HW1 Solutions: CS425 FA24

Recommended solutions for HW1 (FA24) are below. For many questions, alternate solutions are possible and reasonable correct solutions will be accepted during grading. So please refrain from asking questions about solutions until you receive your HW grades back.

1. (Solution and Grading by Han-Ting Liang.)
   (i) No. When the process Pj in Si got infected, due to the restriction, the next O(log(√ N)) rounds Pj picks only inside-subnet targets. The gossip takes O(log(√ N)) time to spread within Si. After O(log(√ N)) rounds, the process can start to pick inside-subnet or outside-subnet targets. Once an outside-subnet target is selected, on average it takes O(1) time for a gossip to go across from Si to S((i+1)mod√ N). The √ N subnets are "daisy-chained" so the average dissemination time for a gossip to spread across all subnets is √ N * O(log(√ N)) + (√ N - 1) * O(1), which is O(√ N log(√ N)) or O(√ N log(N)).

   (ii) No, it can be O(√ N) towards the end of the gossip. The router load is O(1) when a subnet sends a gossip to another subnet, but the question asks "at any time during the gossip spread". When all N nodes are gossiping (towards the end of the gossip spread), since there √ N subnets, there will be on average √ N gossips going through the router.

2. (Solution and Grading by Taimoor Tariq.)
    a. AWS Lambda provides a Serverless computing environment, following a pay-as-you-go model, allowing for auto-scaling according to the customer's needs. EC2 follows a more typical cloud-based architecture where the customer can reserve virtual servers, but is primarily responsible for manual scaling of resources based on usage.
    b. Pricing:
        - **AWS Lambda**: Customer is charged based on the number of requests and the duration of execution but there is no charge for idle time, i.e. when machines are not in use.
        - **AWS Spot Instances**: Spot Instances allow you to bid for unused EC2 capacity at a reduced price compared to On-Demand instances. The pricing can fluctuate but it's generally much cheaper than standard EC2 instances.

        Duration:

        - **AWS Lambda**: Lambda functions are designed to handle short-lived tasks, with a maximum execution time of 15 minutes. Once the task is complete, the function automatically terminates.
        - **AWS Spot Instances**: Spot Instances can run as long as your bid price exceeds the current Spot price. However, they can be interrupted by AWS when the Spot price exceeds your bid or if the capacity is needed for other customers which makes them much less predictable.
    c. A GPU instance on AWS is an instance of a server that includes Graphics Processing Units (GPUs) to accelerate computation for relevant tasks such as machine learning.

        Hardware Acceleration:

        - **GPU Instance**: These are equipped with dedicated GPUs designed to handle parallel processing tasks.
        - **General Purpose Instance**: are primarily designed for a balanced mix of CPU, memory, and networking resources without any specialised hardware accelerators.

Applications:

- **GPU Instance**: Ideal for workloads that involve heavy mathematical computations or parallelization, like deep learning or graphics rendering.
- **General Purpose Instance**: These instances are suitable for a wide variety of general purpose computing tasks.

Processing:

- **GPU Instance**: Optimised for Parallel Processing
- **General Purpose Instance**: Designed for Serial Instruction Processing.

d. **Regular Instances** provide consistent CPU power, suitable for steady workloads. **Burstable Instances** offer a baseline CPU level with the ability to "burst" above it using additional CPU credits, ideal for workloads with highly variable/ 'bursty' CPU demand.

e. A GPU accelerates parallel computing tasks such as graphic rendering or AI workloads, while a TPU (Tensor Processing Unit) is specifically designed accelerating TensorFlow-based machine learning tasks, offering optimized performance for neural network computations.
This is achieved due to the fact that a GPU processes one instruction at a time using multiple 1D arrays, whereas a TPU uses a single 2D matrix to execute one instruction at a time.

f. **AWS EC2: Video Streaming Server:** For hosting a video streaming service that requires consistent performance, large storage capacity, and control over the server environment

AWS Lambda: **Event-Driven Data Processing** – For tasks like processing uploaded files or handling API requests, only charging for the time the code executes.

3. (Solution and Grading by Xinying Zheng.)

    a. Problem1
        i. Docker: Docker is an open-source platform that leverages containerization technology to build, test, and deploy software.
        ii. Container: Containers are a form of operating system-level virtualization that contains the code and dependencies to run an application.
        iii. Virtual Machine (VM): A VM is an emulation of a physical machine that uses a hypervisor, enabling multiple operating systems and applications to run independently on a single host.
        iv. Kubernetes (K8s): Kubernetes is an open-source platform to deploy, scale, and manage containerized applications.
        v. K8s Pod: A K8s Pod is the smallest manageable unit in Kubernetes, which contains one or more containers that share storage and network resources.
        vi. K8s cluster: A K8s cluster is a collection of nodes running pods, often consisting of a control plane and worker nodes that collectively manage workloads

    b. Problem2
        i. Docker: Deploy, manage, and test a small application on a single machine with a consistent development environment, e.g., a web application or games.
        ii. Container: Container runtime is preferred when you want to get lower overhead, access to lower-level container internals, or more seamless interaction with other tools, e.g., a basic web server or Redis cache, for improved resource efficiency and better interaction with Kubernetes.
        iii. VMs: need to run applications on different platforms, e.g., testing applications in different OS platforms.
        iv. K8s: run large-scale distributed systems, e.g., Hadoop, Apache Spark.

    c. Problem3
        i. Scope: A Kubernetes cluster is a unified system that includes multiple nodes, with pods deployed on these nodes.
        ii. Definition: Node and cluster are one or a set of physical or virtual machines providing computational resources, while pods are the smallest deployable units representing application instances within the cluster.

iii. Failure handling: K8s can automatically construct new Pods if one fails. The Pods executing on a failed node must be rescheduled to other nodes. Cluster usually remains available even when some node failure happens.

4. (Solution and Grading by Anna Karanika.)
    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 nh nheartbeat 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.

5. (Solution and Grading by Maleeha Masood.)

    Problem

    1. There are N nodes in the system
    2. There is a link between node A and B with the probability p
    3. What is the smallest p such that all N nodes are connected?

    Solution

    We know p < 1 as p = 1 means all links exist and so that is the maximum value of p. Similarly we know that p > 0 otherwise all nodes are disjoint. p = 1/N^x (where x is 1, 2, 3, …) is too small of a probability to ensure connectivity. The only other logical answer given the information provided is p = O(logN/N). Additionally, the graph is random and follows the Erdős–Rényi model for the analytical solution. The connectivity problem for that random graph model has already been solved and p = O(logN/N).

6. (Solution and Grading by Kai-Siang Wang.)

   Problem

   1. Each node has k neighbor (k is large enough to ensure connectivity)
   2. For a given process, the total set of gossip neighbors is m = O(log(N))

   Solution

   The reasoning is correct. Suppose we let x and y to be the number of uninfected nodes and y the number of infected nodes, where x0=N-1 and y0=1, we can have dx/dt to be -Bxy, where B is the probability that an infected node selects an uninfected node to gossip. In this setting, since m is randomly picked, B is equal to m/N * 1/m = 1/N, which is exactly the same as in the original gossip protocol.

7. (Solution and Grading by Pete Stenger.)

MR1 - output dataset 1 by name

- reads from: D1
- input format: key=(name, location, start, end) values=null

```
M1:
  Output key=name value=("WHERE", location, start, end)

R1: null
```

MR2 - output dataset 2 by name

- reads from: D2
- input format: key=name values=null

```
M2:
  Output key=name value=("positive", -, -, -)

R2: null
```

MR3 - mark time intervals as "positive", and group by location

- reads from: merged output of MR1 and MR2
- input format: key=name, values=("WHERE", location, start, end) U ("positive", -, -, -)

```
M3:
  Output key=key value=value

R3:
  locations = ("WHERE", *) from values
  for (_, location, start, end) in locations
    if ("positive") in values
      Output key=location, value=(start, end, name, "positive")
    else
      Output key=location, value=(start, end, name, "unknown")
```

MR4 - mark unknown people that were in contact with infected

- reads from: MR3 output
- reduce input format: key=location, values=(start, end, name, "positive" U "unknown")

```
M4:
  Output key=key value=value

R4:
  positive_entries = (*, "positive") from values
  unknown_entries = (*, "unknown") from values
  for (positive_interval, _) in positive_entries
    for (unknown_internal, name) in unknown_entries
      if (positive_interval overlaps unknown_interval)
        Output key=name, value=1
```

MR5 - deduplicate

- reads from: MR4 output
- reduce input format: key=names, values=(1)

```
M5:
  Output key=key value=value

R5:
  Output key=name, value=null
```

```
the program should output
1) anyone in contact with an infected person
2) all infected people (optional)
```

8. (Solution and Grading by Aryan Bhardwaj.)

```
# read from input file
M1(a, b):
        Output (key=a, val=(FOLLOWING, b))
        Output (key=b, val=(FOLLOWER, a))
R1(key=user_U, V):
        Collect following = set of all (FOLLOWING, *) value items from V
        Collect followers = set of all (FOLLOWER, *) value items from V

        Output (user_U, (FOLLOWER_COUNT, |followers|)

        if (|following| < 100 AND |followers| >= 10M):
                for user_V in following:
                        Output (user_V, (COND1&2, user_U))
```

```
# read from R1 output
M2(user, value):
        Output (user, value)
R2(user, V):
        If a (COND1&2, arbitrary user_U) entry in V:
                If (FOLLOWER_COUNT, |followers|) in V and |followers| > 10M
                        Output (user_U, 1)
```

```
# read from R2 output
M3(user, value):
        Output (user, value)
R3(user, V):
        Output (user, -)
```

9. (Solution and Grading by Zikun Liu.)

1st Mapper/Reducer

```
Map1 (key=null, value=(a, b)) // where key follows value
{
Emit (a, b)
Emit (b, a)
}
Reducer1 (key=a, value=set of users that a follows)
{
        If value contains @Olympics:
                If value does not contain @ICC:
                        For each item in value:
                                Emit (lexicographic_sort (a, item), 0)
                Else:
                        For each item in value:
                                Emit (lexicographic_sort (a, item), 1)
}
```

2nd Mapper/Reducer

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

Reducer2 (key=(a,b), value=set V){
        If sum(V) ==1:
                Emit (a,b)
}
```

10. (Solution and Grading by Zhikun Wang.)

The MapReduce happens in three phases: Map, Shuffle, and Reduce.

Map Phase:

As there are 20 machines each capable of running 4 tasks simultaneously, 80 tasks can run at the same time. As there are 160 Map tasks in total, the time took to complete the map phase is (160 / 80) * 10 seconds = 20 seconds

Shuffle Phase:

Total amount of data produced by the Map tasks are 160 * 100MB = 16000 MB. As for each Reduce task, it fetches an equal amount of data from each machine, including the local machine. Therefore, 1/20 of the data does not need to be transferred with the network and the total amount of data to be transferred is 16000 MB * 0.95 = 15200 MB.

As the total bandwidth is 10Gbps = 1250MBps, the time taken to transfer all the data is 15200 MB/1250MBps = 12.16 seconds.

Reduce Phase:

Each Reduce task takes 40 seconds to execute. Since there are 80 Reduce tasks and all can run in parallel, the total time for the Reduce phase is 40 seconds.

Total time:

Map Phase: 20 seconds, Shuffle Phase: 12.16 seconds, Reduce Phase: 40 seconds

Total time is 20 seconds + 12.16 seconds + 40 seconds = 72.16 seconds

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