1. In Sloppy's notes on the Hamilton paper on internet services, Sloppy wrote:

> *design for failure;*
> *implement redundancy and fault recovery;*
> *don't depend upon a commodity hardware slice;*
> *support multiple versions of software;*
> *avoid multi-tenancy;*

Did Sloppy get these lessons right? What should be fixed, and why?

do depend on commodity hardware slice

develop single version software

embrace multi-tenancy

All explained on pp. 2-3 of paper

2a. Sloppy took notes on frequency of failures from Jeff Dean's talk:

~10,000 network rewirings
~3 router failures
~10 individual machine failures
~1000s of hard drive failures
Everything else works perfectly, i.e., when a machine is up, it just works well without any performance problems, etc.

Did Sloppy get these numbers right? Which look too high, which look too low, which just right? What else should be corrected?

→ ~1 net rewiring (not ~10000)

→ ~1000 machine failures (not ~10)

→ All else does not work perfectly (slow disks, bad memory, flaky machines, etc.)

2b. Sloppy also wrote down these "numbers everyone should know":

L2 cache reference	100ns
Main memory reference	100ns
Send 2KB over 1 Gbps network	20,000ms
Disk seek	1,000ms
Read 1MB from disk sequentially	2,000ms

What did Sloppy get wrong here? What should (approximate) correct values be?

→ L2 7ns (not 100)

→ Send 2KB 20K ns (not ms)

→ Disk seek 10 ms (not 1000 ms)

→ Read 1MB 20 ms (not 2000 ms)

3. Sloppy wrote down these lessons from Gray's analysis of Tandem systems:

- The key to high availability is tolerating hardware and software faults
- The key to software fault-tolerance is to hierarchically decompose large systems into modules
- Processes are made fail-safe by defensive programming
- Most production software bugs are soft Bohrbugs

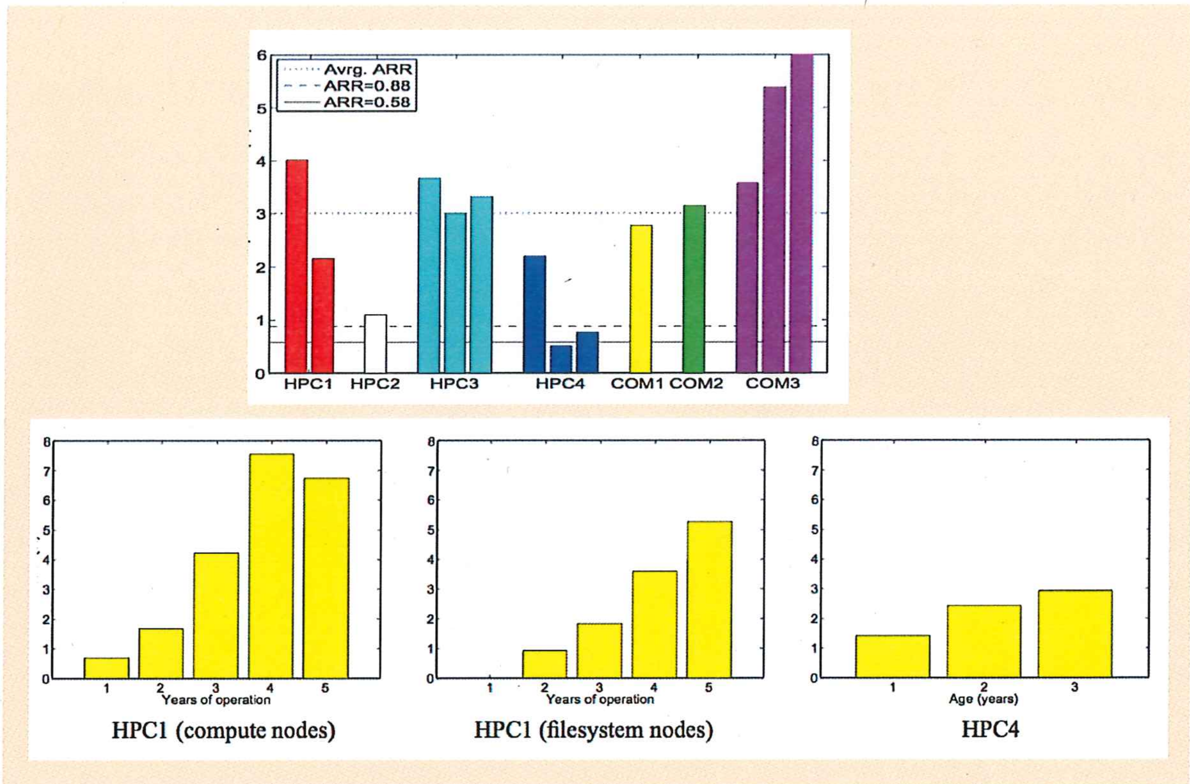
What did Sloppy get wrong here? Please fix as need be, and explain each point.

=> key to HA:
tolerating operations + S/W faults

=> processes are made fail-fast
by defensive prog

=> most prod. bugs are Heisenbugs

4. Sloppy put these figures into Sloppy's notes, but forgot to add some key points of clarification. Here are the figures:



What is ARR, as these figures are reporting?

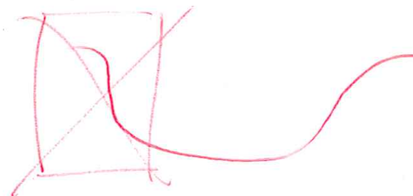
Annual Replacement Rate (ARR)

What is the conclusion you can draw from the first figure (the top graph)?

⇒ Observed ARR >> Datasheet ARR

What is the conclusion you can draw from the second figure (with three graphs)?

⇒ No "infant mortality" period just steady, then increase

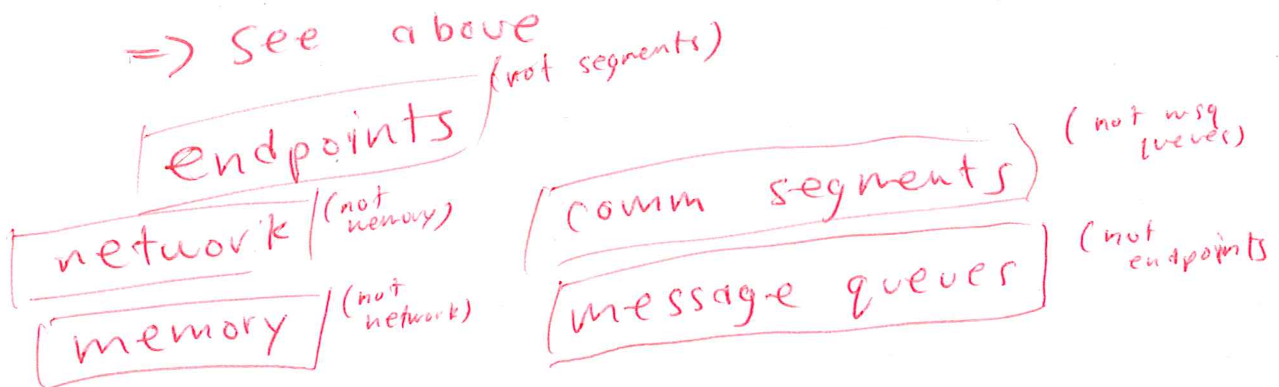


5. Sloppy thought that U-net was easy to understand. However, Sloppy's notes seem to be full of mistakes about U-net, including some word mixups:

> ... communication segments serve as an application's handle into the memory system and contain message queues which are regions of the network that hold message data, and endpoints which hold descriptors for messages that are to be sent or that have been received.

(Handwritten corrections above the words: "endpoints" over "communication segments"; "network" over "memory system"; "comm. segments" over "message queues"; "memory" over "network"; "message queues" over "endpoints".)

What did Sloppy mix up here? Please fix, perhaps by rewriting the paragraph to make sense.



Sloppy also had jotted down that "*true zero copy is hard to achieve*". Is this true? If so, why? If not, why not?

Sec 3.3

yes, true → need NI to integrate w/ MM/VM system

Limited memory (pinned) for data send + receive

Must move in/out of app data structures as result (copy)

6. Sloppy loves the Lamport clocks paper. Unfortunately, Sloppy can't quite get the rules right. This is what Sloppy wrote down:

```
To send a message:
- time = time + 2;
- send_message(msg, time);

To receive a message:
- (message, time_stamp) = receive_message();
- time = min(time_stamp, time) + 2;
```

What did Sloppy get wrong here? Please fix, and explain each point.

time = time + 1

time = max (ts, t) + 1

Are all of Sloppy's errors equally problematic? Explain.

min is wrong

+2 or +1 don't really matter

7. Sloppy also likes vector clocks. However, because Sloppy is sloppy, sometimes Sloppy refers to them as "victor clocks"; Sloppy thinks someone named Victor invented them. Oddly, Sloppy didn't write down much information about them:

> Victor clocks have the following send/receive rules:
> - send ...
> - receive ...
>
> Man, am I bored today in class. I wish the teacher was more "lively", like Victor probably was. I also wish I knew Victor: how did this genius come up with such awesome clocks? Perhaps because Victor had a nice watch.

What are the rules for sending/receiving in vector clocks, specifically regarding how time is updated at message send/receive?

send: inc own clock by 1,
send vector

recv: inc own clock by 1,
then update clock w/
pairwise max of (own vector,
recv'd vector)

8. Sloppy jotted a few things down about the Flash web server:

uses "writev()" system call - what is that?

uses "mincore()" system call - should look this up

uses "pathname translation cache" - what in the world is that?

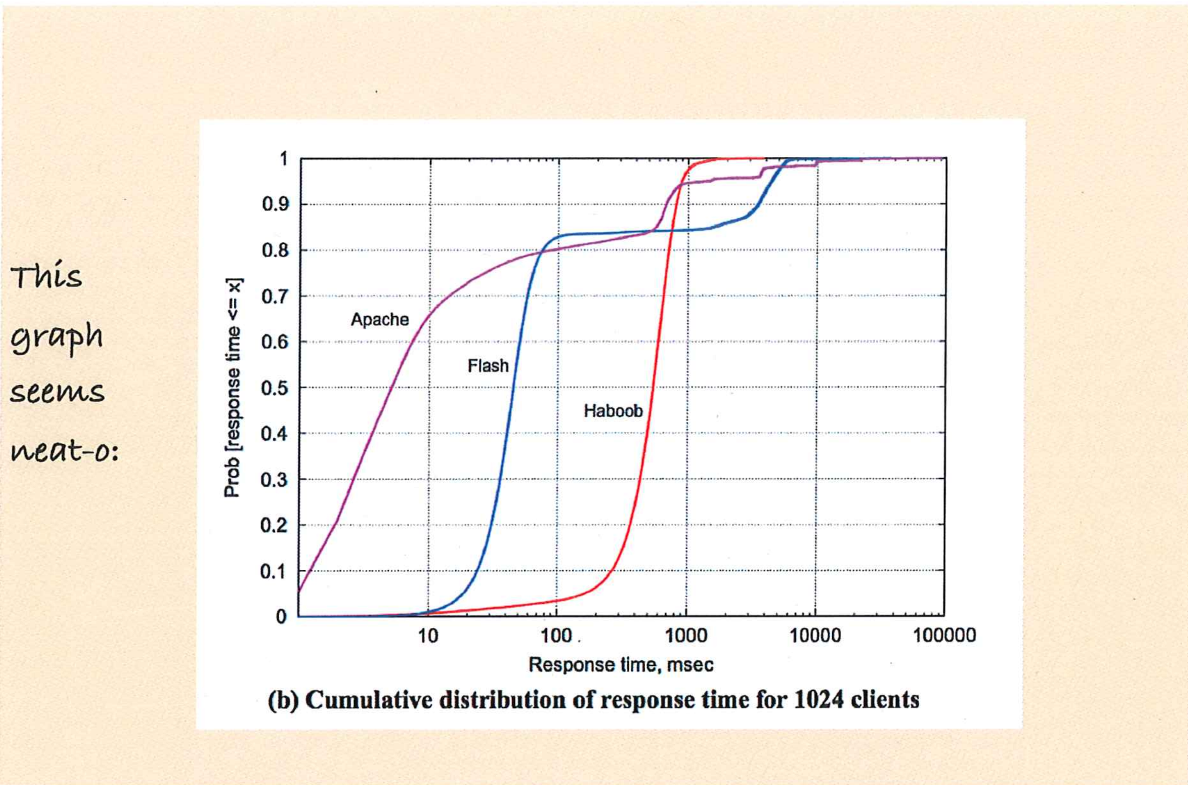
What are these things, and why does the Flash web server use them?

writev : vectored write
⇒ can send non-contig data w/ one syscall
⇒ useful to send headers+data
⇒ Flash does this carefully wrt alignment

mincore : check if file pages are in memory
⇒ Flash: serve those pages first

pathname translation cache :
(e.g. ~bob ⇒ /home/users/bob/public/html)
⇒ avoid expensive helper when possible

9. Sloppy copied this into Sloppy's notes about SEDA, unfortunately without much other useful information:



(b) Cumulative distribution of response time for 1024 clients

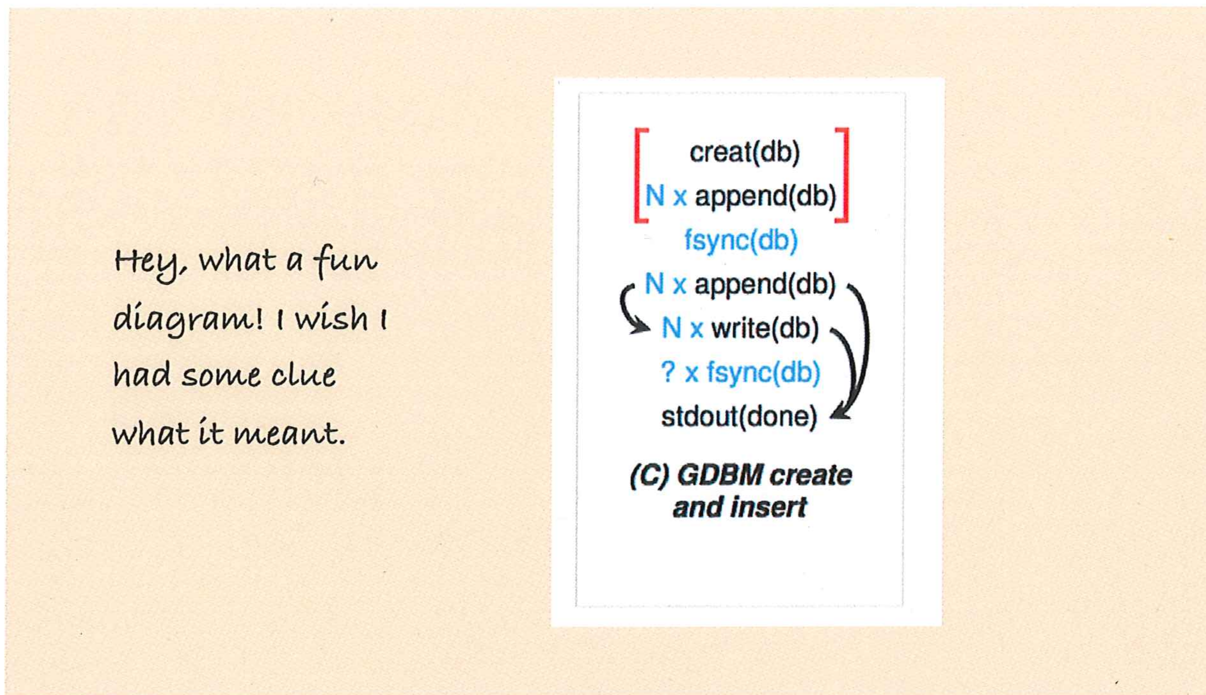
What is this graph showing? What conclusions can you draw from it?

=> Cumulative dist of response times of Apache, Flash, Haboob

=> For most reqs, Apache > Flash >> Haboob

=> But, Haboob has much shorter tail (1 sec not 10 secs)

10. Sloppy liked the ALICE paper, especially those fun diagrams like this one:



What does this diagram reveal about the update protocol? (what do the arrows and red brackets indicate?)

[ ] => need to be executed atomically for protocol to work

x → y => means x must come before y for protocol to work

Does the protocol show any correctness problems with the protocol? If so, describe, and if not, say why not.

yes: => no two syscalls exec. atomically (in general)

=> fsync dir?

=> first append must come before write

(stdout after write/append is OK, due to fsync)

11. Sloppy took copies notes about the WiscKey paper, sensing it might be one of the most important papers that Sloppy has ever read:

> I/O amplification is important
> Really, I/O amplification is the real deal
> This is the professor's paper, so take extra careful notes (!!)
> Also: There can be read or write amplification; remember this, Sloppy!

It seems like Sloppy thinks I/O amplification is important. Explain what I/O amplification is, and why it happens in typical LSMs.

=> I/O amplification: ratio of total read/written / requested by user

=> LSMs do a lot of compaction (extra reads/writes)

lookups also bounce through many files, through each file (extra reads)

Describe the difference between read and write amplification. Which one is generally higher, as measured in the WiscKey paper?

difference: obvious (as above)

higher: read (Fig 2)

lookup to find file, to find k/v in file

Finally, why does the WiscKey approach reduce I/O amplification? By how much, roughly?

all the shuffling of k/v pairs is reduced to just k/ptr pairs

if v is large, this is a big savings

Also, tree is small + fits in memory; fast lookup w/o I/O

12. Sloppy wrote the following down about NFS:

> Understanding NFS means understanding behavior on writes.
> When a write() system call occurs on a client, the client must immediately force it to the server
> When a WRITE protocol request reaches the server, the server must immediately force it to persistent storage (i.e., the disk)

What did Sloppy get wrong here? What did Sloppy get right? Explain (in detail).

write on client: wrong
can be buffered
flush on close() (or earlier)

write on server: correct
(otherwise may lose data)

13. Sloppy wrote down the following protocol that uses leases to provide mutual exclusion in a distributed service:

> There are three machines in the system: Client, Lease Server, Server. The Server has files on it (which can be read or written), and the Lease server gives out leases to clients as desired. Use the Lease Server to ensure mutual exclusion during file update from different Clients.
>
> To update a file:
> - Client: obtain lease (time_length=10seconds) from Lease Server (if lease not already held)
> - Client: Issue request to update file to Server, thus ensuring mutual exclusion while update takes place
> - Client: If lease is about to run out, renew it (if desired)

What did Sloppy get wrong here in the file-update protocol, if anything? Describe why this works or why it doesn't. If it's broken, how can the protocol be fixed?

Clarify: per-file Leases
Problem:
client has lease, issues request
request: takes long time
client loses lease
other client gets it
mut. ex broken

Solution: fence
req includes lease info
server can reject out of date requests

14. Sloppy studied HA-NFS to better understand Primary/Backup replication. However, it's not clear from these notes that Sloppy really understands it:

> HA-NFS uses primary/backup approach to replication.
> Key advantages of primary/backup approach:
> - Professor talking about something, but can't hear... perhaps should take earbuds out of ears
>
> Key problems with primary/backup approach:
> - Professor writing this down on the board, but can't read it from all the way in the back row... note to self: sit closer next time!

Can you fill in Sloppy's notes with the general pros and cons of primary/backup?

Pros
- -> simpler
- -> solves replication problem
- -> Can make it work w/ existing systems

Cons
- -> hard to do reliable failure detection (split brain)
- -> Slow
- -> Blip during fail/recover
- -> Hard to use multiple backups

Now, add specifics. What does HA-NFS do well? What problems does it **not** solve?

Neat: diff approaches to server, whole disk, net failure

Not solved: disk block corruption, flaky (but working) machines, disk, net

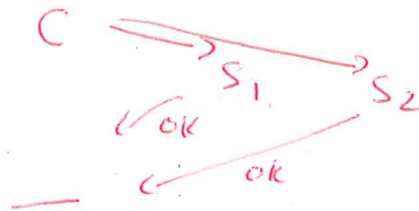
many others (colluding, malicious, servers)

physical proximity req'd

15. Sloppy thinks Sloppy has figured out a new protocol even better than two-phase commit. It's called "one-phase commit", and works as follows:

> Sloppy's one-phase commit:
>
> The coordinator sends the command to each (replicated) server.
>
> Each server does the command (after all, the coordinator's the boss!) and then acknowledges the coordinator.
>
> The one-phase commit is now over. Turing award, here I come!

Is anything wrong with Sloppy's one-phase commit? If so, what? Are there circumstances or environments where it could work?



Problems: many!

=> interleaving of reqs from clients

=> diff ordering

=> fail on one, succeed on other:

=> no agreement

etc..

Could work:

Coord is highly reliable,

only one coord,

server can't reject request,

etc.

(unrealistic)

16. Sloppy took the following notes on quorums.

Quorums based on idea of node union.
Assume N nodes (N>0).
To ensure union, send operations to at least N/2 nodes.
Common configuration: N=4, write set size=3, read set size=2.
Weaken consistency by writing to all N, only reading from 1. Cool optimization!

What did Sloppy get wrong here? Please fix, and explain each point.

node intersection not union
($r+w>N$, $w+w>N$)

send ops ~~to~~ such that intersection ≥ 1 node

$N=4$ $w=3, r=2$ OK

$w=N, r=1$ is not weak?
(strong actually)