Tutorial
Distributed Systems (IN2259)
Pezhman Nasirifard
WS 2019/20

Issued on: 18.12.2019
Due on: 09/10.01.2020

Sample Solution: Exercises on Peer-To-Peer Systems (Part 1)

Exercise 1 - Chord

Let C be the Chord ring given in Figure 1.1. The network contains eight peers (blue points), each of which is mapped to a node $N_i \in C$ by a base hash function, e.g., SHA-1, with bit-string length $m = 5$. Key k is assigned to the first peer whose identifier is equal to or follows k within C. For instance, in Figure 1.1, key K_8 is assigned to the peer at node N_{10}. The *successor* function receives as input an integer $i \in \{0, \ldots, 2^m - 1\}$ and returns the closest peer node N_j such that $j \geq i$.

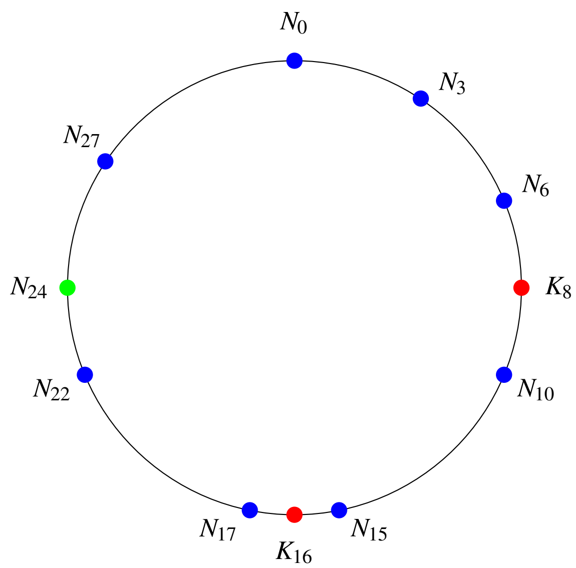


Figure 1.1: Chord ring.

(a) Suppose that every peer node stores only a reference to its successor node, e.g. at N_3, $successor(3) = N_6$. Give an asymptotic upper bound on the number of hops you would need to look for a key k in the worst case. Which operations do you need to perform when a peer joins/leaves the ring? What if we also reference to a predecessor node?

Solution:
We basically have a circular single linked list. In the worst case we need to traverse it entirely $\implies O(n)$.
When a peer joins, it contacts a random peer, and so we need to perform a lookup to find its position in the ring. Then, we need to update successor pointers. Lookup needs $O(n)$ hops in both cases. Hence, insertion is dominated by the lookup. The predecessor pointer helps performing deletion updates in constant time instead of linear when a node willingly wants to leave the ring.

(b) From now on, consider the standard Chord implementation where peer nodes store a finger table. Fill in the finger table of the peer at node N_3 in Figure 1.2. Will N_3 have a reference to all the other peers?

Solution: No, N_3 will not have a reference to all other peers, e.g., N_{17}

Peer ID	Successor
$(3+2^0=4)mod32$	N_6
$(3+2^1=5)mod32$	N_6
$(3+2^2=7)mod32$	N_{10}
$(3+2^3=11)mod32$	N_{15}
$(3+2^4=19)mod32$	N_{22}

Figure 1.2: **Solution** of subtask b: Finger table for node N_3.

(c) By looking at the finger table you just filled in, can you directly determine the peer responsible for an arbitrary key $k \in C$? Show how you would search for K_{16} from N_3.

Solution:
No. It doesn't contain enough information.
From N_3 we should move to the finger entry with highest ID that precedes K_{16}, which is N_{15}. Then, from N_{15} we are lucky and can directly determine the peer responsible for K_{16}, which is N_{17}.

(d) Suppose a new peer wants to join the network, and it is mapped to node N_{24} (in green). Show all the changes that are necessary to perform such an operation. How much do these changes cost?

Solution:
Update the fourth finger of N_{15}: $15+2^3=23 \Longrightarrow N_{24}$, and the first and second fingers of N_{22}: $22+2^0=23 \Longrightarrow N_{24}$, $22+2^1=24 \Longrightarrow N_{24}$.
At a node v, the finger table have $O(log(n))$ entries and for each finger i we need to look for $v+2^{i-1}$ to see if something changed. Total cost $O(log^2(n))$ per peer node.

(e) Now suppose that the peer at node N_0 leaves C. Write down the updates needed and discuss the cost of this operation.

Solution:
Update the fifth finger of N_{15}: $15+2^4=31 \Longrightarrow N_3$, the fourth finger of N_{22}: $22+2^3=30 \Longrightarrow N_3$, and the first, second and third fingers of N_{27}: $27+2^0=28 \Longrightarrow N_3$, $27+2^1=29 \Longrightarrow N_3$, $27+2^2=31 \Longrightarrow N_3$.
At a node v, the finger table have $O(log(n))$ entries and for each finger i we need to look for $v+2^{i-1}$ to see if something changed. Total cost $O(log^2(n))$ per peer node.

(f) When peers fail, it is possible that a peer does not know its new successor anymore, and this could lead to an incorrect lookup. How would you approach this problem? Assume that peer failures occur independently with probability p.

Solution:
To avoid this situation, peers maintain a successor list of size r, which contains the peer's first r successors. When the successor peer does not respond, the peer simply contacts the next peer on its successor list. Peer failures occur independently with probability p; therefore, the probability that every peer on the successor list will fail is p^r. Increasing r makes the system more robust. Thus, by tuning this parameter, any degree of robustness with good reliability and fault resiliency may be achieved.

Exercise 2 - Content Addressable Network

For simplicity, in this exercise we see a CAN as a d-dimensional unit cube $[0,1]^d$ instead of a d-dimensional torus. However, remember that, in a real CAN, the sides should wrap around.

(a) Suppose peer P lying in a d-dimensional CAN responsible for virtual coordinate zone $(x_1, x_2, \ldots, x_d)$ such that $x_i \in [x_{min}^{(i)}, x_{max}^{(i)}]$ for each dimension $i \in \{1, \ldots, d\}$, where min, max refer to the minimum and maximum interval endpoint, respectively. For example, in Figure 2.3, peer P_1 would be responsible for all the points (x_1, x_2) whose $x_1 \in [0.25, 0.375]$ and $x_2 \in [0.625, 0.75]$. Peer P needs to search for a key k mapped to the point $\mathbf{y} = (y_1, y_2, \ldots, y_d)$ of the CAN. Specify a greedy algorithm that forwards the request to the nearest neighbour peer to $\mathbf{y}$ in pseudo-code. What is the time complexity of your algorithm assuming a d dimensional space that is partitioned into n equal zones?

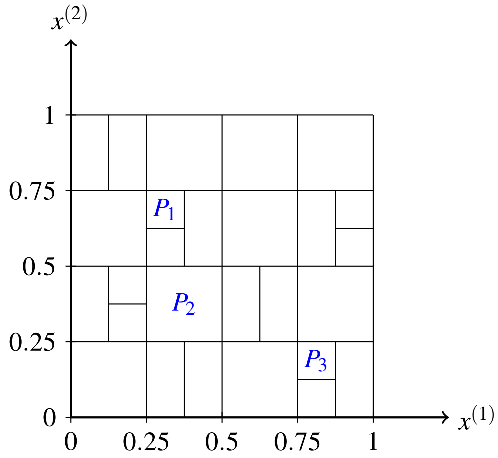


Figure 2.3: 2D CAN

Solution:

GREEDY-FORWARDING(y):

if $y \in myZone$ then
  return
end-if
$nearestNeighbour := NULL$
$minDistance := \infty$
for each $n \in$ neighbors
  // For each dimension i
  for $i := 1$ to d do
    // Get middle point z
    $z_i := \frac{(n.x^{(i)}_{min} + n.x^{(i)}_{max})}{2}$
  end-do
  if $\|z - y\| < minDistance$ then
    $nearestNeighbour := n$
    $minDistance := \|z - y\|$
  end-if
end-do
return $nearestNeighbour$

For a d dimensional space that is partitioned into n equal zones, the worst-case routing path length is $d \cdot (n^{\frac{1}{d}} - 1)$ since we must do in each dimension d at most $n^{\frac{1}{d}} - 1$ hops to route from one corner of the cube to the opposite corner. The complexity of the algorithm is therefore $\in O(dn^{\frac{1}{d}})$

(b) Draw the path determined by the search algorithm from Part (a) in the CAN given in Figure 2.3 to route from $P_1([0.25, 0.375], [0.625, 0.75])$ to $\mathbf{x} = (0.4, 0.3)$ and $\mathbf{y} = (0.8, 0.2)$. Who are the peers responsible for these two points?

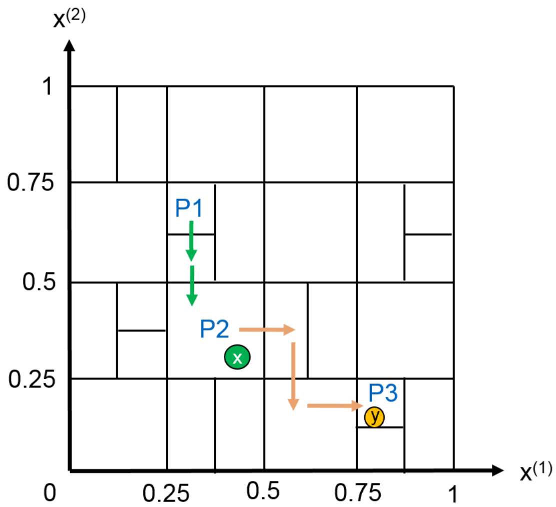
Solution:

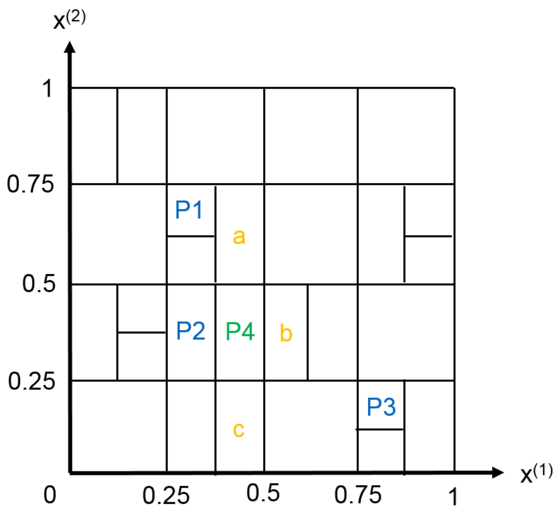
The green path reaches $\mathbf{x}$. The peer responsible is P_2. To reach $\mathbf{y}$, continue with the orange arrows up to P_3.

(c) A new peer P_4 wants to join the network of Figure 2.3, and it is mapped to coordinate point $(0.4, 0.4)$. How would you modify the CAN? Assume that we split a square cell vertically, rectangular cells are split horizontally. Which peers should update their neighboring set?

Solution:

The CAN will be modified as in the figure below. P_2, a, b and c, which are P_4's neighbors, should update their neighboring set.


x(2)
1
0.75
0.5
0.25
0
0.25
0.5
0.75
1
x(1)
P1
P2
P3
x
y


x(2)
1
0.75
0.5
0.25
0
0.25
0.5
0.75
1
x(1)
P1
a
P2
P4
b
P3
c