Yuqing

Problem 1

(a)

The key difference is that AWS Lambda is a serverless compute service where you run code in response to events without managing servers, while AWS EC2 provides virtual machines where you have full control over the server, including provisioning, configuration, and management.

(b)

AWS Lambda's pricing is based on request count and duration, while AWS Spot Instances use bidding for variable pricing. Lambda functions are short-lived and event-driven, while Spot Instances can run longer but may be terminated with a two-minute notice.

(c)

Azure functions and Google cloud functions

(d)

For simple event-driven tasks and lightweight processing, choose serverless options like AWS Lambda or Azure Functions. For complex, long-running applications (like game server) requiring customizability and computational power , AWS EC2 is a better choice.

Yuqing

Problem 2

From https://azure.microsoft.com/en-us/pricing/details/machine-learning/, https://cloud.google.com/compute/docs/gpus, and https://aws.amazon.com/ec2/instance-types/

(a)

	AWS	Azure	Google Cloud
Name	g5.16xlarge	NC6s	a2-highgpu-1g
GPU Memory Size (GB)	24	32	40

The `a2-highgpu-1g`

(b)

	AWS	Azure	Google Cloud
Name	g4ad.xlarge / g3.4xlarge	NC6s v2 / Nv6 / Nv12s v3 / NC4as T4 v3	P4
GPU Memory Size (GB)	8	16	8

Either g4ad.xlarge, g3.4xlarge, or P4.

(c)

GPUs are versatile, widely used for various tasks for its parallel execution ability (primarily graphic rendering), including machine learning. TPUs, specific to Google Cloud, excel in deep learning due to optimized architecture and faster performance for matrix operations, making them ideal for AI workloads, but less versatile and widely available than GPUs.

Yuqing

Problem 3

- `Map1(a, b)`
    - Output `(key = a, value = (OUT, b))`
    - Output `(key = b, value = (IN, a))`
- `Reduce1(key = x, values = V)`
    - Collect `S_out` set of all `(OUT, *)` value items from `V`
    - Collect `S_in` set of all `(IN, *)` value items from `V`
    - If `sizeof(S_out) < 10` and `sizeof(S_in) >= 100 million` then
        - For each `(_, followed_user)` of `S_in`
            - Output `(followed_user, (user = x, type = "followed"))`
    - If `sizeof(S_in) >= 100 million`
        - Return `(x, (user = x, type = "following"))`
- `Map2 = identity()`
- `Reduce2(key = x, values)`
    - If exist `(user = x, type = "following")` in `values`
        - Then foreach `(user = y, type = "followed")` in value
            - Output `y`

Yuqing

Problem 4

- `Map1(a, b)`
    - Output `(key = a, value = (OUT, b))`
    - Output `(key = b, value = (IN, a))`
- `Reduce1(key = x, values = V)`
    - Collect `S_out` set of all `(OUT, *)` value items from `V`
    - Collect `S_in` set of all `(IN, *)` value items from `V`
    - If `sizeof(S_in) >= 100 million` then
        - Output `(1, (x, S_in))`
- `Map2 = Identity()`
- `Reduce2(key = 1, values = V)`
    - For all combination pair `((x, Sx_in), (y, Sy_in))` in `V`
        - If `sizeof(Intersection(Sx_in, Sy_in)) >= 100`
            - Output `(x, y)`

Yuqing

Problem 5

> Note: Similar to Exercise 3's construction

- `Map1(a, b)` Read from D1
  - Output `(key = b, value = a)`
- `Reduce1(key = x, values = V)`
  - Read from D2, get the `time_intervals` from user `x`
  - For all `a` in `V`
    - Output `(lexicographic_sorted_pair(x, a), time_intervals)`
- `Map2 = Identity`
- `Reduce2(key = (a, b), values = {time_intervals_1, time_intervals_2 ...})`
  - If `sizeof(values) = 1`,
    - Output `Nothing`
  - If `sizeof(values) = 2`
    - If the `HasIntersection(time_intervals_1, time_intervals_2)` is true
      - Output `(a, b)`
    - where `HasIntersection` is a common function that checks whether two list of ranges has intersections (this could be a naïve algorithm, could be a dynamic programming algorithm learned in CS374, etc. It's just a black-box utility function, and its finer detail is not concerned / very important here)

Yuqing

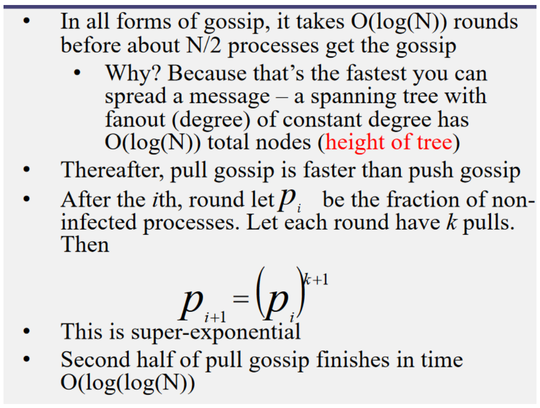
Problem 6

- `Map1(V)`:
    - Output `(key = (V.1, V.2), value = 1)`
    - Output `(key = (V.2, V.3), value = 1)`
    - Output `(key = (V.1, V.3), value = 1)`
- `Reduce1(key = K, values = count)`
- Output `(1, (K, count))`
- `Map2 = Identity()`
- `Reduce2(key = 1, values = P)`
    - Initialize 6 variables
        - If `((A, B), count)` exists, set `AB = count` else `AB = 0`
        - If `((A, C), count)` exists, set `AC = count` else `AC = 0`
        - If `((B, A), count)` exists, set `BA = count` else `BA = 0`
        - If `((B, C), count)` exists, set `BC = count` else `BC = 0`
        - If `((C, A), count)` exists, set `CA = count` else `CA = 0`
        - If `((C, B), count)` exists, set `CB = count` else `CB = 0`
    - Note, `AB` means the number of votes that rate `A` higher than `B`
    - Initialize three counts, `A_condor = 0`, `B_condor = 0`, `C_condor = 0`, then
        - If `AB > BA`, then `A_condor ++`
        - If `BA > AB`, then `B_condor ++`
        - If `AC > CA`, then `A_condor ++`
        - If `CA > AC`, then `C_condor ++`
        - If `BC > CB`, then `B_condor ++`
        - If `CB > BC`, then `C_condor ++`
    - Make list `condors = [A_condor, B_condor, C_condor]` and `names = [A, B, C]`. Get `maxIndices = MaxIndices(condors)`.
        - If `sizeof(maxIndices) = 1`
            - Return `names[maxIndices[0]]`
        - Else
            - Return `names.filter(index in maxIndices)` (that is, return list of elements where its index is in `maxIndices`)

Yuqing

Problem 7

The claim is *half-correct*. As lecture 5 slides proved, both form of gossip have a time complexity of $O(\log N)$ for spreading the $N/2$ of all nodes. Yet, for the second half of the pull gossip, the pull gossip takes $O(\log \log N)$ rather than $O(\log N)$ in pull gossip. (For the proof, it will just refer the slides rather than proving it again)



- In all forms of gossip, it takes O(log(N)) rounds before about N/2 processes get the gossip
  - Why? Because that's the fastest you can spread a message – a spanning tree with fanout (degree) of constant degree has O(log(N)) total nodes (height of tree)
- Thereafter, pull gossip is faster than push gossip
- After the *i*th, round let p_i be the fraction of non-infected processes. Let each round have *k* pulls. Then

$$p_{i+1} = (p_i)^{k+1}$$

- This is super-exponential
- Second half of pull gossip finishes in time O(log(log(N))

So, for the first half of the gossip, the claim is not correct, the time complexity between is both $O(\log N)$, for the second half of the gossip, the claim is correct.

Yuqing

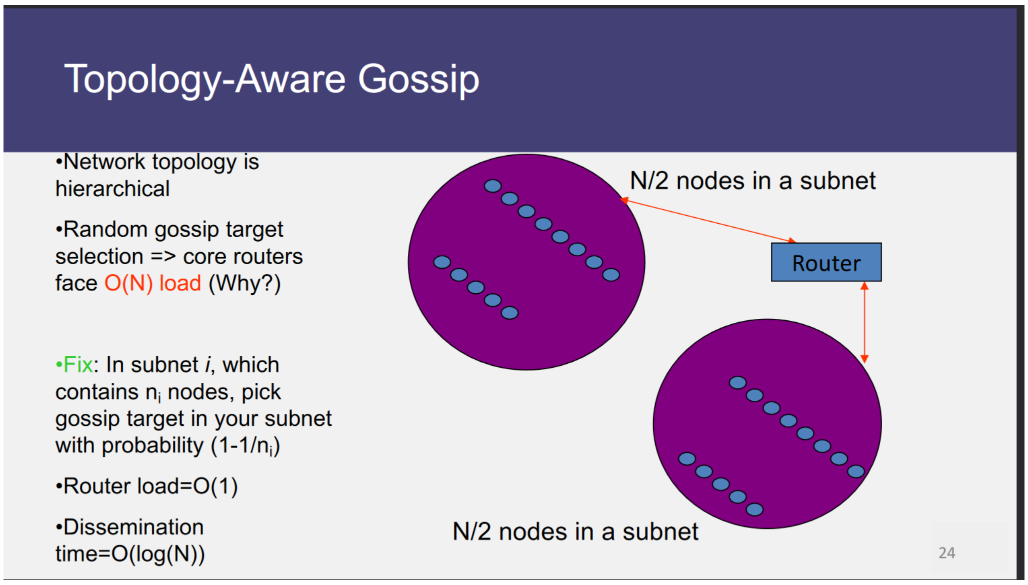
Problem 8

(a)

No, you couldn't. Since foreach subnet, you could only infect your next subnet. That means, suppose we start at subnet 1, then it could only infect subnet 2. For subnet 2, the shortest time that it must wait before infecting subnet 3 is $O(\log\sqrt{N})\cdot$ time. We notice that to infect next subnet, we have to take $O(\sqrt{N})$ time. Since we start from $S(0)$ subnet, and we need to infect all the way to the $S(\sqrt{N}-1)$ subnet, that means, we have to take $O(\log\sqrt{N})\cdot\sqrt{N}$ time at minimum, that is $O(\sqrt{N}\cdot\log\sqrt{N})$, this minimum time is even bigger than the $O(\log N)$ we aim for. So, it's **impossible** to have an $O(\log N)$

(b)

We know that for the topology-aware gossip given in the lecture 5



Topology-Aware Gossip

- Network topology is hierarchical
- Random gossip target selection => core routers face O(N) load (Why?)
- Fix: In subnet *i*, which contains n_i nodes, pick gossip target in your subnet with probability $(1-1/n_i)$
- Router load=O(1)
- Dissemination time=O(log(N))

24

If we want to achieve router load of $O(1)$, then we need to pick the gossip target with probability $(1-1/n_i)$. The reason why it's the case is because for each node in subnet i, it has a probability choosing a target outside of the node of $1/n_i$, therefore, all the nodes in that subnet has a expectation value of choosing the outside node for single gossip round is $n_i\cdot 1/n_i=1$, since in the example given in lecture, there are only 2 nodes (therefore the expected inter-subnet gossips is 2), therefore the router load is $O(1)$.

However, in the problem cases, considering one scenario where there are half of the subnet $\frac{1}{2}\sqrt{N}$ infected, and all of them passed that 'restriction' cool down period. (This scenario is rare, but possible) In this case, foreach subnet, the expected value of choosing the target outside that subnet is $\sqrt{N}\cdot 1/\sqrt{N}=1$, and there are $\frac{1}{2}\sqrt{N}$ subnet, so the total load of the router will be $O(\sqrt{N})$. That means we couldn't maintain $O(1)$ load on router at any time