CS425 Fall 2022 – Homework 1

(a.k.a “Friday Night Plights”)

Due: September 21, 2022 2:00PM

Mohan Amritesh Dasari

mdasari2

**Solution to Q5)**

1. AWS Lambda is a service used for event driven programming. An external event triggers the AWS Lambda function say for example an API call. The key difference between AWS Lambda and AWS EC2 is that Lambda is a serverless application deployment whereas EC2 gives you access to VMs on AWS Cloud Servers. Lamda is scalable based on real-time demands whereas EC2 has a fixed resource pool that needs to be provisioned depending on use-case.
2. Applications for which AWS lambda is a better choice than EC2:
3. Hosting and Managing Serverless Websites: Hosting Frontend on S3, using Lambda functions for logic and requests from the website and using a relational DB for backend. The persistent cost here would be the DB service and S3 service you can pay depending on the traffic. In case of EC2, provisioning a full-on VM would require you to pay for it as well as endure the hassle of maintaining and updating the core OS functionality of the VM.
4. Backend Management for Mobile Application Events: You have developed a mobile application that generates events that need a backend to be managed. You can use Lambda functions to receive the said events and process them in real-time. Managing said system on an EC2 instance would require additional set-up and overhead as well.
5. The key difference between AWS Lambda and AWS Spot Instances is that AWS Spot instances use spare EC2 capacity that is available for less price than the on-demand alternative. Spot instances are cheaper than normal EC2 instances but still you’ll have to manage the entire VM and underlying OS, whereas in Lambda, you only focus on developing the code and do not have to worry about underlying processes. Lambda is a serverless instance whereas Spot is a smaller EC2 instance.

**Solution to Q10)**

1. AWS Amazon Elastic Graphics – 8GB (eg1.2xlarge)

Microsoft Azure – 32GB (NC6s v3 v100)

Google Cloud – 40GB (Nvidia A100)

Hence, Google Cloud offers the most GPU Memory among its competitors which is 40GB of HMB2 Memory.

1. Advantage of using GPUs instead of CPUs for machine learning applications is that GPUs have a High Data throughput. What this means is that a GPU can perform the same task on many data points in parallel with speeds that cannot be matched by CPUs.
2. Disadvantage of using GPUs instead of CPUs for machine learning applications is that GPUs can only perform one application at a time and cannot perform general purpose tasks with great efficiency or speed. They also cannot handle branching logic efficiently.

**Solution to Q1)**

1. The shard with the lowest availability will be fetched first, which in this case is the Shard F, as it is only found on the servers of Venus and Mars.
2. The Order of the shards that will be fetched by the querying node will be  
   **F(2)->B(3)->A(6)->C(6)->E(6)->D(7)**The number in the brackets shows the number of times these shards are replicated on this network of planetary servers.
3. Any combination of shards with the Shard F and not with Shard B will make the computer on-board the shuttle fetch B shard before fetching F shard and hence that is the answer. E.g. DeGrasseTyson(FADE)

**Solution to Q7)**

Diagram, venn diagram

Description automatically generated

We retain the original ring with all the peers in it. We would add another separate ring that would have peers in the same geographical location called the zone ring. The finger selection algorithm doesn’t change from the original one. The only difference is that we’ll also be comparing the latency between the calculated node and the nodes in our zone ring. If the node from the zone ring as lower latency than the calculated node, we replace the calculated node from the zone ring node in the finger table.

1. The lookup cost remains O(log(N)) as we are just changing the way we are storing nodes. With this change the search algorithm won’t be affected and will have the same outcome as before.
2. Memory cost will also remain unchanged as we are neither increasing nor decreasing the size of the finger table. We are just comparing the latencies and choosing the node which is better suited to be a successor. As the memory needed is still the same, Memory cost is neither increased nor decreased.
3. This version of chord will be more topologically aware about its neighbours whereas in the original version, we added the closest node to n+2^i(mode2^m).

**Solution to Q9)**

1. Completeness will be violated when the value of M is 3k. Take for example a node, whose k succeeding nodes have crashed, k preceding nodes have crashed and all the other random nodes have crashed then there is no real way to detect whether this node has crashed or not, since none of the remaining active nodes are expecting a heartbeat from it.
2. The algorithm is not 100% accurate as the heartbeat listeners have no wait time, meaning that in case the service is busy and couldn’t sent the heartbeat in time, the listening services will mark that service as failed even if the service is still active.
3. Worst case calculation should be done with N=3k and Best case calculation should be done with N=2k

**Solution to Q6)**

*Case 1*: Fans only store partial membership list of size k and send to m neighbours,  
N = Total Number of Members

Gossip takes O(log(N)) time to spread. We are sending a message of size k where k<N. The bandwidth of the message being sent is N/k.

The Dissemination time is therefore O((N/k) \* log(N))

*Case 2*: Here the Fans store full membership lists but randomly choose k size partial lists from it to send to m neighbours.

Gossip takes O(log(N)) time to spread. While members store the entire membership list, they are sending only k random entries from the entire N membership list. Therefore, the message bandwidth is N/k.

The Dissemination time here also is therefore O((N/k) \* log(N))

Therefore, the dissemination time for both the cases is same.

**Solution to Q3)**

**M1(a,b):**  a follows b

Output (a, (FOLLOW,b))

**Parallely M1’(a,b)**

Output (b, (FOLLOWER,a))

**R1(x,V):**

followers\_x = {(FOLLOWERS,\*)}

**Parallely R1’(x,V):**

x\_follows = {(FOLLOWS,\*)}

Combining the output of both the Map-Reduce methods we can use the following logic on it:

If |followers\_x|>=4Million and |x\_follows|>=500:

return (followers\_x,V)

Here followers\_x is the user who satisfied the above conditions and V is the set of members who are followers of x

For the next part we apply conditions on the same result from the above Map-reduce function:

If |followers\_x|>=2Million:

return (x,\*)

The conditional statement returns keys which have more than 2 million followers and the value is just an arbitrary \*.

Now we need to combine these 2 results in another Map-reduce method

**M2(a,b)**  Input is the output of second conditional statement

If b = \*:

return (a,\*) # These are the users who have 2 million followers

else:

for x in a:

return (x,b)

# We are returning users who satisfy having at least 4 millions followers and following at least 500 people. Here x is the user that user b (which satisfies the above conditions) follows

**R2(x,V):**  In this reduce function, we are just gonna pair elements in V with the keys x

For i in V:

return (i,x)

The return statement here interprets as i follows x where i has 4 million or more followers and follows 500 or more users. And x satisfies the condition that it has equal to or more than 2 million followers.

**Solution to Q4)**

We’ll be chaining 2 map-reduce methods here for this question

*Map-reduce1:*

M1 Input: *(a,b)* such that a follows b

M1 Output: *(b,a)* The first map function returns the key, value pairs b,a meaning that b is followed by a

R1 Input: *(b,a)* Same as the output from the first Map function

R1 Output: *(x,[V])* The output of this reduce method is the user x along with a set of users V. Here the output is some x user who follows all users in the set V.

So in our case, the values with the key as Packers are the followers of Packers. The values for the key Chicagobears will be the users that follow ChicagoBears. We can find the users from these 2 sets who follow Packers and don’t follow ChicagoBears.

Of the remaining users from the list V after eliminating users who follow ChicagoBears, we can run another Map-reduce method to find whether they follow each other or not.

*Map-reduce2:*

M2 Input: *(a,b)* such that a follows b

M2 Output: *(a,b)* This map function returns the key, value pairs a,b. i.e It is an identity Map.

R2 Output: *((a,b), V)*  This reduce method takes in the Map output and passes it through the function such that the key (a,b) shows us the number of times (a,b) or (b,a) shows up in the Map output. If the value of this key in the Reduce output is equal to 2, then it means that both a and b follow each other and hence that satisfies all the conditions mentioned in the question.