CS425 Fall 2023 – Homework 1

(a.k.a. "All the Presidents' People")

Out: Aug 23, 2023. Due: Sep 13, 2022 (Start of Lecture, 2 pm US Central)

Topics: Clouds, Mapreduce, Gossip, Failure detectors, Grids (Lectures 1-6)

Clarifications added 8/30 to Q3,4,5,6

Instructions:

1. **Attempt any 8 out of the 10 problems** in this homework (regardless of how many credits you're taking the course for). If you attempt more, we will grade only the first 8 solutions that appear in your homework (and ignore the rest). Choose wisely!
2. Please hand in **solutions that are typed** (you may use your favorite word processor. We will not accept handwritten solutions. Figures and equations (if any) may be drawn by hand (and scanned).
3. **All students (On-campus and Online/Coursera)** – Please submit PDF only! Please submit on Gradescope. [https://www.gradescope.com/]
4. Please **start each problem on a fresh page**, and **type your name at the top of each page**.
5. Homeworks will be **due at the beginning of class on the day of the deadline. No extensions. For DRES students only:** once the solutions are posted (typically a few hours after the HW is due), subsequent submissions will get a zero**. All non-DRES students must submit by the deadline time+date.**
6. Each problem has the same grade value as the others (10 points each).
7. Unless otherwise specified in the question, the only resources you can avail of in your HWs are the provided course materials (slides, textbooks, etc.), and communication with instructor/TA via discussion forum and e-mail.
8. You can discuss lecture concepts and the questions on Piazza and with your friends, but you cannot discuss solutions or ideas on Piazza.

Prologue: The 2024 US Presidential Election Primary campaigns are heating up! Most Presidential Campaigns today run distributed systems and cloud computing for storage and analytics of data such as the campaign, events, voters, schedules, etc., (e.g., Obama and Romney campaigns in 2012 effectively built and used distributed systems,

including cloud computing and mobile computing, and later candidates also used it, some effectively some less so.).

This homework uses fictitious stories and characters from the ongoing presidential campaigns to frame the homework problems, and to keep the questions fun and relevant (and also informative!). The choice of candidates or parties in questions, or their frequency across questions, is purely arbitrary, and there is no political message in the question framing. Any resemblance to persons, places, or events, living or dead, past, present or future, is purely coincidental. These stories and this homework are not intended to side with or against any candidate, or make comments about any candidate. They are not at supporting or endorsing, nor at criticizing or disparaging, any candidate, campaign, people in campaigns, political parties or affiliations, or voters or citizens or persons living in the US. All interns are fictional, as are their goals and actions.

Problems:

1. (You can use other websites for this question, but you should not cut and paste text. Please write answers in your own words!) 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!)
    a. What is the key difference between AWS Lambda and AWS EC2?
    b. What are the two key differences between AWS Lambda and AWS spot instances (think: pricing and how long instances last)?
    c. What are the names of the corresponding Microsoft Azure and Google Cloud counterparts (names) of Amazon Lambda?
    d. Give one example application (class) where you would prefer AWS EC2, and one where you would prefer AWS Lambda. Justify your choices briefly.
2. (You can use other websites for this question, but you should not cut and paste text. Please write answers in your own words!) 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).

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).
b. Repeat the previous question but find instead the *lowest* available memory.
c. What is the difference between a GPU and a "TPU" (among cloud offerings)?

3. *(Note: From this question onwards you CANNOT use websites but can only use course material. Also, for all Mapreduce questions please only use the class definition/template for Mapreduce jobs, i.e., Map/Reduce tasks/functions, and not other sources from the Web, since there are many different variants on the Web!)* 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.
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.

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.
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.

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)
8. One of the less popular candidates is polling at very small numbers in most of the states. They want to analyze the "topology-aware gossip" protocol you've seen in lecture. However, instead of the lecture slide example of 2 subnets joined by 1 router, here we have a total of N nodes (processes), evenly spread out across $\sqrt{N}$ subnets (each subnet containing $\sqrt{N}$ nodes), all joined by 1 router. The subnets are numbered S0, S1, S2, … S($\sqrt{N}$-1). All these $\sqrt{N}$ subnets are connected together via 1 router. You can assume all nodes have a full membership list, and there are no failures (messages or processes). The topology-aware gossip works as follows. Consider a process Pj choosing gossip targets. The process' gossip targets depend on the subnet Si that it lies in. During a gossip round, the process Pj selects either *b* "inside-subnet Si gossip targets" with probability (1-1/$\sqrt{N}$), OR *b* "outside-subnet Si gossip targets" with probability 1/$\sqrt{N}$. The only "restriction" is that after process Pj is infected, for the next O(log($\sqrt{N}$)) rounds Pj

picks only inside-subnet targets (no outside-subnet targets) -- thereafter in a gossip round at Pj, either all its targets are inside-subnet or all are outside-subnet. Inside-subnet gossip targets from Pj (in Si) are selected uniformly at random from among the processes of Si. Outside-gossip targets from Pj (in Si) are *only* picked from the processes in the "next" subnet S((i+1)mod$\sqrt{}$ N), and they are picked uniformly at randomly from the processes lying in that "next" subnet. The gossiping of a message does not stop (i.e., it is gossiped forever based on the above protocol). Does this topology-aware gossip protocol satisfy both the requirements of: (i) O(log(N)) average dissemination time for a gossip (with one sender from any subnet), and (ii) an O(1) messages/time unit load on the router at any time during the gossip spread? Justify your answers.

9. 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 $T_{cleanup} = 0$.
    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.
10. An intern in the Independent Party campaign designs an independent SWIM/ping-based failure detection protocol, for an asynchronous distributed system, that works as follows. Assume there are N=M*K*R processes in the system (M, K, R, are positive integers, each > 2). Arrange these N processes in a MxKxR 3-dimensional matrix (tesseract), with M processes in each column, and K processes in each row, and R processes in the 3rd dimension (aisles). All processes maintain a full membership list, however pinging is partial. Each process P_{ijk} (in *i*-th row and *j*-th column and *k*-th aisle) periodically (every T time units) marks a subset of its membership list as its *Monitoring Set.* The monitoring set of a given process, once selected, does not change. The monitoring set of P_{ijk} contains: i) all the processes in in its own column P_{*jk} , ii) all the other processes in its own row P_{i*k} , and ii) all the processes in in its own aisle P_{ij*} . At this point, there are two options available to you: Option 1 – Each process sends heartbeats to its monitoring set members. Option 2 – Each process periodically pings all its monitoring set members; pings are responded to by acks, just like in the SWIM protocol (but there are no indirect pings or indirect acks.). Failure detection timeouts work as usual: Option 1 has the heartbeat receiver timeout waiting for a

heartbeat, while Option 2 has the pinging process (pinger) time out. The suspected process is immediately marked as failed. This is run in an asynchronous distributed system.

a. How many failures does Option 1 take to violate completeness? That is, find the value *L* so that if there are (*L-1*) simultaneous failures, all of them will be detected, but if there are *L* simultaneous failures then not all of them may be detected.
b. Answer the same above question for Option 2.
c. An opposition party candidate claims that for K=R=2, *both Option 1 and Option 2* provide completeness for all scenarios with up to (and including) 9 simultaneous failures. You gently respond that they are wrong and that it also depends on M. What are all the values of M (given K=R=2) for which your opponent's claim above is true? Justify your answer clearly.
d. A different opponent claims this algorithm satisfies accuracy for *S* simultaneous failures or fewer, *for both Option 1 and Option 2*. Find the value of *S* (as a function of K, M, R, N, etc.).

====== END OF HOMEWORK 1 =====