6.033 Lecture 14 -- Fault Tolerant Computing 3/31/2014

So far what have we seen:

- Modularity
    - RPC
    - Processes
    - Client / server
- Networking
    - Implements client/server

Seen a few examples of dealing with faults -- e.g.,

- fault isolation via client /server (faults can't propagate between systems)
- providing exactly once semantics in a best-effort system via retries

But haven't really focused on a systematic way to provide fault tolerance

Rest of this semester:
- how to keep running despite failures
- broaden class of failures to include malicious ones

Threats:

- software faults (e.g., blue screen)
- hardware (e.g., disk failed)
- design (e.g., UI, typo in airport code results in crash, therac-25)
- operation (e.g., AT&T outage)
- environment (e.g., tsunami)

For the next few weeks out goal is to understand how to systematically build reliable systems from a collection of unreliable parts.

Basic way this is going to work is to introduce *redundancy*, e.g.:

- in TCP this takes the form of sending data multiple times;

- today: a general technique -- replication of critical system components

- next time: in a file system or database it often takes the form of writing data in more than one place.

Some high level points:

- These techniques improve reliability but building a perfectly reliable system is impossible. Best we can do is to get arbitrarily close to reliable. Our guarantees are going to be essentially probabilistic.

- Reliability comes at a cost -- hardware, computational resource, disk bandwidth, etc. More reliability usually costs more. Have to decide -- as with TCP -- whether the additional reliability is worth the cost. Always a tradeoff.

This lecture:

Understand how to quantify reliability

Understand the basic strategy we use for tolerating faults

Look at replication as a technique for improving reliability / masking faults

Where do computer systems fail?
Diagram (disk, cable, computer, controller, file system sw)

(Everywhere.) Show slide.

Fault anywhere in this system makes compute unusable to user.

Goal is to improve the general *availability* of the system.

Measuring Availability

Mean time to failure (MTTF)
Mean time to repair (MTTR)
Availability = MTTF / (MTTF + MTTR)
MTBF = MTTF + MTTR

Suppose my OS crashes once every month, and takes 10 minutes to recover

MTTF = 720 hours (43,200 minutes)
MTTR = 10
43200 / 43210 = .9997 ("3 nines") -- 2 hours downtime per year

(show availability in practice slide))

<u>What is the general approach to designing fault tolerant systems?</u>

1. Identify possible faults

2. Detect & contain (client/server, checksum, etc.)

3. Decide a plan for handling fault:
   nothing
   fail-fast
   fail-safe
   ..
   mask (through redundancy space or time)

Difficult process, because you must identify *all possible* faults. Easy to miss some possibilities, which then bite you later.

Iterative process

Going to focus on disks for rest of lecture disk systems, but many of these lessons apply to other components. (Will come back to this at the end.)

<u>Why disks?</u>

Unlike other components, disk is the one that if it fails your data is gone -- can always replace the CPU or the cable, or restart the file system.

You can replace the disk, but then your system doesn't work. So cost of disk failure is high.

Ok, so why do disks fail?

(show disk slide)

spinning platters
magnetic surface
data arranges in tracks and sectors
heads "float" on top of surface, move in and out to r/w data

what can go wrong?

whole disk can stop working (electrical fault, broken wire, etc.)

sector failures:

can read or write from the wrong place (e.g., if drive is bumped)
can fail to read or write because head is too high, or coating too thin
disk head can contact the drive surface and scratch it

How often do these failures happen. What does the manufacturer say?

Show Seagate slide

Mean time between failure : 700,000 hours
80 years? 80 years ago we were programming with abacuses.

What's going on here? How did they measure this?

Ran, say, 1000 disks for say 3000 hours (3 million hours total)
Had 4 failures
So claimed that they 1 failure every 750,000 hours

Would buying a more
expensive disk help?

not really   (show slide) -- double the MTTR, 5x the price

What does the actual failure rate look like? Bathtub (5 years)

So what does this 700,000 hours actually tell us?   Probability of failure in the bottom of the tub. Helpful if I am running a data center with a thousand machines and want to know how frequently I will have to replace a disk, assuming whole machine replaced after say 3 years.

Aging happens in practice (show slide)

Do errors really happen?

```
ssh root@db.csail.mit.edu

# smartctl -A /dev/sda > /tmp/smartctl.out
# diff /tmp/smartctl.out <(smartctl -A /dev/sda)
```

.. do something disk-intensive, like du -ks /usr

Coping with failures:

Consider one type of failure -- sector errors

What can I do

1) Silently return the wrong answer

2) Detect failure ("fail fast" — show code)
   each sector has a header with a checksum (e.g., xor all bytes together)
   every disk read fetches a sector and its header
   firmware compares checksum to data
   returns error to driver if checksum doesn't match

3) Correct / mask (show code)
   Re-read if firmware signals error (if an alignment problem)
   (Many other strategies)

These are general approaches -- e.g., could have checksums at the application level (in files), or a backup to recover / correct from a bad file.

What about whole disk failures?

   Powerful solution: replication

   Write everything to two or more disks

   If read from one fails, read from the other (RAID next time)

Diagram:

(show code)

Does this solve all problems? (No)

- Other parts of the system that fail
  - Show Non-stop Slide -- approx cost 300K for 2 nodes;
  - Replicate everything, including operating system -- run N
    - copies of every application, compare every memory read / write
    - Cost is huge, performance is poor

- Assumes failure independence
  - (what about a power failure?

General problem: Software complicated -> bugs! (more likely than most hardware faults!)

==> Fault-tolerant software systems rely on correctness of software inside disk!
how to write bug-free software?

split up code: stuff that “matters” and stuff that “doesn’t matter”

most software on computer doesn't matter — if word crashes its annoying but not critical; code in disk controller causes whole system to not work

for code that does matter,:

- Stringent development practices
- Well-defined stable specification
- Modeling, simulation, verification, etc.
- N-version programming (tricky) / 3MR — tricky!

    Can do voting / triple modular replication -- diagram

Summary

- Disks:
  - lifetime about 5 years
  - 1 %/ year random total failure
  - 10% of disks will develop unreadable sectors w/in 2 years
  - small fraction of uncaught errors
- Replication is a general strategy that can be used to cope

URLs:

http://h20223.www2.hp.com/NonStopComputing/downloads/KernelOpSystem_datasheet.pdf

HP Integrity NonStop NS16000 Server data sheet - PDF (May 2005)

http://www.usenix.org/events/fast08/tech/full_papers/bairavasundaram/bairavasundaram.pdf

http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_7200_10.pdf