HW1 Solutions: CS425 FA16

1. (Solution and Grading by: Hongwei Wang)

function map(line) {
    userA, userB = parse(line)
    emit(key=sort(userA, userB), value="") // ie: key is lexicographical sort of userA, userB
}

function reduce(key, values) {
    count = 0
    while (values.hasNext()) {
        count++
    }
    if (count == 2) {
        emit(key=key, value="")
    }
}

function parse(line) {
    // implementation
}

2. (Solution and Grading by: Bo Teng). Updated 9/25

A chain of two MapReduce jobs is required.

Map1(line):
For every input line (a,b) do:
   Emit (a: (out, b)) and (b: (in, a))
Done

Reduce1(key, [value1, value2, ...])
For every key a:
   For i from 0 to value.length do:
      If value[i] starts with “out” then:
         // toNode is the 2nd tuple in the value
         Emit ((key, toNode): exists)
      Else if value[i] starts with “in” then:
         // toNode is the 2nd tuple in the value
         Emit((toNode, key): exists)
   End if
   If value[i] starts with “in” then:
Find another value value[k] that starts with “out” such that (k>i and value[i] != value[k])
Emit((value[k], value[i]): needed, sort(value[i], key, value[k]))
End if
Done
End if
Done

Map2(key, value)
// Identity
Emit(key, Value)

Reduce2(key, [value1, value2, ...])
For every key(a,b)
    For i from 0 to value.length do:
        If value[i] starts with “needed” then:
            Find another value[k] such that value[k] starts with “exists”
            Emit tuple 2 in value[i] which is a triple as key, 1 as value
        End if
    Done
Done

Map3(key, value)
// Identity
Emit(key, Value)

Reduce3(key, [value1, value2, ...])
Emit the key as unique triple

3. (Solution and Grading by: Bo Teng)

The solution below chains two map-reduces.

Map1:
    Input: (a, b) tuples where a follows b
    Output: (a, (‘follows’, b)), (b,(’is followed by’, a))
Reduce1:
    Input: (a, all value(a)) where value(a) is either of the form (‘follows’,b) or (‘is followed by’, b); all value(a) is the collection of all values associated with a

numFollowers = 0  //# of followers of a
numFollowed = 0    //# of people followed by a
for val in all value(a):
    if val[0] == ‘follows’:

numFollowed += 1

else:
    numFollowers += 1
if numFollowers > 1 million: // a has > 1 million followers,
    for val in all value(a): // satisfies condition (1)
        // if ‘a’ has more than 1 million followers,
        // everyone who follows ‘a’ satisfies condition (3)
        if val[0]=='is followed by':
            output((val[1], 'follows someone with million followers'))
    if numFollowed >= 10: // a also satisfies condition (2)
        output((a,'has million followers and follows at least 10 people'))

Map2: (Identity)
Input: (a,s) outputs from Reduce1 where s is either ‘is followed by someone with million
followers’ or ‘has million followers and follows at least 10 people’
Output: same as above

Reduce2:
Input: (a, S) where S is the collection of all values associated with a
// Output: b such that b is ‘similar’ to Hillary Clinton (delete this line)

    if ‘follows someone with million followers’ in S and ‘has million followers and follows at
least 10 people’ in S:
        output(a)

4. (Solution and Grading by: Si Liu)
   a. No, the algorithm is not complete. In the worst case, all N processes may end up
      selecting the same k processes to heartbeat from, say p1 through pk. This means pk+1 through
      pN do not have anyone receiving heartbeats from them, and their failure(s) will not be detected.
   b. No, the algorithm doesn’t satisfy accuracy. No heartbeating protocol is accurate, because
      one cannot distinguish between a failed process and a process whose heartbeats are being
      missed.

5. (Solution and Grading by: Si Liu)
Each process must heartbeat to N/3 processes. It doesn’t matter how these processes are
selected. If a given process pi were to fail, it cannot be that all its heartbeat targets have also
failed, otherwise we would have (N/3+1) failures. Thus at least one heartbeat target would be
alive, and would time out waiting for pi’s next heartbeat, and mark pi as failed.

6. (Solution and Grading by: Sili Hui) The point guard is right. In gossip with partial membership
lists, the probability of a process pi picking another process pj as gossip target (for a given
message) is = Probability (pi has pj in its membership list) X Probability (pi picks pj for the
gossip) = k/N X m/k = m/N. This is the same probability as in the full membership case. Thus the two protocols behave the same.

7. (Solution and Grading by: Shiv Verma)
For the ith finger table entry at node n, instead of selecting the first peer beyond n+2^i, select among all peers between n+2^i and n+2^i(i+1), that peer which is closest in round trip distance. This ensures that as a query progresses, initially hops will be short in round trip distance (but long along the ring), and later they will be longer in round trip distance (and short along the ring).

Memory is clearly O(log(N)), since the number of finger table entries is the same as Chord.

Routing distance is no worse than Chord, since each finger table entry is at least as far (along the ring) as the corresponding Chord finger table entry (Chord selects the first beyond 2^i for fingers). Thus a query progresses along the ring no slower than a Chord route, thus it is O(log(N)).

8. (Solution and Grading by: Shiv Verma)

A) Have a Chord-like scheme where machines (and files) hash to a position on a ring, which is used to store/route files as well as migrate files during churn. The difference from Chord will be that we won’t be using finger tables as all peers know all other peers.

B) Let’s say there is only one multicast/broadcast per join/leave/fail.
   100% churn rate implies 1M casts per hour.
   Each multicast will have 1M messages.
   So, we get overall 1M X 1M = 1T messages per hour.
   The per-peer bandwidth usage will be 1T / 1M = 1M messages per hour.
   Even for relatively large message sizes, 1M messages per hour should be tenable.

C) IPv6 address -> 16 Bytes
   Port number -> 2 Bytes
   Gossip count -> 4 Bytes
   Timestamp -> 8 Bytes
   (assuming 64-bit unix timestamp. Because the 32-bit version only works till 2038\(^1\) and our distributed systems are built to last).
   All that adds up to 30 Bytes.
   2GB / 30 Bytes = 66.6 Million

\(^1\) https://en.wikipedia.org/wiki/Year_2038_problem
So, that’s how large the P2P system can be. Answers in the ballpark will also be accepted.

9. (Solution and Grading by: Sili Hui)
   a. 12, all processes will receive the msg in ring
   b. 2, msg with 2 ttl will be received by all nodes
   c. 2, send from any node

10. (Solution and Grading by: Hongwei Wang)
   c. 2014