

Bonus Modules: (Awarded only if the primary spec is met)

finger: a list of addresses of all nodes on this pastry.
dumpall: All information of all the nodes.

Phase 3 [Deadline: Oct 4 2015 23:55 PM] [Updated final deadline]

Now this is where things get interesting.

While you were implementing the DHT in the campus here, a hypothetical country Utopia was working on its satellite launch programme. They wanted to save their satellite launch codes for all 365 days of the year in a secure facility. To achieve that, they very astutely bought a huge server, used a lot of encryption algorithms, secured the network, hired the best hackers and stored all **365** codes on that machine. (In case you are wondering, Utopians saved all their codes in

MD5, that said, satellite launch code KILLTHEM would be saved as 997544822d65c89639c86a3de0caf39c5c7ee383).

Then they got to know about **rainbow** tables, and there was pandemonium all around. Anyone could use a rainbow table to decrypt the hash, sometimes the rainbow tables worked, and sometimes they didn't. All hashes in one place and someone can try and decrypt them any time - this was more than a nightmare for Utopia. (Try this: https://crackstation.net/)

Now they have hired you to tackle the matter at hand. **Here** is what you do: Given a satellite launch code *C* for a day *D*, you MD5 it to *H*, then break *H* into ***n* parts** and distribute the parts (hint: these parts will be the *values* in the hash table) on ***N*** nodes of Pastry. Now when someone wants to launch a missile they enter the code *C* into any of the *N* nodes for the day *D*, and you have to verify if the code is correct, i.e, the code is stored as broken parts into the system.

We will enter the satellite launch mode by using a switch --sat while running your program. In normal mode, only the DHT commands should be supported. In sat mode, DHT + satellite launch commands should be supported. So you would be using join, create commands in satellite launch mode too.

Extend your command line app to also support these:

> store <DAY> <SAT-CODE>: Stores the launch code as parts on the network. DAY ranges from 1-365 and SAT-CODE is any valid combination of ascii characters. If the launch code is already stored, should display warning first and if allowed to proceed, over-write it.
> verify <DAY> <SAT-CODE>: Verify if this SAT-CODE is actually valid for the given day. If the launch code is not stored on the network, should report error.

Save the hash of the code in your network and not the code, else no marks will be awarded.

BONUS(Awarded only if basic spec met):

getall D1-D2: get all the hashes for all the days from D1 to D2(both inclusive) from your network. For example: getall 1-365 should return all the hashes day wise sorted by day number. If some day's hash code is missing should display MISSING.

We can test on ***N*** machines - any number we want, we can destroy any machine we want using the quit command, and we can add more machines using the join command. The hashes shouldn't get lost.
The question that remains is what should be value for ***n parts***? Choose whatever you find appropriate. **n** should be more than equal to **three**. The higher the number, the more the messages on the network, more the keys on the network, more complex everything will become. Advice: Implement your DHT correctly, and your life will be easy.

Pre-requisite is stability. It should work for **three** parts correctly. And if you can show us that your system works with a large n and large N, we can negotiate on a bonus!

References:

Pastry Original Paper: http://research.microsoft.com/en-us/um/people/antr/PAST/pastry.pdf
Pastry Video: https://www.youtube.com/watch?v=iPVAOh_slsU
Another overview: http://www.ccsenet.org/journal/index.php/cis/article/viewFile/4282/3729
Pastry Lecture Slides: http://www.cse.iitd.ac.in/~srsarangi/csl860/docs/pastry-lec.pdf
Short Overview: http://www.freepastry.org/PAST/overview.pdf
P2P N/W Video: https://www.youtube.com/watch?v=kXyVqk3EbwE