CS425 Fall 2017 – Homework 1
(a.k.a. “Friday Night Plights”)

Out: Sep 5, 2017. Due: Sep 26, 2017 (Start of Lecture)

Topics: Clouds, Mapreduce, Gossip, Failure detectors, Membership, Grids, P2P Systems (Lectures 1-8)

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 hardcopy 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.
3. Please start each problem on a fresh sheet (not just page), and type your name at the top of each sheet.
4. Homeworks will be due at the beginning of class on the day of the deadline. No extensions.
5. Each problem has the same grade value as the others (10 points each).
6. Unless otherwise specified, 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.
7. You can discuss lecture concepts and the questions on Piazza and with your friends, but you cannot discuss solutions or ideas. All work must be your own.

Prologue: Today, almost all professional sports teams (e.g., NFL, NBA, MLB, NHL, MLS, etc.) use distributed computing to analyze games, give feedback to players (sometimes in real time), and to decide whom to trade and when.

This homework uses fictitious stories and characters from sports teams to frame the homework problems. Any resemblance to persons, places, things, or events, living or dead, past, present, or future, is purely coincidental. These stories and this homework is aimed neither at endorsing, nor criticizing, any league, any sports team or persons associated with these teams or leagues.
Problems:

1. The Chicago Bears would like to increase their Twitter presence to find big donors. As a warm up exercise they wish to their heavy-hitter followers on social media. They would like to use MapReduce for this. In MapReduce, one writes a program for Map that processes one input line at a time and outputs a (key, value) or nothing; 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 are given an input file containing information from an asymmetrical social network (e.g., twitter) about which users “follow” which other users. If user a follows b, the entry line is (a, b), and this line appears exactly once. Write a MapReduce program (Map and Reduce separately) that outputs all users u such that u follows the @ChicagoBears, and u has at least a million followers. You don’t need to write code – pseudocode is fine as long as it is understandable. Hint: Think about the “key” in Map output. You can chain Mapreduces if you want (but only if you must, and even then, only the least number).

2. (Read the previous question) The Green Bay Packers, being a more democratic ownerless (and some say nonprofit team) laugh at the Chicago Bears and decide to one-up them by finding pairs. They want to output all pairs (a,b) such that both a and b follow @packers, and a and b follow each other (there is no restriction on how many followers a and b have). Your input is the same as question 1. You can chain Mapreduces if you want (but only if you must, and even then, only the least number). Be sure to output each pair at most once (e.g., in sorted order). You don’t need to write code – pseudocode is fine as long as it is understandable. Hint: Think about the “key” in Map output.

3. (Read the previous two questions) But LeBron James snickers at both the above teams and says he by himself is more popular than either of those teams (it’s true! @KingJames has over 37 M Twitter followers!). King James would like to know who are the Twitter users most similar to him, and would like to use Hadoop for this. The input is the same as Question 1. 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. 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 a million followers, and u herself/himself follows at least 10 users, and there is at least one other user v who also has at least million followers so that u and v follow each other (e.g., @KingJames satisfies this). 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.

4. Everyone hates the NE Patriots team, so they decide to design a failure detector. A cornerback has designed a failure detection protocol that works as follows: each process $i$ selects $k$ other target processes at random (once selected, these targets don’t change) and asks these $k$ processes to send it ($i$) heartbeats directly. Heartbeats are not relayed (so this is not gossip, but more like ring failure detection, except there is no ring), and process $i$ times out if it doesn’t receive heartbeats. A process is detected as failed if any of its heartbeat targets do not receive expected heartbeats within a pre-specified timeout.

   a. The quarterback (Tom Brady himself) claims that this algorithm provides completeness up to $(k+1)$ failures. Is he right? If yes, argue why (informal proof). If no, give a counter-example, and also state what completeness the algorithm does provide.

   b. Is the algorithm 100% accurate?

   c. If the period is fixed (say 1 s) at all processes, what is the load on each process in terms of heartbeats that it needs to send? Calculate the worst case, best case, and average load.

5. (For this question you will need to read the original SWIM pre-paper from PODC: http://dl.acm.org/citation.cfm?id=384010 )

   The Chicago White Sox would like to detect failures quickly in their team with $N=100$ players. They plan to use a SWIM-like failure detection protocol. Given a 0% message loss rate and a desired expected detection time of 5 s, how often should pinging happen for the following values of $k$ (or $K$, i.e., number of indirect ping targets)? You can assume only one process fails.

   a. $K=1$

   b. $K=10$

6. Since the Chicago Bulls decided that like they don’t need Derrick Rose, gossip protocols don’t need all processes either in their membership lists. So they decide to build a membership protocol (for a system of $N$ processes) where each process has a membership list of size $k$. The membership list at each process is selected uniformly at random across the entire group (somehow, the messages take care of it – don’t worry about the protocol part). Each message is gossiped to $m$ randomly selected neighbors (from the membership list), where $m < k$, and $m = O(\log(N))$, with the latter needed to ensure spread of gossip. The new Bulls’ point guard (du jour) argues that due to random selection, the overall “behavior” of this protocol (in terms of dissemination time of gossips, etc.) is the same as in the case where all processes might have had full membership lists (known
everyone in the group), and each gossip was sent to \( m \) neighbors. Is he right? If yes, then give a proof. If no, show why.

7. The new manager and coach of the LA Lakers team feel that the team sucks. He wants to connect all Lakers fans together using a P2P system, and have it be topologically aware. Design a variant of Chord that is topology-aware and yet preserves the \( O(\log(N)) \) lookup cost and \( O(\log(N)) \) memory cost. Use examples or pseudocode – whatever you chose, be clear! Make the least changes possible. Hint: Try to change only the finger selection algorithm, but not the routing. Show that (formal proof or informal argument):
   a. Lookup cost is \( O(\log(N)) \) hops.
   b. Memory cost is \( O(\log(N)) \).
   c. The algorithm is significantly more topologically aware than Chord, and almost as topology aware as Pastry.

8. Someone smart on the Pittsburgh Penguins NHL team (after all, they won the Stanley Cups in 2016 and 2017!) designs a new P2P DHT called “AHDHT” (A Hop DHT) where all peers know the addresses of all peers. Their idea is to fully replicate metadata information about the files at all nodes. Their argument is that memory and bandwidth are cheap and plentiful. This way, essentially every node is a super node and searches are local and fast. Suppose each node is allowed only 4 GB of memory. Also assume failure detection comes cheap and you don’t need to worry about its costs. In all the following questions you can use averages to simplify your calculations.
   a. Assume we replicate metadata about files at all nodes. Consider a p2p network where each node uploads about 100 files of average size 5 MB per file and with 100 bytes for file metadata, and there are 3 remote (additional) replicas per file in the p2p system (at other nodes). What is the largest size of the p2p network which would fit the memory limits?
   b. With a 1000 nodes in the p2p network, 100 bytes for file metadata, and 3 remote replicas per file, what is the largest file size that can be supported within the memory limits? Is this realistic, given the p2p system will be used to store short mp3 files that are about 2.4 MB in size.
   c. What is the highest churn rate that can be supported? The incoming bandwidth limit is 100 KBps. (Churn rate is specified in number of nodes joining + leaving per second). You can assume that for each node joining/leaving, 100 B needs to be sent within the next 5 seconds to all the other nodes in the system. (Hint: Read the question carefully. There be dragons, I mean, tricks!).

9. In order to have their players share team videos with each other (but not during the game, because that would be illegal!), the Indianapolis Colts decide to use a
peer to peer system. They use the Gnutella system, except that they change it so that duplicate queries are also forwarded (but with TTL decrement in place as usual). At one point of time, the Gnutella topology looks like a virtual ring with 15 processes, with each process connected to its immediate three clockwise neighbors and immediate three anticlockwise neighbors (thus each process has six neighbors). All links are bidirectional. Answer three parts:

a. A process sends a Query message with TTL=4. How many of the processes receive this Query message? Include the sender in this list.

b. What is the minimum TTL in a Query message to ensure all nodes in the system receive the Query?

c. If we add a 16th process (the coach) that is connected to all the 15 processes, what is the minimum TTL in a Query message to ensure all nodes in the system receive the Query?

10. In order to win back to back championships (after the century-plus drought) the Chicago Cubs realize they need to stop losing fans. They decide that they need to get their fans excited. They decide to use a Chord-like peer to peer system to connect the mobile phones of their fans with each other. The players also start using this system – after a particular game, the players’ mobile phones are in a Chord ring using m = 12, nodes with the following peer ids (or node ids) join the system: 1908, 1945, 1990, 2002, 2013, 2016, 2017, 2020. Then, answer the following questions:

a. Show or list all finger table entries for node 2016.

b. When all finger tables and successors have converged, show the path taken by a search (or query) message originating from node 2016 intended for the key 1999.

c. Node 2020 fails. List all the nodes whose finger tables need to be updated.