LAB #2 on DISTRIBUTED HASH TABLES

Advancing our DHT implementation in Python

Lorenzo Ghiro

lorenzo.ghiro@unitn.it

Where we start from

- Previous LAB we started implementing a basic DHT by implementing:
  - hash-based function to compute keys and nodeIDs
  - clockwise distance metric
  - Node class with successor and predecessor to be setup so to achieve a "DHT ring" aka "double linked-list"
  - Recursive JOIN for placing new nodes dynamically in the DHT
  - Recursive STORE and LOOKUP methods

What we (YOU!) do today

1. JOIN/LEAVE protocol:
    - JOIN -> not just placing nodes, but also getting initial items from predecessor
    - LEAVE -> graceful goodbye, passing own keys to new responsible node
2. FINGER TABLE! To improve efficiency
    - Learn how to initialize/update a finger table
    - then reimplement efficiently Lookup, this time based on Finger table
3. Make everything more realistic… with Flask :)

JOIN/LEAVE protocol

- JOIN: "the predecessor of JOINER has some keys that now should belong to JOINER"
  - Identify these keys and store them in JOINER
  - Then remove them from predecessor
- LEAVE: "all keys of LEAVER should go to its predecessor"
  - Store all LEAVER items at predecessor
  - Then leave by informing pred and succ to rewire so to close the gap

Towards the Finger Table

- Our JOIN/LEAVE will make the insertion and removal of nodes independent of the insertion and removal of data! :)
- However, the performance is terrible
  - `O(n)` with an expected performance of `n/2`
  - Consider a DHT with 1000 nodes & the need of setting up a TCP/IP connection for each request forwarding... traversing n/2=500 nodes can be quite slow! (e.g. `20ms x 500 = 10s`)
- Add Finger Table to to access O(log n) performance!

Finger Table idea

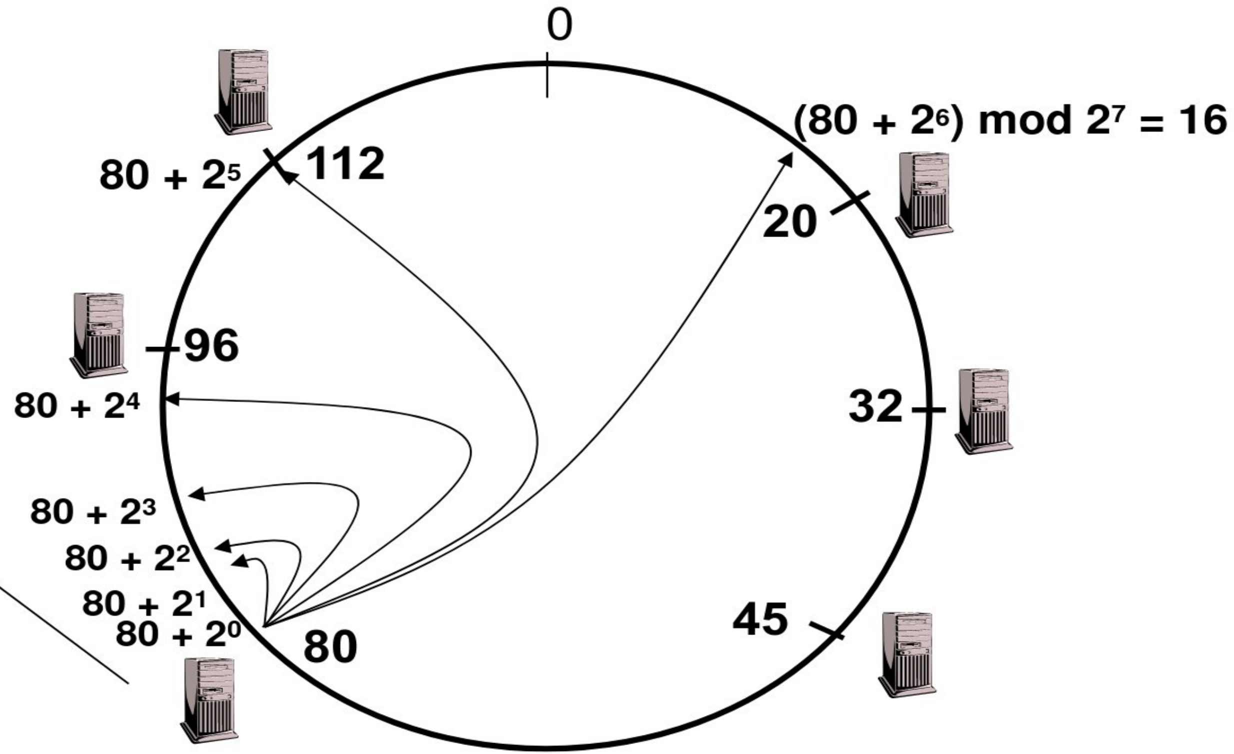
- Instead of storing a pointer to the succ node, each node stores a *“finger table”* containing the addresses of **k** nodes
- The distance between the current node's ID and the IDs of the nodes in the finger table increases *exponentially*
- So each node on the path to a given key is logarithmically closer than the last ⇨ O(log n) nodes traversed worst-case :)
- Updating a finger table requires that a node address is found for each of the **k** slots in the table…

Lookup with Finger Table

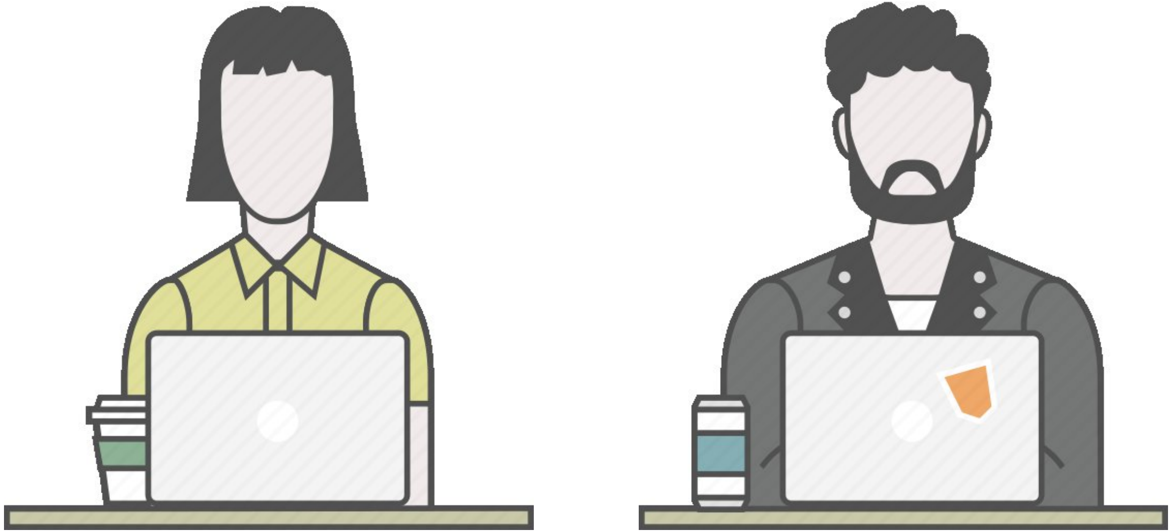
- For any slot `x`, where `x` is `1` to `k`, `finger[x]` is determined by taking the current node's ID and looking up the node responsible for the key $(id + 2^{(x-1)}) \% 2^k$
- When doing lookups, you now have k nodes to choose from at each hop, instead of only one at each
- [**SHORTCUT**] When a node receive a Lookup request, it will now forward it to the node in his finger-table which has the shortest distance to the key

Finger Table Visualization

i	$ft[i]$
0	96
1	96
2	96
3	96
4	96
5	112
6	20



Coding Time!



Errata Corrige and hints

- OK, now you are already working on implementing JOIN/LEAVE protocol, the finger table and fingerLookup but… wait few more minutes please!
- Some corrections from the last time and further recommendations for you

compute_key(string, bitlength=k)

- We wanted a function that
  1. Gets the binary digest computed by some hash function (sha256)
  2. Extract only the k rightmost bits
  3. Returns the int number given by considering these k bits as an uint
- check out bitstring
  - pure Python module designed to help make the creation and analysis of binary data as simple and natural as possible

```
digest = sha256(bytes(string, 'utf-8')).hexdigest()
bindigest = BitArray(hex=digest).bin
subbin = bindigest[:bitlength]
return BitArray(bin=subbin).uint
```

Further recommendations

- Separate the main file, responsible for interacting with the DHT, from the DHT code
  - e.g., LAB solution consists of 2 files, **main.py** and **advancedDHT.py**
- Define a **findNode(startnode, key)** function which...
  - "Recursively find the node whose ID is the greatest but smaller ID in the DHT compared to key"
  - it embeds the JOIN criterion of last lab, reused in lookup and store
- Define an **update()** method for the node class
  - compute there the finger table

Further recommendations

- At some point it's a good idea to inject more nodes and contents to test the finger table initialization
  - printing a large DHT becomes almost impossible... use [tabulate]to print the DHT in html and open it in a browser (NB: `tablefmt = html`)
  - Draw the DHT as circular graph with networkx!!!

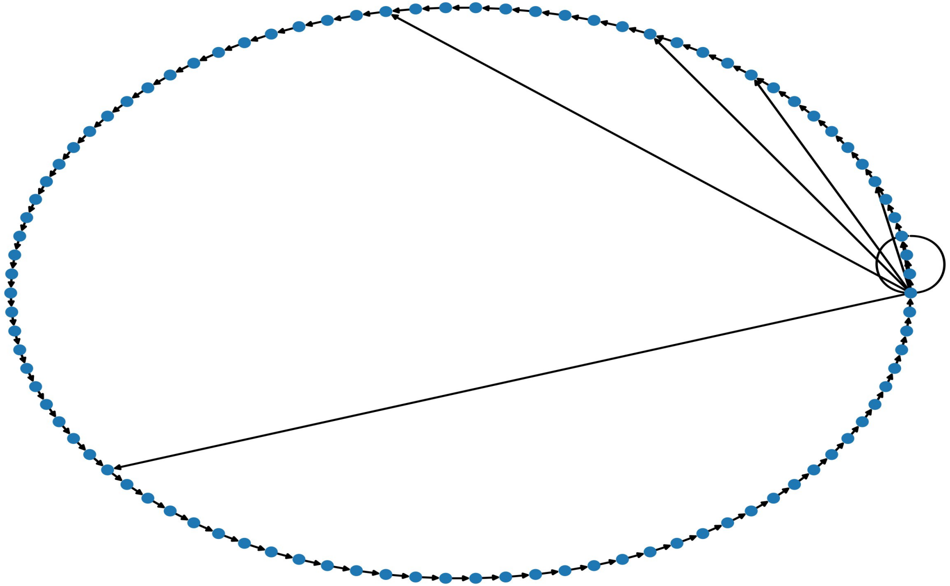
```
nx.draw(G, pos=nx.circular_layout(G), with_labels=False, node_size=0.1)
```

  - For some node, draw also the finger edges, check they make "exponential jumps"

Wishful output

55645056	87597561	224879733	286887193	290831257	363956850	413061507	421328900	424567226	434941935	514995864	564425828
(60892216, 'vestibulum')	(89185933, 'ex,')	(240372897, 'hendrerit')		(299666858, 'potenti.')	(371134046, 'una')			(434878662, 'posuere')	(446850923, 'euismod,')		(576640598, 'di')
(68666679, 'ipsum')	(102432683, 'pharetra')	(245057219, 'Aliquam')		(312425105, 'dolor.')	(401783973, 'dolor,')				(453383841, 'tellus')		(584931784, 'Nam')
	(131996937, 'Nunc')	(255750582, 'scelerisque')		(332192578, 'justo')	(403017947, 'molestie')				(457465400, 'congue')		
	(162821717, 'neque')			(359755483, 'at,')					(461341798, 'Lorem')		
	(201142793, 'purus')			(361487215, 'porta')					(500151768, 'volutpat.')		
	(210510558, 'mattis.')										
	(219018596, 'mauris,')										

592618713	601978482	607141728	721177868	750060701	762231136	781455581	786228284	892917589	933381702	99586
		(621792469, 'che')	(735749037, 'Praesent')	(756675235, 'Suspendisse')	(762550102, 'eu,')		(790556497, 'fringilla')	(920038354, 'convallis')	(951161061, 'mauris')	(100227 'dignissi
		(657380466, 'feugiat')					(791211055, 'pulvinar,')	(922508243, 'orci')	(989530358, 'vel,')	(101036 'Donec')
		(668664828, 'lectus.')					(794352347, 'erat.')			
		(678854987, 'Integer')					(807465578, 'Etiam')			
		(716836176, 'sapien.')					(857253085, 'Sed')			
		(717623776, 'dapibus')					(872463177, 'lorem')			
							(874058052, 'pulvinar')			



Further recommendations

- In the proposed solution for the main file

1. (initDHT) -> printDHT -> Create new node and JOIN -> printDHT -> make one node LEAVE -> printDHT; so you check content passing works properly
2. Add up to 100 nodes and 1000 items -> printDHThtml -> drawCircularGraph
   NB: BITLENGTH >> 8 or may have hash conflicts!!!
      - Check graph is truly circular ^ check from html that content is fairly distributed
3. Compute finger table for each node -> drawGraphWithFingerLinks4someNode
4. Issue many fingerLookups and standard lookups
   - keep track of recursionLevel (how many forwardings)
   - compare the recursionLevel for same key looked up with/without finger-table… who performs better???

Wishful output

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

Comparison of recursion level for same key looked-up at the same node first WITH then WITHOUT finger-table

content	fingerSteps	standardSteps
nel	0	17
mezzo	4	77
del	2	65
cammin	3	23
di	3	36
nostra	4	91
vita	0	6
mi	1	19
ritrovai	1	93
per	4	89
una	1	30
selva	3	87
oscura	3	24
che	1	39
la	4	91
diritta	1	70
via	0	57
era	0	57
smarrita	1	93

Questions?