CS 425: Distributed Systems
Homework 1

Net ID: ps71

4/18/2024

Question 1: One of the candidates who used to coach a former president now wants to beat him in the election. This candidate is known for thinking out of the box. While his campaign has been running their services on AWS EC2 instances, they recently became aware of serverless cloud services. Can you help them? (Please limit your answer for each part to less than 50 words. Be concise!)

Part (a): What is the key difference between AWS Lambda and AWS EC2?

AWS Lambda	**AWS EC2**
AWS Lambda is a Function as a Service	AWS EC2 is Infrastructure as a Service
AWS EC2 runs a full copy of the operating system and all the necessary hardware to run the OS. Thus we need to manage and provision the EC2 environment.	AWS Lambda only needs a few system resources and dependencies to run a specific function and AWS handles everything else for us.
We have greater extent of control on the application and its environment on AWS EC2, including RAM, type and number of CPUs and Storage type and volume.	Lambda provides us with a design to run code without the need to deliberately deploy, use or manage VM instances.
Scaling Differences - EC2 instances automatically scale during peak times and decrease during off- peak times, boosting performance and saving money.	Scaling Differences - Lambda make our application event-driven instead of remaining active around the clock, which boosts performance during peak times and saves costs off-peak

Reference - Article [1]

Part (b): What are the two key differences between AWS Lambda and AWS spot instances (think: pricing and how long instances last)?

AWS Lambda	**AWS Spot Instance**
Availability- Always available and can have upto multiple(1000-3000) instances.	Availability- Depends on spot instances availability in the region.
Pricing - True per second billing, useful for short running process.	Pricing - Billing cycle is per 60 seconds, useful for long running process
Always available but runs only when executed. Has up to 100ms of delay before code is loaded and executed.	Instance runs until it is deliberately stopped. The application in the instance is immediately available and running.
Runtime limited to 15 minutes and memory < 3008MB	Can run any application in a suitable instance.

Part (c): What are the names of the corresponding Microsoft Azure and Google Cloud counterparts (names) of Amazon Lambda?

1. Microsoft Azure: The counterpart to AWS Lambda in Azure is Azure Functions. Azure Functions allows you to run code in response to triggers without managing server infrastructure, much like AWS Lambda.

2. Google Cloud: The counterpart in Google Cloud is Google Cloud Functions. This is Google Cloud's serverless execution environment for building and connecting cloud services. Like AWS Lambda, it supports writing function code that responds to events without needing to manage servers.

Part (d): Give one example application (class) where you would prefer AWS EC2, and one where you would prefer AWS Lambda. Justify your choices briefly.

1. EC2 - Long running compute jobs like ML models, Long calculation models.
2. Lambda - Data Analytics, Process data in real-time.

Question 2: Many of the campaigns are using Machine Learning, which of course benefits from GPUs. (Please limit your answer for each part to less than 50 words. Be concise!) Because the campaign has limited budget, they are only looking at single GPU VM instances, i.e., only those that have one (and only one) GPU inside them (but arbitrary amount of memory and CPUs).

Part (a): They want to find the single GPU VM type across all 3 major cloud providers (AWS, Azure, Google Cloud) that has the highest available memory. Can you find it for them? (Note this refers to single GPU, not cloud instance).

1. AWS - The most memory-rich single GPU instance is the G4dn instance. The G4dn.12xlarge variant provides a single NVIDIA T4 GPU and comes with 192 GB of system memory.
2. Microsoft Azure: Azure's NVasv4 series offers the NV32asv4 instance, which includes a single NVIDIA T4 GPU and comes with 224 GB of RAM, making it Azure's highest-memory single GPU virtual machine.
3. Google Cloud: In Google Cloud, the A2 VMs stand out for single GPU configurations. Specifically, the a2-highgpu-1g instance, equipped with one NVIDIA Tesla A100 GPU, offers up to 85 GB of memory.

Part (b): Repeat the previous question but find instead the lowest available memory.

1. AWS - The G4ad instance types offer the smallest memory for single GPU configurations. Specifically, the G4ad.xlarge includes a single AMD Radeon Pro V520 GPU and comes with 16 GB of system memory.
2. Microsoft Azure- The NV series, particularly the NV6 instance, offers a single NVIDIA Tesla M60 GPU. The NV6 comes with 56 GB of RAM, which is the smallest available memory for a single GPU instance in Azure's current offerings.
3. Google Cloud - For single GPU instances, Google Cloud's N1 standard instances configured with a single NVIDIA Tesla T4 GPU, such as the n1-standard-4 with a T4 GPU, can be considered. This setup would generally start with around 15 GB of memory, depending on the exact configuration and availability.

Part (c): What is the difference between a GPU and a "TPU" (among cloud offerings)?

1. Availability: GPUs are more universally available across various cloud platforms like AWS, Azure, and Google Cloud. Each provider offers various types of GPUs tailored to different computing needs.
2. TPUs, on the other hand, are offered primarily through Google Cloud as they are a proprietary technology developed by Google. TPUs in the cloud are integrated deeply with Google's machine learning frameworks and tools.

Question 3: One of the candidates always imitates and emulates a previous president. This candidate believes they will win by becoming the most popular person on social media. An intern in their campaign wants to write a Mapreduce program. In MapReduce, one writes a program for Map that processes one input line at a time and outputs zero or more (key, value) pairs; and one writes a program for Reduce that processes an input of (key, all values for key). The iteration over input lines is done automatically by the MapReduce framework. The intern would like to know who are the influential Twitter users most similar to their candidate, and would like to use Hadoop for this. The intern uses an input file containing information from Twitter (which is an asymmetrical social network) about which users "follow" which other users. If user a follows b, the entry line is (a, b) – you can assume this data is already sharded in HDFS and can be loaded from there. Can you help the intern? Write a MapReduce program (Map and Reduce separately) that outputs the list of all users U who satisfy the following three conditions simultaneously: U has at least 100 million followers, and U herself/himself follows fewer than 10 users, and U follows at least one user V who in turn has at least 10 million followers (e.g., @BarackObama would be such a U). You can chain Mapreduces if you want (but only if you must, and even then, only the least number). Your program must be as highly parallelizable as possible. Correctness is paramount, though you should try to make it as fast as possible. As a rule, all Mapreduce programs must avoid duplicates in the final output. That is, the same output element must not be repeated multiple times in the output.

The conditions the user must follow are:

1. U has at least 100 million followers
2. U herself/himself follows fewer than 10 users
3. U follows at least one user V who in turn has at least 10 million followers

Sol ③

M_1 (K, V) (from HDFS) → [(K, V) ...] (to local storage)

R_1 (K, [V_1, V_2 ..] (from local storage) → (K, _) (to HDFS)

↓
user

① I/p
(a, b) ⇒ a follo
② already stored in HDFS

M_1 (a, b) :

o/p (key = a, value = (OUT, b))

o/p (key = b, value = (IN, a))

R_1 (key = a, V)

⇒ collect S_{out} = set of all (out, *) value items from V

⇒ collect S_{in} = set of all (in, *) value items from V

if $|S_{out}| < 10$ and $|S_{in}| > 100$ M : → eliminate 10M follower acc :

o/p (key = a, value = "#")

if $|S_{in}| > 10$M :

for each x in S_{in} :

o/p (key = x, value = "*") → x follows a user with >10M followers

M_2 : (a, b)

o/p : (a, b)

R_2 : (a, V)

if # & * both in V :

o/p (key = a, value = _)

Question 4: A rival campaign manager believes that finding the best donors is the way to go. They use the same dataset from the previous question to instead find all user pairs (U,V) such that: (i) both U and V have at least 100 million followers each, and (ii) U and V follow at least 100 accounts in common (excluding each other). Note that U and V may or may not follow each other (either way)! Write a Mapreduce program for this. Same instructions as the first Mapreduce question in this series apply. As a rule, all Mapreduce programs must avoid duplicates in the final output. That is, the same output element must not be repeated multiple times in the output. Further if the output type is a pair (a,b), the pair must appear only once - both (a,b) and (b,a) appearing is not allowed.

Soln 4

M_1 : (a, b)

o/p : (key = a , value = (out, b))

o/p : (key = b , value = (in, a))

R_1 : (key = a , value = V)

→ collect S_{out} = set of all (out, *) value item from V

→ collect S_{in} = set of all (in, *) value item from V

if $|S_{in}| > 100M$:

for x in S_{out} :

o/p (key = x, value = a) : a follows x

M_2 : (key = a , value = <list of followers>)

→ for x in V :

for y in V and x != y :

o/p (key = (x, y) value = a)

R_2 : (key = (a, b) , value = < list of acc. (a, b) follows >)

if |V| > 100 :

o/p (key = (a, b) , value = _)

Question 5: One of the social media billionaires is considering running for President. They run a social media named Quitter, and they have access to a lot of data inside the company. As an intern in this campaign, you have the same social network dataset (named D1) specified in the previous question ((a,b) directed pairs indicating a follows b), but you also have an additional dataset (named D2) with entries $(a, start_time, end_time)$ **indicating that user a was online starting** $start_time$ **and ending at** end_time**. The data is only for one day. All times are** $hh : mm : ss$**. However, each user a may have multiple entries in D2 (since users log in simultaneously). Write a Mapreduce program that extracts all pairs of users (a,b) such that: (i) a and b follow each other, and (ii) a and b were online simultaneously at least once during that day. Same instructions as the first Mapreduce question in this series apply. Please ensure that a Map stage reads data from only one input dataset (i.e., if a Map reads directly from D2, don't use it to also read from D1. And vice-versa.) – this is good practice consistent with good Map programming practices. As a rule, all Mapreduce programs must avoid duplicates in the final output. That is, the same output element must not be repeated multiple times in the output. Further if the output type is a pair (a,b), the pair must appear only once - both (a,b) and (b,a) appearing is not allowed.**

Sol^n 5

M_1 : read from D_1 (a, b) $\Rightarrow$ a follows b

o/p : (key = lexicographically sorted (a, b), value = 1)

R_1 : (key = (a, b), value = V)

if V == 2 :

o/p (key = (a, b), value = "#")

M_2 : read from D_2 (a, ST, ET)

o/p : (key = ST, value = a) : assuming users logging in sim. consider online sim.

R_2 : (key = a, value = V)

for x in V :

for y in V and x != y :

o/p (key = (x, y), value = "*")

M_3 : reads from R_1 & R_2 o/p (key = pair (a, b), value = V)

o/p : Identity

R_3 : (key : pair (a, b), value = V)

if "#" & "*" in V :

o/p : (key = (a, b), value = _)

Question 6: Questioning and Reforming the election system seem all the rage nowadays. There are some ways distributed systems folks can help with elections. Someone at the election office thinks MapReduce could be useful for "instant runoff voting" in primaries. (Fun fact: several states, including Alaska, now use instant runoff voting!) Here's how instant runoff voting works. Consider an election with three candidates on the ballot – A, B, C. Every voter ranks the three candidates as first preference, second preference, and last preference. Between any two candidates X and Y, if a majority of voters ranked X above Y, then X dominates Y (and vice versa)—note that this only takes into account X and Y's relative rankings, not where they appear in the preference order, or where the third candidate appears. A Condorcet winner is a candidate that dominates all other candidates (pair wise) on the ballot. By definition an election can have at most one Condorcet winner (however, there may be zero). You are given a dataset of votes from N voters (N is odd and large, and so dataset is sharded), where each vote V has three fields V.1, V.2, V.3, respectively for the first, second, and third preference votes of that voter. Each line of input is one such voter's vote V (input to initial Map function). Write a MapReduce program that outputs either the unique single Condorcet winner among the three candidates A, B, or C, or if there is no single Condorcet winner, then it outputs the list of candidate(s) with the highest Condorcet count (those that dominate the most number of candidates). For background – in MapReduce, one writes a program for Map that processes one input line at a time and outputs zero or more (key, value) pairs; and one writes a program for Reduce that processes an input of (key, all values for key). The iteration over input lines is done automatically by the MapReduce framework. You can assume this data is already sharded in HDFS and can be loaded from there. Each line is one vote V and is read as the value and the key is empty (in the first by Map stage). Note that intermediate data from a Map is not available for subsequent stages! Correctness is important, efficiency is secondary (but you must have some parallelism). Write either pseudocode, or clear unambiguous descriptions. As a rule, all Mapreduce programs must avoid duplicates in the final output. That is, the same output element must not be repeated multiple times in the output. Further if the output type is a pair (a,b), the pair must appear only once - both (a,b) and (b,a) appearing is not allowed.

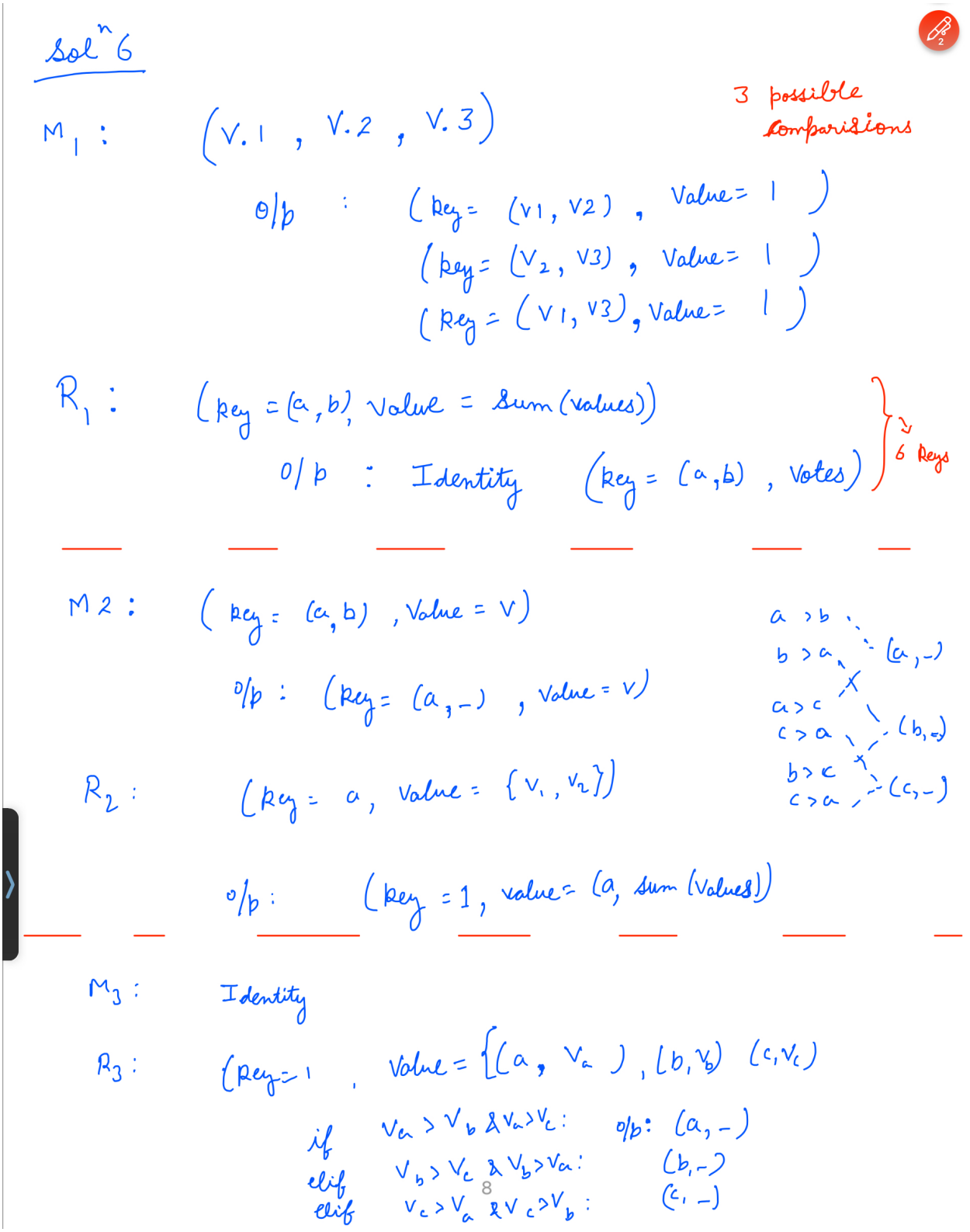

Figure 1: Enter Caption

Question 7: At a presidential debate, one of the candidates loudly proclaims, "You idiots are so slow!". Then the moderator asks, "Can you elaborate please?" At a loss for words, the candidate reaches deep into their CS425 knowledge and screams, "You're all so slow! You're all doing push gossip. I do pull gossip, and even with fixed fanout, it converges in O(log(log(N)) time!" Are they right? If yes, give a proof (informal proof ok). If they are wrong, give a proof (informal proof). (Note: Push gossip and pull gossip mentioned here are the same protocols discussed in lecture)

Yes the claim by the candidate about pull gossip mechanism being faster than push gossip mechanism is correct. The push gossip mechanism follows O(logn) throughout its life, while the pull gossip spreads using O(logn) for dissemination to the first half process and then uses O(log(logn)) for the later half.
Proof- In both the gossips, it takes O(logn) rounds before about N/2 processes get the gossip. This is because the fastest we can spread a message with a spanning tree of constant fanout degree - has a height of O(logn).
But for the later half:

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, and leading to later half of pull gossip to finish in $O(log(logn))$ time.

Question 8: One of the campaigns is always looking for shortcuts. Their distributed system uses a failure detector but to "make it faster", they have made the following changes. For each of these changes (in isolation), say what is the one biggest advantage and the one biggest disadvantage of the change (and why). Keep each answer to under 50 words (give brief justifications). a. They use Gossip-style failure detection, but they set Tcleanup = 0. b. They use SWIM-style failure detection, but they removed the Suspicion feature. c. They use SWIM-style failure detection, but they removed the round robin pinging + random permutation, and instead just randomly select each ping target.

1. One advantage is that it saves space in the short term, and another advantage is that it leads to a shorter time to mark a failed process as failed, because a failed entry is removed immediately. One disadvantage is that the failed servers may remain as ghost entries in the membership list, because every time a member receives gossip containing a server already failed, it treats it as a new entry and will gossip it to other nodes.

2. One advantage of not using the suspicion feature is that the failure detection time is smaller, because a member is considered to have failed immediately after detection rather than wait for suspicion timeout. One disadvantage is that it will have a higher false positive rate, because the suspicion feature reduces false positives by suspect first and then consider the member as failed.

3. One advantage is that it removes the overhead of randomizing the list. One disadvantage is that the completeness is not time-bounded; in the worst case a failed node will never be selected as the ping target.

References

[1] CloudZero. Ecs vs ec2: Choosing the right aws container management service. https://www.cloudzero.com/blog/ecs-vs-ec2/#:~:text=Compared%20to%20AWS%20Lambda%2C%20EC2,AWS%20handles%20everything%20else, 2023. Accessed: 2024-04-18.