# 6.824 Spring 2015 Paper Questions

For each paper, your assignment is two-fold. By 10PM the evening before lecture:

- Submit your answer for each lecture's paper question via the submission web site, and
- Submit your own question about the paper (e.g., what you find most confusing about the paper or the paper's general context/problem). You cannot use the question below. To the extent possible, during lecture we will try to answer questions submitted the evening before.

You can also upload your questions and answers using curl:

```
## Answer goes into lecN.txt
$ curl -F file=@lec2.txt \
       -F key=XXXXXXXX \
       http://6824.scripts.mit.edu/submit/handin.py/upload
## Question goes into sqN.txt
$ curl -F file=@sq2.txt \
       -F key=XXXXXXXX \
       http://6824.scripts.mit.edu/submit/handin.py/upload
```

Hypervisor-based Fault-tolerance Suppose that instead of connecting both the primary and backup to the same disk, we connected them to separate disks with identical copies of the data? Would this work? What else might we have to worry about, and what are some things that could go wrong?

Flat Datacenter Storage Suppose tractserver T1 is temporarily unreachable due to a network problem, so the metadata server drops T1 from the TLT. Then the network problem goes away, but for a while the metadata server is not aware that T1's status has changed. During this time could T1 serve client requests to read and write tracts that it stores? If yes, give an example of how this could happen. If no, explain what mechanism(s) prevent this from happening.

Paxos Made Simple Suppose that the acceptors are A, B, and C. A and B are also proposers. How does Paxos ensure that the following sequence of events can't happen? What actually happens, and which value is ultimately chosen?

1. A sends prepare requests with proposal number 1, and gets responses from A, B, and C.
2. A sends `accept(1, "foo")` to A and C and gets responses from both.

Because a majority accepted, A thinks that `"foo"` has been chosen. However, A crashes before sending an `accept` to B.
3. B sends prepare messages with proposal number 2, and gets responses from B and C.
4. B sends `accept(2, "bar")` messages to B and C and gets responses from both, so B thinks that `"bar"` has been chosen.

Russ Cox is one of the leads on the Go project. What do you like best about Go? Why? Would you want to change anything in the language? If so, what and why?

Replication in the Harp File System Figures 5-1, 5-2, and 5-3 show that Harp often finishes benchmarks faster than a conventional non-replicated NFS server. This may be surprising, since you might expect Harp to do strictly more work than a conventional NFS server (for example, Harp must manage the replication). Why is Harp often faster? Will all NFS operations be faster with Harp than on a conventional NFS server, or just some of them? Which?

Spanner Suppose a Spanner server's TT.now() returns correct information, but the uncertainty is large. For example, suppose the absolute time is 10:15:30, and TT.now() returns the interval [10:15:20,10:15:40]. That interval is correct in that it contains the absolute time, but the error bound is 10 seconds. See Section 3 for an explanation TT.now(). What bad effect will a large error bound have on Spanner's operation? Give a specific example.

Secure Untrusted Data Repository (SUNDR) In the simple straw-man, both fetch and modify operations are placed in the log and signed. Suppose an alternate design that only signs and logs modify operations. Does this allow a malicious server to break fetch-modify consistency or fork consistency? Why or why not?

Memory Coherence in Shared Virtual Systems. Answer this question using ivy-code.txt, which is a version of the code in Section 3.1 with some clarifications and bug fixes. You can see that the manager part of the WriteServer sends out invalidate messages, and waits for confirmation

messages indicating that the invalidates have been received and processed. Suppose the manager sent out invalidates, but did not wait for confirmations. Describe a scenario in which lack of the confirmation would cause the system to behave incorrectly. You should assume that the network delivers all messages, and that none of the computers fail.

Distributed Shared Memory on Standard Workstations and Operating Systems Suppose that a simplified version of Treadmarks, called Dreadmarks, simply sent all modifications of variables between an acquire and a release to the next processor to acquire the same lock. No other modifications are sent. What changes does Treadmarks send that Dreadmarks does not? Outline a specific simple situation in which Treadmarks would provide more useful or intuitive memory behavior than Dreadmarks.

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing What applications can Spark support well that MapReduce/Hadoop cannot support?

Ficus Imagine a situation like the paper's Figure 1, but in which only Site A updates file Foo. What should Ficus do in that case when the partition is merged? Explain how Ficus could tell the difference between the situation in which both Site A and Site B update Foo, and the situation in which only Site A updates Foo.

Managing Update Conflicts in Bayou Suppose we build a distributed filesystem using Bayou, and the system has a copy operation. Initially, file A contains "foo" and file B contains "bar". On one node, a user copies file A to file B, overwriting the old contents of B. On another node, a user copies file B to file A. After both operations are committed, we want both files to contain "foo" or for both files to contain "bar". Sketch a dependency check and merge procedure for the copy operation that makes this work. How does Bayou ensure that all the nodes agree about whether A and B contain "foo" or "bar"?

PNUTS: Yahoo!'s Hosted Data Serving Platform Briefly explain why it is (or isn't) okay to use relaxed consistency for social applications (see Section 4). Does PNUTS handle the type of problem presented by Example 1 in Section 1, and if so, how?

Dynamo Suppose Dynamo server S1 is perfectly healthy with a working network connection. By mistake, an administrator instructs server S2 to remove S1 using the mechanisms described in 4.8.1 and 4.9. It takes a while for the membership change to propagate from S2 to the rest of the system (including S1), so for a while some clients and servers will think that S1 is still part of the system. Will Dynamo operate correctly in this situation? Why, or why not?

Kademlia: A Peer-to-peer Information System Based on the XOR Metric Consider a Kademlia-based key-value store with a million users, with non-mutable keys: once a key is published, it will not be modified. The k/v store experiences a network partition into two roughly equal partitions A and B for 1.5 hours.

X is a very popular key. Would nodes in both A and B likely be able to access X's value (1) during the partition? (2) 10 minutes after the network is joined? (3) 25 hours after the network is joined?

(optional) Would your answer change if X was an un-popular key?

Distributed Programming in Argus. Starting at the bottom-left of page 310, the paper mentions that a participant writes new versions to disk **twice**: once before replying to a prepare message, and once after receiving a commit message. Why are both writes necessary? What could go wrong if participants replied to the prepare without writing the disk, instead only writing the disk after receiving a commit message?

Efficient Optimisitic Concurrency Control Using Loosely Synchronized Clocks. The paper mentions that, after a server commits a transaction, the server sends out invalidation messages to clients that are caching data written by that transaction. It may take a while for those invalidations to arrive; during that time, transactions at other clients may read stale cached data. How

does Thor cope with this situation?

Plan 9 from Bell Labs List three features introduced by Plan 9 that have not been adopted by today's common operating systems. For each, why do you think the idea hasn't become popular?

MapReduce How soon after it receives the first file of intermediate data can a reduce worker start calling the application's Reduce function? Explain your answer.

Bitcoin Try to buy something with Bitcoin. It may help to cooperate with some 6.824 class-mates, and it may help to start a few days early. If you decide to give up, that's OK. Briefly describe your experience.

Memcache at Facebook. Section 3.3 implies that a client that writes data does not delete the corresponding key from the Gutter servers, even though the client does try to delete the key from the ordinary Memcached servers (Figure 1). Explain why it would be a bad idea for writing clients to delete keys from Gutter servers.

Although there's no paper assigned for today, please send us a question about Go (the RPC library, channels, threads, lab 1, etc).

The Remus paper's Figure 6 suggests that less frequent checkpoints can lead to better performance. Of course, checkpointing only every X milliseconds means that up to X milliseconds of work are lost if the primary crashes. Suppose it was OK to lose an entire second of work if the primary crashed. Explain why checkpointing every second would lead to terrible performance if the application running on Remus were a Web server.

When will an EPaxos replica **R** execute a particular command **C**? Think about

when commands are committed, command interference, read operations, etc.

Suppose we have the scenario shown in the Raft paper's Figure 7: a cluster of seven servers, with the log contents shown. The first server crashes (the one at the top of the figure), and cannot be contacted. A leader election ensues. For each of the servers marked (a), (d), and (f), could that server be elected? If yes, which servers would vote for it? If no, what specific Raft mechanism(s) would prevent it from being elected?

Give an example scenario where an application would behave incorrectly if it waited for just the first hop in a chain to commit, but would behave correctly by waiting for the entire chain to commit.

Building distributed systems in the real world have both technical challenges (e.g., dealing with bad data) and non-technical ones (e.g., how to structure a team). Which do you think is harder to get right? What examples can you cite from your personal experience?

Experiences with a Distributed, Scalable, Methodological File System: AnalogicFS. In many ways, this experiences paper raises more questions than it answers. Please answer one of the following questions, taking into consideration the rich history of AnalogicFS and the spirit in which the paper was written:

a) The analysis of A* search shown in Figure 1 claims to be an introspective visualization of the AnalogicFS methodology; however, not all decisions are depicted in the figure. In particular, if I <= P, what should be the next node explored such that all assumptions in Section 2 still hold? Show your work.

b) Despite the authors' claims in the introduction that AnalogicFS was developed to study SCSI disks (and their interaction with lambda calculus), the experimental setup detailed in Section 4.1 involves decommissioned Gameboys instead, which use cartridge-based, Flash-like memory. If the authors had used actual SCSI disks during the experiments, how exactly might have their results changed quantitatively?

c) AnalogicFS shows rather unstable multicast algorithm popularity (Figure

5), especially compared with some of the previous systems we've read about in 6.824. Give an example of another system that would have a more steady measurement of popularity pages, especially in the range of 0.1-0.4 decibels of bandwidth.

d) For his 6.824 project, Ben Bitdiddle chose to build a variant of Lab 4 that faithfully emulates the constant expected seek time across LISP machines, as AnalogicFS does. Upon implementation, however, he immediately ran into the need to cap the value size to 400 nm, rather than 676 nm. Explain what assumptions made for the AnalogicFS implementation do not hold true for Lab 4, and why that changes the maximum value size.

---

Questions or comments regarding 6.824? Send e-mail to 6.824-staff@pdos.csail.mit.edu.

**Top** // **6.824 home** //