739 Midterm: Fall '17

Sometimes, it's just a bunch of questions

ASKING QUESTIONS
SMALL QUESTIONS
LEAD TO SMALL DISCOVERIES.
BIGGER QUESTIONS
LEAD TO BIGGER DISCOVERIES.
SOME QUESTIONS
ONLY REVEAL DEEPER MYSTERIES.
EVEN IF YOU KNOW WHAT QUESTION TO ASK
THE ANSWER MAY SURPRISE YOU.
ASKING ENORMOUS QUESTIONS
CAN CREATE ENORMOUS PROBLEMS.
ASKING TOO MANY QUESTIONS
CAN MAKE YOU LOOK RIDICULOUS.
WHEN YOU COME ACROSS AN UNUSUAL QUESTION
THERE'S NOT MUCH ELSE TO DO
BUT TO STICK WITH THE QUESTION
AND SEE WHERE IT TAKES YOU.
GRANT SNIDER

Usually, exams have something fun that ties them together. But not always. In this case, there is one thing that unites this exam: it's a bunch of questions about distributed systems, some small, some bigger, and hopefully, if we're lucky, unusual enough to take you somewhere new in your thoughts. Think, then answer. Good luck!

NAME:

1. Hamilton, in his advice paper on internet services, writes the following:
   [You should have] “zero trust of underlying components”. What does he mean? Is this sensible advice?

2. Dean advocates the use of “back of the envelope” calculations.
   (a) Why is this useful for system architects?
   (b) In WiscKey, you read about LSMs and some of their advantages. Perform a back-of-the-envelope calculation that demonstrates how much write amplification is tolerable in LSMs on hard drives (i.e., how many sequential writes can one do to replace some random ones?)

3. In the RPC paper, the server receives requests from clients and sets up per "activity" information to track this interaction.
   (a) What is an activity?
   (b) How does the server know when it can garbage collect an activity and related info?
   (c) Finally, what could go wrong if the server prematurely garbage collects such info?

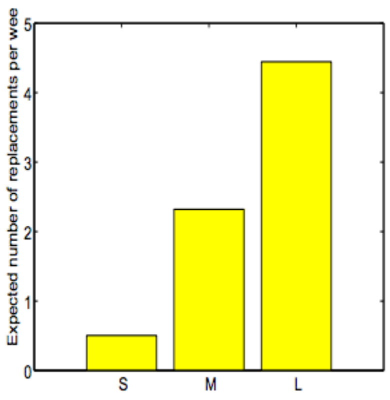
4. Gray shows how to calculate availability as a function of the mean-time-between-failure and the mean-time-to-repair.
   (a) What is this equation? (i.e., Availability = …?)
   (b) Assuming we want a system that has 99.99% uptime and has a mean-time-between-failure of 1 day; how fast does the system have to repair itself in order to reach this availability goal? (approximately)

5. In the Toronto paper about bugs, the following code is shown to both indicate a bug and possible fix:

```
try {
    split(…);
} catch {Exception e) {
    LOG.error("split failed…");
+   retry_split(); // fixed!
}
```

(a) Why was this a bug without the added line?
(b) What is potentially bad about this solution?

6. The Schroeder paper on disk failure makes a number of surprising conclusions. From Figure 7 (shown here), they show the number of disk replacements in a given week, bucketed by the number of replacements in the previous week. What can you conclude from this graph, and why is it important?



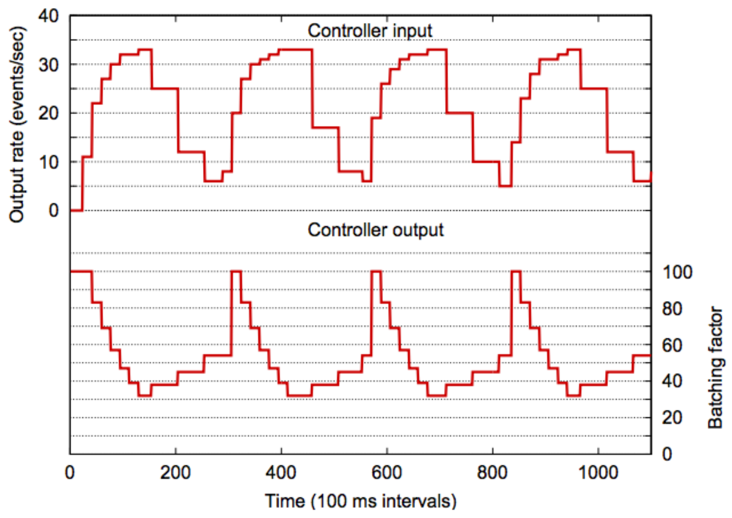
7. In the study of Flash failures from CMU, the authors state “The amount of data written by the operating system to an SSD is not the same as the amount of data that is eventually written to flash cells.” Why do they state this, and why might it be important?

8. Lamport’s clock condition states: For any events a, b: if $a \rightarrow b$, then $C(a) < C(b)$.
   (a) Given two logical clock values for processes on the same machine, what can we conclude about the ordering of the two events?
   (b) What if the clock values we wish to compare were from events on different machines?

9. Vector clocks use more information than Lamport clocks. Give an example where vectors clocks are more useful than Lamport clocks.

10. The Flash web server paper introduces the single-process event-driven (SPED) architecture. What is SPED, and what are the fundamental limits of such an approach?

11. SEDA implements a number of "controllers" to help with performance. One such controller is the "batching" controller, as shown in this graph. What is the batching controller doing, and why is it useful? (what tradeoff is it managing?)



12. The ALICE paper explores how file systems implement crash consistency and how applications built atop such file systems implement crash consistency as well. Why is crash consistency implemented at both of these levels of the system? (why not just in the application, or just in the file system?)

13. The WiscKey paper shows that both read and write amplification are a problem in LSMs (especially in classic LevelDB).
(a) Describe what these two amplifications are.
(b) In LevelDB, which is worse and why?
(c) Finally, in what ways does WiscKey reduce read/write amplification?

14. The NFS system utilizes idempotency where possible to allow retry of operations in case of failure with little ill effect. However, some operations are not (generally) idempotent.
(a) An example is “mkdir”; why is mkdir not idempotent?
(b) Could you design a mkdir with different semantics that is idempotent?

15. Assume the following protocol used in a distributed storage system: to write block X, a client acquires a lease for X (i.e., lease(X)) from a lease server; it then writes to shared storage at location X; it then contacts the lease server to release lease(X). Other clients follow the same protocol. Is this protocol correct? What can go wrong?

16. We now discuss HA-NFS.
    (a) How does HA-NFS handle disk, server, and network failures?
    (b) Overall, what do you think of the HA-NFS design? (its strengths, its weaknesses?)

17. Bonus: We saw two talks in class: the first was from Muthian Sivathanu (MSR) talking about his experiences inside Google; the second was from Venkat Venkataramani (RockSet) talking about his new work at a startup. Describe one thing you learned from each talk.