Federated Architectures Lecture 10

Fault Tolerant Systems (I)

1

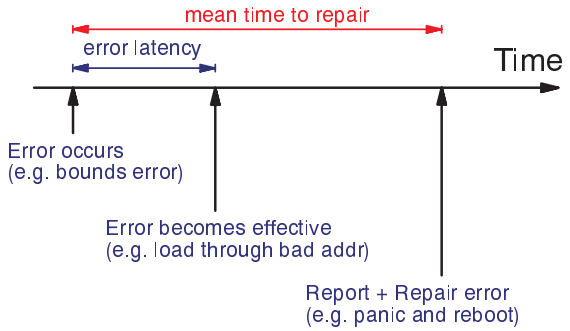
Today

Today: Fault-Tolerance

- Why do machines fail?
- Six-Sigma
- How Six-Sigma is related to software and computer systems
- Best commentary of the week:

  The selection of papers for this week is depressingly reminiscent of the stuff that our quality assurance director keeps shoving under my nose.

Terms



3

Reliability Definition

Probability module A fails over period t = $P_f(A)$

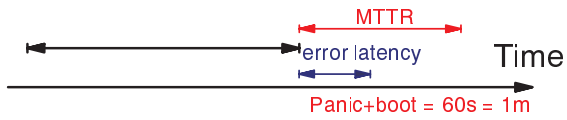
Assume memoryless;
Mean Time To Failure (MTTF) = $\frac{1}{P_f(A)}$

Example: Probability OS fails any day: 0.01

MTTF = $\frac{1}{0.01}$ = 100 days

Q: How accurate is this model for software? Are software errors due to a memoryless process?

Availability Definition



$$\text{Availability} = \frac{\text{Time Accomplished Service}}{\text{Total Elapsed Time}} = \frac{\text{MTTF}}{\text{MTTF+MTTR}}$$

OS example = 99.9993 % available

So what's the problem?

5

More Realistic Data

- Hennessy and Patterson, Figure 7.20
  - 58 DECstation 5000's
  - 654 reboots over an 8974 hr. period (in hrs)
- Assume even distribution over workstations and time (somewhat bogus) and a 60s MTTR

$$\frac{654 \text{ boots}}{58 \text{ machines}} = \frac{11.28 \text{ boots}}{\text{machine}} = \frac{8974 \text{ hr}}{11.28 \text{ boots}} = \text{MTTF} = 33.2\text{d}$$

$$\frac{\text{MTTF}}{\text{MTTF+MTTR}} = 99.998 \text{ \% available}$$

Adding components

Recall: $P_f(A)$ and $P_f(B) \approx P_f(A) + P_f(B)$

$MTTF = 58 \times P_f(machine) \approx MTTF\ 13.65\ hr.$

Clustered systems will be highly UNavailable!

Has any cluster project really dealt with this?
- definate problem with the Berkeley NOW cluster!
- local 10% rule: 10% of the cluster is always unavailable

What can fail?

- Enviroment
  - External Power
  - Sabotage
  - Panics
- Software
  - OS
  - System applications
  - User applications
- Operators
- Hardware

Can we build systems tolerant to as many of these kinds of faults as possible?

With what Freq. do they occur?

What are the effects of each kind?

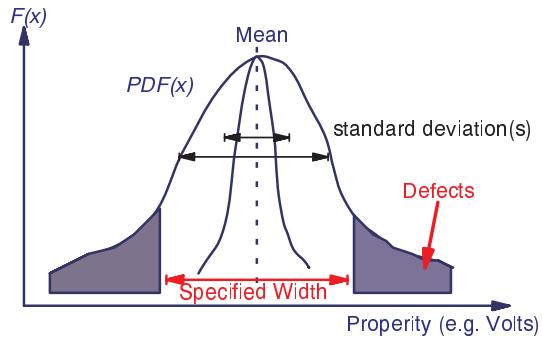
Six-Sigma

Is it all Hype?

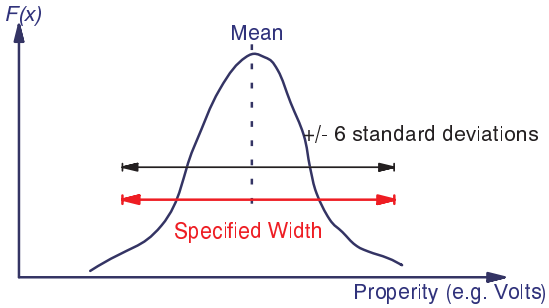
No, actually, some good ideas in there

Q: Can we apply these ideas to software and computer systems in general?
- What does it mean for something to work?
- What is a defect?
- What's a SW "component"?

Six-Sigma Defined

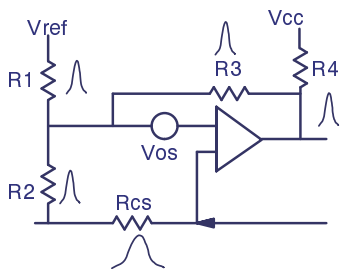


Six-Sigma Defined



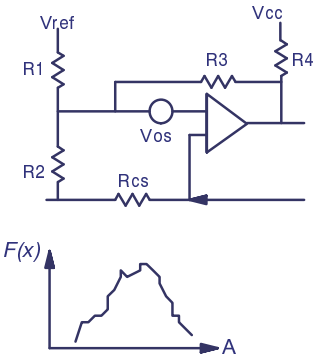
Assuming a normal distribtion, if SW = $\pm 6\sigma$
then 3.4 defects per million units

Six-Sigma design example



Part	Nom.	Tol.
R1	2.49 KΩ	1%
R2	100 Ω	1%
R3	10 KΩ	1%
R4	110 KΩ	1%
Rcs	0.01 Ω	3%
Vref	2.495 V	2.5%
Vcc	12.0 V	5%
Vos	0.0 V	9mV

Six-Sigma Example (cont)



Simulation method:

Assign values for parts based on the PDF for that part

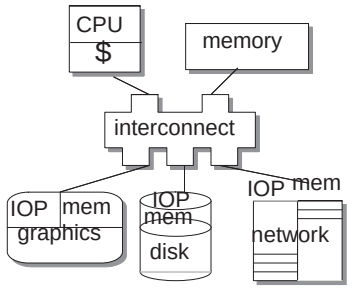
Figure out the A of circuit

Repeat, building a PDF for the whole circuit

Does the distribution match the Q level?

13

Six-Sigma for System Design



What's the Pf for the hardware components?

For the software firmware?

What happens on a failure of these components?

Panic()? Retry N times Something better?

Six Sigma For Software

One idea:

- Identify software components
  - VM, network stack, Filesystem X or Y?
- Define statistical properties about the components
  - NB: Pf(A) depends on environment too
  - Where to get the data? Empirical measurement?
- Create a statistical quality model of the system
  - What's the Pf(system) given the ΣPf(components)?

15

Transactional Example

T = start_transaction()

- write start(T)

do work ….

- write updates to the log

call end_transaction(T)

- write commit(T) to the log

Periodically clean log and update the DB

Can we apply the notions of defered commit and rollback more generally to system state? System calls? File updates?

- What about performance?

Can we build FT software?

Notion of a transaction helps!

- Otherwise, how to restart execution on failure?

A definition of a Heisen vs. Bohr bugs:

- Prob (P) of success given a previous abort
- Heisenbugs are memoryless

Can push this definition through the system all the way back to the user

- Doesn't work? Try again....

17

Testing Availability

- If we have gone through the trouble to quantify P_f for components and the system, we could quantify the Pf of the whole system
- How could we test our P_f(system) calculation? If we had real systems, could see how far the numbers are off
- Need monitoring SW to quantify P_f

Next Week

FT Unix, Borg Et. Al.
Chapter on Tandem Non-Stop OS
Hive, Chapin, Et. Al.

1999 (C) R.P.Martin