CS5412 CLOUD COMPUTING: PRELIM EXAM

Open book, open notes. 90 minutes plus 45 minutes "grace period", hence 2h 15m maximum working time.

We prefer typed responses: Download the exam, and edit it using any editor you like. Then create an Adobe PDF file and upload the PDF file for us. Save your work periodically just to be safe.

Your answers should be fit within the space given, with an 11pt font.

I am aware of the Cornell academic integrity policy, and affirm that this is solely my own work:

Name: _____ *NetId:* _____

Signed: _____

(we prefer a real signature, but a typed signature is ok if you don't know how to insert a signature).

I started at Time: _____ *and finished at* Time: _____

Note: Total time limited to 2h 15m unless you have shown us a letter from Gannett indicating that you require extra time.

In class we often used smart highway (SH) systems as an example of edge computing. Suppose you are building a SH system that acts like an air traffic controller, giving cars permission to drive fast in designated lanes. It scales, dealing with huge numbers of such lanes, all over the country. The input includes photos, video, lidar, car tag information, etc.

1. **Think of CAP. Describe a "sub-task" (a μ-service) needed in such a system where we might relax "C" to gain higher "A and P". Be sure to tell us <u>what the task is</u>.**

2. **Tell us what "C" <u>means</u> for this task.**

3. **Tell us <u>what it means to relax "C" and why doing so is safe.</u>**

4. **For data used by the SH task described in your answers to Q1-Q3, would you employ a scalable key-value store (KVT) to hold that data? If so, explain exactly what the KVT key would be, and what the values would be. If not, explain.**

5. **With respect to your answer to Q4, why would this technology avoid a scalability limit that would be a show-stopper for the system?**

6. **Now assume we decided to use a KVT (no matter what you answered in Q4). Define the concept of a KVT "hot spot".**

7. **Would the Q6 KVT be at risk of hot spots? Either explain why hot spots shouldn't arise, or give two concrete examples in which hot spots could arise.**

Now, continue to think about a service to support a Smart Highway (SH) system, but consider *other* aspects (not the sub-task from Q1-Q7). Identify some other task that requires a Paxos-based solution.

8. **Define this task, and explain why a CAP-style solution with weak consistency would not be adequate, and why Paxos does solve the problem.**

9. **Will your Paxos-based μ-service for Q8 come under heavy load? Justify your answer, thinking about how your system behaves as it scales to larger and larger uses. For example, you could think about whether the average interaction with the average car needs to trigger a Paxos update (if so, the answer is that yes, it would be very loaded).**

For the remaining questions, we are no longer thinking about a SH system. Suppose that we wish to keep an in-memory copy of a very large balanced tree (for example, an AVL tree structure). This tree has an official version in some form of back-end service, but the goal is to have the whole thing cached, to soak up a massive query (read-only) workload.

The AVL tree will cache hundreds of billions of nodes, and requires enough memory to fill the DRAM memory of 1000 computers. To support this model, your group at SkyCloud.com has decided that each node will have a unique node-id used as key, a string representing the node "name", and an associated list of edges pointing to other AVL nodes. In addition, each node would have an associated "value".

The main operations would be the standard AVL operations: look up a node (by its name, to learn the value), insert a node (with a new name and value pair), delete a node (again, by name), etc. Notice that none of these make direct access to the node-id.

10. **Would you recommend using a completely random hashing policy for this kind of data, one in which each key is mapped to some arbitrary value in a hashed space? Explain why this is a good idea, or why it is a bad idea, giving a clear example.**

11. Suppose that SkyCloud.com has a huge amount of data about actual patterns of access to the tree, and from this learns that the read accesses are a lot like the ones for Facebook photos: heavily skewed, so that the great majority of the read operations access a small subset of the nodes. But assume that these accesses *search* for the node. The query doesn't know the key of the node, and has to start from the root and examine (key-value pairs) until it reaches the node it is seeking.

An AVL node lookup of this kind normally takes O(log N) time. For us this means O(log N) nodes will be visited. Invent a way to reduce the cost to O(1) for the most popular AVL nodes. Explain how your solution would work. Hint: you can modify the KVT sharding policy or hashing policy, or both.

12. For your solution in Q11, suppose that nodes have an additional operation, update-value, which doesn't insert or delete the node, but does change the associated value. Does your solution to Q11 support updates? If so, explain why. If not, explain how you would add this extra capability. Would updates also cost O(1), or would they have a higher cost, like O(log(N))? Is locking required? Be detailed!

13. [Skipped, in case anyone is superstitious]

14. A different KVT question. We will use a KVT to store (key,value) pairs where the key is a string and the value is an integer. A task must scan from key A to key B (in alphabetic order), and sum up the corresponding values. Would you prefer for the KVT key hashing function to be highly random, or would you wish for it to preserve the ordering on keys, so the node "owning" A is the first in a sequential list of KVT nodes, with the last one in the list owning B? Explain briefly.

15. With respect to your answer to Q14, discuss the expected performance in terms of delay seen by the query caller.

16. With respect to your answer to Q14, discuss efficiency for the overall KVT μ-service.