Implementing a Circular DHT

An application of Concurrency to a DB distributed over a large community of peers

In this assignment we will consider how to implement a simple database in a P2P network. Our database will be designed as a *circular distributed hash table* (DHT). The database will only contain (*key*, *value*) pairs.

Building such a database is straightforward with a client-server architecture that stores all the (*key*, *value*) pairs in one central server. Instead, we are going to consider how to build a *distributed*, PSP version of this database that will store the pairs over millions of peers. In the P2P system, each peer will only hold a small subset of the totality of the pairs. We will allow any peer to query the distributed database with a particular key. The distributed database will then locate the peers that have the corresponding (*key*, *value*) pairs and return the pairs to the querying peer. We will also allow to insert new pairs into the database.

The circular DHT will be designed as follows: each peer has an identifier assigned to it; this identifier is an integer in the range $[0, 2^n - 1]$, for some fixed n. Each peer only keeps track of its immediate sucessor and immediate predecessor (module 2^n).

Let f be some *hash function* such that $f(key) \in [0, 2^n - 1]$ — i.e., the range(f) is contained in $[0, 2^n - 1]$ for the same n. In order to store (*key*, *value*), compute $k = f(key)$ and place the pair in the peer whose number $i = \min\{j : k \leq j\}$. If no peer exists whose number is at least k, then "wrap around," and let $i = \min\{j\}$. Thus, we think of the peer ids as being counted modulo 2^n, and hence the name "circular DHT." Once the peer id i is computed, the pair (*key*, *value*) is sent to i for storage.

Q1 Consider the following peer metric. Since $i, k \in [0, 2^n - 1]$, both i, k can be represented with n-bit binary numbers:

$$(i)_b = i_{n-1} i_{n-2} \ldots i_0$$
$$(k)_b = k_{n-1} k_{n-2} \ldots k_0$$

The following is called the "XOR distance" between i and k:

$$d_{\text{xor}}(k, i) = \sum_{j=0}^{n-1} |k_j - i_j| 2^j.$$

Describe how d_{xor} can be used as a metric for assigning (*key*, *value*) pairs to peers; how does it compare to the circular metric described above?

Q2 Back to the circular DHT: explain how a peer can retrieve a given (*key*, *value*) pair. In order to retrieve such a pair, a peer must first find it — that is, it must find the id i of a peer that has (*key*, *value*) stored. More precisely, the ids of the peers that have it stored; also note, that the pairs are searched by key only, and hence there may be several pairs,

$(key, value_1), (key, value_2), \ldots$, in which case the peer ids for *all* of them must be returned. If the same pair $(key, value)$ is stored in several peers, it is enough, for now, to find the id of just one of them.

Explain how a peer can retrieve the ids. One way to do this is for a given peer to send a message to all the other peers asking if they have a pair $(key, value)$ corresponding to $k = f(key)$. But this requires every peer to have the list of ids of all other peers — this does *not* scale well. Give a solution that *scales well*; your solution should examine the tradoff between how many neighbors a peer has to track *and* the number of messages a circular DHT needs to send in order to resolve a single query.

In summary, you start with a circular DHT, and propose an expanded *structured overlay.*

Q3 Explain how to implement "peer churn," that is, how to add and remove peers. Assume that there is a peer, known to the "outside," with id i_0, that handles requests to join the DHT. Again, the solution has to scale well — it is impractical to assume that any peer has the ids of all the peers in the DHT.

Q4 Implement a simulation of the P2P system with **Python Threads** — where each peer is simulated with one thread. The program, call it `p2p`, will be invoked from the command line and it will await the following commands:

```
addp 537
delp 537

find 537 920
insert 537 (920,some value)
```

where the first command adds a peer with id 537; if it already exists, the user is informed, and nothing happens. The second command deletes peer with id 537; if it does not exist, the user is informed of that. The third command asks peer 537 to locate the peers with pairs $(key, value)$ where $key = 920$. The output should be all the peers and corresponding pairs as in:

```
45:(920,some value)
79:(920,some other value)
12:(920,yet another value)
```

The fourth command delegates to peer 537 the insertion of the pair $(920, \textit{some value})$ into the P2P database.

Note that we assume here that $k = key$, i.e., the hash function f is the identity on $\mathbb{N}$.

Document well your program.