HW1 Solutions: CS425 FA22

1. (Solution and Grading by: Fangqi Han.)
   BitTorrent downloads the local rarest shard first. The spacecraft is directly connected to all planets, implying they are all its neighbor. Each shard is replicated the following number of times: A: 6 times, B: 3 times, C: 6 times, D: 7 times, E: 6 times, F: 2 times.
   a. F is the rarest shard among neighbors and thus will be fetched first.
   b. F->B->A->C->E->D.
   c. Any set that has shard F but not shard B is correct, as the tiebreaker will give B the advantage.

2. (Solution and Grading by: Samarth Aggarwal.)
   We use shorthand Unique_Person_Name = name, start_time = start, end_time = end. Remember -- You are writing code for a Map function and Reduce function (not Map task and Reduce task!).
   MR1 // Converts D1 data into Key-value pairs
   - M1 reads from D1, for each input line:
     Emit (key=name, value=(location, start, end))
   - R1 inputs (key=name, value=list of (location, start, end)):
     Emit (key=name, value=list of (location, start, end))

   MR2 // Converts D2 data into Key-value pairs
   - M2 reads from D2, for each input line:
     Emit (key=name, value = “positive”)
   - R2 inputs (key=name, value=list of “positive”) //one “positive” for each key
     Emit (key=name, value=”positive”)

   MR3 // Merges D1’s and D2’s KV pairs
   - M3 reads from MR1’s and MR2’s outputs, inputs (key=name, value=value):
     // value could be “positive” or a list of (location, start, end)
     Emit (key=name, value=value)
   - R3 inputs (key=name, value=list of values from M3)
     // values: (list of (location, start, end), “positive”), if this key was tested positive, otherwise list of (location, start, end)
     Emit (key=name, value=value)

   MR4 // Checks for positivity
   - M4 reads MR3’s outputs, inputs (key=name, value=value)
     If value contains “positive”, then //this key was tested positive
     For each triple(location, start, end) in value:
     Emit (key=location, value=(“positive”,(start, end, name)))
     else
     For each triple(location, start, end) in value:
     Emit (key=location, value=(“testcase”,(start, end, name)))

- R4: inputs (key=location, value=list of (state, (start, end, name))
  // state could be “positive” or “testcase”
  Let L1 = list of entries with state = “positive”
  Let L2 = list of entries with state = “testcase”
  Let testSet = empty set
  For each entry e1 in L1:
    For each entry e2 in L2:
      If e1’s and e2’s time interval overlap:
        testSet.add(e2’s name) // e2 is from testcase list
  For name in testSet:
    Emit(key=name, value=null)
  // Note that there might be duplicate names across Reduce tasks!

MR5 // Outputs final results

- M5 reads from MR4’s outputs. Inputs (key=name, value=null)
- R5: inputs (key=name, value=list of null)
  Emit (key=name, value=”needs-to-be-tested”).

Note that the above solution also includes the names of known infected individuals (from D2). If you’d like to eliminate these, then you can add an additional MR.

3. (Solution and Grading by: Xiaojuan Ma.)

MR1:

M1: for each input line (contains (a, b) // a follows b):
  Emit (key=a, value=(b , empty_set))
  Emit (key=b, value=(empty_set, a))

//value[0] is followed by the key; value[1] follows the key

R1: input (key= account, value=list of (following, follower)):
  followingSet = empty
  followerSet = empty
  For each v = (a, b) in value:
    followingSet = followingSet.union({a})
    followerSet = followerSet.union({b})

  If |followerSet| >= 4 million and |followingSet| >= 500:
    Emit (key, #)

// this key(account) has at least 4M followers and follows 500: satisfies the first 2 conditions

  If |followerSet| >= 2 million:
    For each u in followerSet:
      Emit (u, *)

// each account u satisfies the third condition since the key account it follows has at least 2 million followers

```
MR2
M2: input (key=k, value=v) // k, v is the output of R1
        Emit (k, v)
R2: input (key=k, value=list of v)
        If # and * are in value:  // indicate that all three conditions are satisfied
            Emit(k, -)
```

4. (Solution and Grading by: Taksh Soni.)

1st Mapper/Reducer

```
Map1 (key=null, value=(a, b)) // where key follows value
{
    Emit (a, b)
}

Reducer1 (key=a, value=set of users that a follows)
{
        If value contains packers and value does not contain ChicagoBears:
            For each item in value:
                Emit (a, item)
}
```

2nd Mapper/Reducer

```
Map2 (key=a, value=b)
{
        Emit (key=lexicographic_sort(a,b), value=1)
}

Reducer2 (key=(a,b), value=set V){
        If |V| ==2: // if number of entries in value is 2
            Emit (a,b)
}
```

5. (Solution and Grading by: Ritwik Deshpande.)
    a. AWS EC2 is a service that allows the deployment of code on virtual machines called EC2 instances, which can be heavy at times but are good for computationally intense jobs or jobs that need to run continuously. AWS EC2 web service is known as **Infrastructure as a Service**. In AWS Lambda the program

code(Lambda function) are lighter-weight functions (rather than entire VMs), which are generally associated with an event that triggers them. AWS Lambda is referred to as serverless as AWS implicitly manages the server configurations to run the program code. AWS Lambda is known as **Function as a Service**.

b. AWS lambda is recommended for applications that are designed to run non-contiguously, with variable processing time, **generally triggered by an event or polling based applications**.
   i. **Large Scale Batch Processing of Data**: In enterprise applications where GBs of Data is received from 3rd party vendors to reconcile the internal data, it would be prudent to use the AWS lambda event-driven architecture to satisfy the variable computational needs, along with huge savings on cost with by invoking the AWS instances only when required.

   ii. **Document Conversion:** AWS lambda is an apt tool for rapid document conversion. Storing static documents can take up a lot of space and is inconvenient if the content of the document changes regularly. All necessary activities such as getting the document, formatting and converting it, and serving it for display or download may be executed on the go using AWS lambda.
   iii. (Other reasonable solutions also accepted)

c. AWS spot instances are **unused instances of EC2** at a discounted price. However the price of AWS spot instances may vary based on the market-availability, while AWS lambda costing is usually fixed (market-independent). For computation the core functionality remains IaaS. While spot instances are on-demand services comparable to AWS lambda the main difference being AWS lambda implicitly manages the server and hardware configurations. Spot instances may suffer more from interruptions if the unused EC2 instances are required again by AWS, whereas AWS lambda is designed to run event-driven stateless processes seamlessly at different periods of time.

6. (Solution and Grading by: Kshitij Phulare.)
   We will calculate the gossip probability of a given processing Pi picking a given process Pj (for the entire message). If these probabilities are identical for two protocols, the two protocols will behave identically.
   Case 1 (Full Membership): A Process with full membership list picks m randomly selected members (for a given message) to gossip with.
   Probability(Pi picks Pj) = m/N

   Case 2 (Partial Membership): A Process with partial membership list (k) picks m randomly selected members per round to gossip with.

Probability(Pi picks Pj) = Probability (Pj is in Pi's membership list) * Probability(Pi picks Pj) = k/N * m/k = m/N

Since the two probabilities are identical, the two protocols behave identically.

Alternately, if you treated m as the number of gossip targets per round in both protocols, the same above equations show that the probability that Pi picks Pj in a given round is identical in both protocols.

7. (Solution and Grading by: Pengyu Lu.)
Variant of Chord: For the i-th finger table entry at node n, instead of selecting the first peer with id>=$n+2^i$(mod 2^m), select one peer among all peers that have $n+2^{i+1}$(mod 2^m)>id>=$n+2^i$(mod 2^m). The selected peer is the one that is closest in round trip distance.
   a) At each step, the distance between query and peer-with-file reduces by a factor of at least ¾ (see PROOF below). If we can prove this, it means there is logarithmic reduction of distance to key at every step, and the routing latency is O(log(N)). For some constant c, it takes c·m hops where 2^m is at most a multiplicative factor above N. [Here c = (ln2)/(ln(4/3))]. The lookup cost is O(log(N)). (Once in the vicinity of the target key, it takes O(log(N)) hops by the same argument).
   PROOF: Assume the distance between current node n and target key k along the ring is d (in points along the ring). We need to prove that the length of the next hop is at least ¼ d.
   Proof by contradiction: Let the query be currently at Q. Assume the next hop node H (i.e., farthest hop from Q, but still at or anticlockwise of the key K) is less than ¼ d points distance from Q. Assume the next hop node H is at ft[i] of current node Q's fingertable. Then we have $Q+2^i <= H < Q+2^{i+1}$. Since H is less than ¼ d away, we can conclude that $Q+2^{i+2} <= K$. This means that ft[i+2] will be between H and K, and thus H is not the farthest next-hop from Q that is to the anticlockwise of K (if there is no node in the interval $Q+2^{i+1}$ to $Q+2^{i+2}$, then K will be the next hop. This is a contradiction to our assumption. Thus the next hop after Q must be beyond ¼ d distance away from Q towards K. This means the distance reduces by a factor of at least ¾ at each hop. Thus the routing latency is $\log_{(¾)}$ N, or log(N).
   b) Since there is no new entry in the finger table, the memory cost does not change from the original Chord algorithm and remains to be O(log(N)).
   c) Routing distance is no worse than Chord since each finger table is at least far along the ring as the corresponding Chord finger table. But this Variant of Chord chooses the neighbor by looking for the closest peer in round trip distance, which is the closest in terms of latency. Thus, it achieves the effect as Pastry does with its locality principle, which makes this variant of Chord almost as topology aware as Pastry.

8. (Solution and Grading by: Matt Hokinson.)

a. We can evaluate the minimum TTL by finding the longest distance between two nodes in this modified tree, which due to the ring containing the leaves would be the two leftmost leaves of each subtree of the root. These nodes are half the number of leaves (2^(m-1)/2, or 2^(m-2)) away, so would require traversing up the tree in some way.
    i. TTL = 2(m-1)
        1. At large m, because the height of the tree is growing linearly (O(log(N)) with m while the number of leaf nodes are growing exponentially (O(N)), it becomes more efficient to walk up to the root of the tree and down to the other leaves. And again, since these are the furthest nodes, they give us the longest path and hence the minimum TTL.
b. Here we will get the nodes contained in the subtree of the child, as well as the nodes in the opposite subtree up to the (m-3)'rd level because of the steps to reach the root node to the other root child. Once again, the ring doesn't help much in terms of cutting the TTL (for the same reasons as in (a)). This gives us the following:
    i. m=3, Nodes reached: 3 (only getting to the root, so we ignore the opposing subtree term below)
    ii. m>3, Nodes reached: 2^(m-1) - 2 nodes in the child subtree (minus the sender), as well as an additional root node and child subtree to the (m-3)'rd level, giving us an additional 2^(m-3) - 1 nodes, for a total of 2^(m-1) + 2^(m-3) - 2 nodes
c. Case by case (you only need to give the answer for large m):
    i. m=2, TTL = 1
    ii. m=3, TTL = 2,
    iii. m>3, TTL = m
        1. When starting at the top of the tree, it quickly becomes faster to move through the root to the other subtree, as well as down your own subtree, rather than using the ring along the leaves. So, we get TTL=m as it takes one step to get to the root, then m-1 steps to move down the opposite subtree.
d. For smaller trees, (m<=4), it is beneficial to use the ring connections of the leaves, giving us minimum TTL's of the following:
    i. m=1 (one node only, ignore)
    ii. m=2, TTL = 1
    iii. m=3, TTL = 2
        1. Thanks to the ring structure, we are able to make the furthest path (the two leftmost leaves of each subtree of the root) within two moves. All other paths are possible within two moves using the ring connections.
    iv. m=4, TTL = 4

1. Each of the leaves can be reached within 4 moves given the ring connections. Any other node higher in the tree can be reached using movements upwards to parent nodes from the leaves, each being reachable within 4 hops either from the first leaf node or from or from another leaf node after moving along the ring.

v. m=5, TTL = 7

1. This is a special case since the fastest way to get from the leaf of one subtree to another is by moving to a node in the other subtree, then moving up the subtree and back down again, which gives us a path of length 7.

9. (Solution and Grading by: Adit Bhagat.)
    a. **M = 2k + 1**
    The max number of failures this algorithm can tolerate is 2k. This is because k is much smaller than N, so it is possible for a process P to never be chosen in one of the random heartbeat sets (i.e. none of the other processes randomly select P to receive heartbeats from). This means that P only has to send heartbeats to its k successors and k predecessors in the ring. Consider the scenario where all those k successors and k predecessors for P fail simultaneously (meaning a total of 2k simultaneous failures). At this point, P would detect all those failures successfully. However, there would then be no remaining processes that P is heartbeating to. If P were to then fail, no process would be able to detect P's failure. Therefore, completeness has been violated at a total of 2k + 1 failures.
    b. **No, this algorithm is not 100% accurate.**
    Heartbeating algorithms do not satisfy accuracy because missed heartbeats (due to message drops or delays) are indistinguishable from a failed process.
    c. **Worst case: N - 1**
    In the worst case, a single process may be randomly selected by every other process in the system, meaning it must send out N - 1 heartbeats every second.
    Best case: 2k
    In the best case, a single process may not be randomly selected by any other process in the system. Therefore, this process would only need to send a heartbeat to its k predecessors and k successors.
    Average case: 3k
    Since each process must select 3k total processes to receive heartbeats from, this means there are (3k * N) total heartbeats sent out every second in the system. Taking the average of this, we have (3k * N) / N = 3k. Therefore, the average load of a process is 3k heartbeats per second.

10. (Solution and Grading by: Tomoko Sakurayama.)

    a. **NVIDIA A100 Tensor Core 80 GB**
    AWS: NVIDIA A100 Tensor Core 40 GB

Azure: NVIDIA A100 PCIe 80 GB
Google Cloud: NVIDIA A100 Tensor Core 40GB

b. An answer that mentions any of the following:
    i. **Speed:** Machine Learning applications require certain mathematical operations that are slow on today's CPUs (or in some cases, not supported), while GPU accelerators are better tuned to these kinds of operations and thus faster. This is also related to the next reason.
    ii. **High Data Throughput for Training**: a GPU consists of hundreds of cores performing the same operation on multiple data items in parallel. Because of that, a GPU can push vast volumes of processed data through a workload, speeding up specific tasks beyond what a CPU can handle.
    iii. **Massive Parallel Computing**: Whereas CPUs excel in more complex computations, GPUs excel in extensive calculations with numerous similar operations, such as computing matrices or modeling complex systems.
c. An answer that mentions any of the following:
    i. **Multitasking**: GPUs aren't typically built for multitasking, so they don't have much impact in areas like general-purpose computing. (However, there are some recent research efforts on multitasking in GPUs, including from NVIDIA).
    ii. **Cost**: While the price of GPUs has fallen somewhat over the years, they are still significantly more expensive than CPUs. This cost rises more when talking about a GPU built for specific tasks like mining or analytics.
    iii. **Power and Complexity**: While a GPU can handle large amounts of parallel computing and data throughput, they struggle when the processing requirements become more chaotic. Branching logic paths, sequential operations, and other approaches to computing impede the effectiveness of a GPU.
    iv. **CPU Advantages over GPU**
        1. **Flexibility**: CPUs are flexible and resilient and can handle a variety of tasks outside of graphics processing. Because of their serial processing capabilities, the CPU can multitask across multiple activities in your computer. Because of this, a strong CPU can provide more speed for typical computer use than a GPU.
        2. **Contextual Power**: In specific situations, the CPU will outperform the GPU. For example, the CPU is significantly faster when handling several different types of system operations (random access memory, mid-range computational operations, managing an operating system, I/O operations).
        3. **Precision**: CPUs can work on mid-range mathematical equations with a higher level of precision. CPUs can handle the

computational depth and complexity more readily, becoming increasingly crucial for specific applications.

4. **Access to Memory**: CPUs usually contain significant local cache memory, which means they can handle a larger set of linear instructions and, hence, more complex system and computational operations.
5. **Cost and Availability**: CPUs are more readily available, more widely manufactured, and cost-effective for consumer and enterprise use. Additionally, hardware manufacturers still create thousands of motherboard designs to house a wide range of CPUs.

========== END of HOMEWORK 1 SOLUTION ==========