Recovery Oriented Computing (ROC)

Dave Patterson and a cast of 1000s:
Aaron Brown, Pete Broadwell, George Candea†, Mike Chen,
James Cutler†, Armando Fox, Emre Kıcıman†, David
Oppenheimer, and Jonathan Traupman

U.C. Berkeley, †Stanford University

November 2010 [2002-2005]
Outline

• Recovery-Oriented Computing: Motivation
• What Can We Learn from Other Fields?
• ROC Principles and Lessons in Retrospect
• ROC => AMP Lab (if time permits)
The past: research goals and assumptions of last 20 years

- **Goal #1:** Improve performance
- **Goal #2:** Improve performance
- **Goal #3:** Improve cost-performance
- **Simplifying Assumptions**
  - Humans are perfect
  - Software will eventually be bug free
  - Hardware MTBF very large
  - Maintenance costs irrelevant vs. Purchase price
Dependability: Claims of 5 9s?

• 99.999% availability from telephone company?
  - AT&T switches < 2 hours of failure in 40 years
• Cisco, HP, Microsoft, Sun ... claim 99.999% availability claims (5 minutes down / year) in marketing/advertising
  - HP-9000 server HW and HP-UX OS can deliver 99.999% availability guarantee “in certain pre-defined, pre-tested customer environments”
  - Environmental? Application? Operator?

5 9s from Jim Gray’s talk: “Dependability in the Internet Era”
“Microsoft fingers technicians for crippling site outages”

By Robert Lemos and Melanie Austria Farmer, ZDNet News, January 25, 2001

• Microsoft blamed its own technicians for a crucial error that crippled the software giant's connection to the Internet, almost completely blocking access to its major Web sites for nearly 24 hours... a "router configuration error" had caused requests for access to the company's Web sites to go unanswered...

• "This was an operational error and not the result of any issue with Microsoft or third-party products, nor with the security of our networks," a Microsoft spokesman said.
Learning from other fields: disasters

Common threads in accidents ~3 Mile Island

1. Latent errors accumulate
2. Operators cannot fully understand system
3. Tendency to blame operators afterwards
4. Systems never working fully properly
5. Emergency Systems often flawed
   - Facility running under normal operation masks errors in error handling

Learning from other fields: human error

- Two kinds of human error
  1) slips/lapses: errors in execution
  2) mistakes: errors in planning

- Human errors are inevitable
  - "humans are furious pattern-matchers"
    » sometimes the match is wrong
  - cognitive strain leads brain to think up least-effort solutions

- Humans can self-detect errors
  - 75% errors immediately detected

Human error

- Human operator error is the leading cause of dependability problems in many domains

- Operator error cannot be eliminated
  - humans inevitably make mistakes: “to err is human”
  - automation irony tells us we can’t eliminate the human

The ironies of automation

• Automation doesn’t remove human influence
  - shifts the burden from operator to designer
    » designers are human too, and make mistakes
    » unless designer is perfect, human operator still needed

• Automation can make operator’s job harder
  - reduces operator’s understanding of the system
    » automation increases complexity, decreases visibility
    » no opportunity to learn without day-to-day interaction
  - uninformed operator still has to solve exceptional scenarios missed by (imperfect) designers
    » exceptional situations are already the most error-prone

• Need tools to help, not replace, operator

Learning from others: Bridges

- 1800s: 1/4 iron truss railroad bridges failed!

- Safety is now part of Civil Engineering DNA

- Techniques invented since 1800s:
  - Learn from failures vs. successes
  - Redundancy to survive some failures
  - Margin of safety 3X-6X vs. calculated load
  - (CS&E version of safety margin?)
Margin of Safety in CS&E?

• Like Civil Engineering, never make dependable systems until add margin of safety (“margin of ignorance”) for what we don't (can't) know?
  - Before: design to tolerate expected (HW) faults

• RAID 5 Story
  - Operator removing good disk vs. bad disk
  - Temperature, vibration causing failure before repair
  - In retrospect, suggested RAID 5 for what we anticipated, but should have suggested RAID 6 (double failure OK) for unanticipated/safety margin?

• CS&S Margin of Safety: Tolerate human error in design, in construction, and in use?
Outline

• Recovery-Oriented Computing: Motivation

• What Can We Learn from Other Fields?

• ROC Principles and Lessons in Retrospect

• ROC => AMP Lab (if time permits)
Recovery-Oriented Computing Philosophy

“If a problem has no solution, it may not be a problem, but a fact, not to be solved, but to be coped with over time”

— Shimon Peres (“Peres’s Law”)

• People/HW/SW failures are facts, not problems
• Recovery/repair is how we cope with them
• Improving recovery/repair improves availability
  - UnAvailability = $\frac{MTTR}{MTTF}$ (assuming $MTTR$ much less than $MTTF$)
  - 1/10th $MTTR$ just as valuable as 10X $MTTF$
• ROC helps Operators
  - Less stress with faster repair times
• ROC enables use of Statistical Machine Learning
  - False positives OK if repair times fast
“ROC Solid” Principles

1. Given errors occur, design to recover rapidly
2. Given humans make errors, build tools to help operator find and repair problems
3. Extensive sanity checks during operation
   - To discover failures quickly (and to help debug)
4. Recovery benchmarks to measure progress
   - Recreate performance benchmark competition?
   - (Skip in interest of time)
Lessons and Observations from ROC

• Fast recovery makes mistakes OK

Observations:

• The power of Statistical Machine Learning

• Visualization to help convince operator of value of Statistical Machine Learning

• REST (Representational State Transfer)
  - Skip for time constraints
Lesson: MTTR more valuable than MTTF???

- Originally: low MTTR allows $MTTR < MTTF$ so $\text{Avail} > 1.0$

- Now: other advantages of low MTTR

- If MTTR is below “human threshold”, failure effectively didn’t occur
  - Example: microrebooting - if can serve a request in <8sec, user doesn’t see the failure

- Tolerates false positives
  - Enables aggressive automatic techniques
  - If Administrator believes there’s a problem, can try “recovery” without incurring high cost if he’s wrong
MTTR more valuable than MTTF???

- MTTF normally predicted vs. observed
  - Include environmental error, operator error, app bug?
  - Much easier to verify MTTR than MTTF!
- If 99% to 99.9% availability, no change in prep
  - 1-3 months => 10-30 months MTTF, still see failures
- Threshold => non-linear return on improvement
  - 8 to 11 second abandonment threshold on Internet
  - 30 second NFS client/server threshold
  - Satellite tracking and 10 minute vs. 2 minute MTTR
- Ebay 4 hour outage, 1st major outage in year
  - More people in single event worse for reputation?
  - One 4-hour outage/year => NY Times => stock?
  - What if 1-minute outage/day for a year?
    (250X improvement in MTTR, 365X worse in MTTF)
Some ROC Accomplishment

- **Crash-only software design philosophy + 3 prototypes**
  - Separation of data recovery from program recovery, and specially-designed crash-only state stores
  - Insight: crash-only-ness simplifies failure detection and recovery

- **Microrebooting components in J2EE applications**
  - “Selective recovery” of bad J2EE components in tens of ms to hundreds of ms (2-3 orders of magnitude faster than restart)
  - Insight: Fast recovery makes mistakes OK
  - Insight: Short transient failures can be completely masked
  - REST (Representational State Transfer) is 2010 version of ideas
Crash Only Software

- Crash-only software refers to code that handle failures by simply restarting, without attempting any sophisticated recovery.
- Components can microreboot into known good state without help of user.
- Failure handling and normal startup use same methods => more likely failure handling code is debugged.
- Managing state: try to have persistent state match running state for quick crash recovery.
- Influenced design of several software products, including Apple OS X.
Visualizing & Mining User Behavior During Site Failures*

• Idea: when site misbehaves, users notice, and change their behaviors; use this as a “failure detector”
  - Partially-supervised learning, since ground truth data often incomplete

• Approach: does distribution of hits to various pages match the “historical” distribution?
  - each minute, compare hit counts of top N pages to hit counts over last 6 hours using Bayesian networks and $\chi^2$ test
  - combine with visualization so operator can spot anomalies corresponding to what the algorithms find

• Evaluation:
  - Which site problems could have been avoided, or to what extent could they have been mitigated, with these techniques in place?
  - Ground truth evaluation of model findings: very hard

Potential Impact: Gaining Operator Trust

- Combining SML with operator centric visualization
  - faster adoption (since skeptical sysadmins can turn off the automatic actions and just use the visualization to cross-check results)
  - earlier visual detection of potential problems, leading to faster resolution or problem avoidance
  - faster classification of false positives
  - Leveraging sysadmin’s existing expertise, and augmenting her understanding of its behavior by combining “visual pattern recognition” with SML

- Increasing operators’ trust in automated techniques
Crisis identification is difficult, time consuming, costly, & lengthen MTTR

Frequent SW/HW failures cause downtime

Timeline of a typical crisis
- detection: automatic, easy
- identification: manual, difficult
  » takes minutes to hours
- resolution: depends on crisis type
  - root cause diagnosis, documentation

Web apps are complex and large-scale
- app used for evaluation: 400 servers, 100 metrics
Insight: performance metrics help identify recurring crises

Performance crises recur
- incorrect root cause diagnosis
- takes time to deploy the fix
  » other priorities, test new code

System state is similar during similar crises
- but not easily captured by fixed set of metrics
- 3 operator-selected metrics not enough

Definition and examples of performance crises

Performance crisis = violation of service-level objective (SLO)
- based on business objectives
- captures performance of whole cluster
- example: >90% servers have latency < 100 ms during 15-minute epoch

Crises Bodik et al analyzed
- app config, DB config, request routing errors
- overloaded front-end, overloaded back-end
Fingerprints capture state of performance metrics during crisis

Metrics as arbitrary time series
- OS, resource utilization, workload, latency, app, ...

1: select relevant metrics
2: summarize using quantiles
3: map into hot/normal/cold
4: average over time

server 1
- 1: CPU utilization
- 2: workload
- 100: latency

server 2
- 1: CPU utilization
- 2: workload
- 100: latency

... server 1000
- 1: CPU utilization
- 2: workload
- 100: latency

OK CRISIS OK
System under study

24 x 7 enterprise-class user-facing application at Microsoft
- 400 machines
- 100 metrics per machine, 15-minute epochs
- operators: “Correct label useful during first hour”

Definition of a crisis
- operators supplied 3 latency metrics and thresholds
- 10% servers have latency > threshold during 1 epoch

19 operator-labeled crises of 10 types
- 9 of type A, 2 of type B, 1 each of 8 more types
- 4-month period
Evaluation results

Previously-seen crises:
- identification accuracy: 77%
- identified when detected or 1 epoch later

For 77% of crises, average time to ID 10 minutes
- Could save up to 50 minutes; more if shorter epochs

Accuracy for previously-unseen crises: 82%

Suggested metrics operators didn't realize were important

Being deployed inside Microsoft
ROC Summary

- Peres’s Law more important than Moore’s Law?
  - Must cope with fact that people, SW, HW fail

- Recovery Oriented Computing is one path for operator synergy, dependability for servers
  - Significantly reducing MTTR (people/SW/HW)
    => Better Dependability (reduce MTTR/MTBF)
    => Reduce Operator Stress (less risk with fast repair)
    => Enable Machine Learning (live with false positives)

  - Compete on recovery time vs. performance?
  - Careful isolation of state for crash only, quick MTTR

http://ROC.cs.berkeley.edu
AMP: Big Data Scalability Dilemma

• Data Analytics frameworks can’t handle lots of incomplete, heterogeneous, dirty data
• State-of-the Art Machine Learning techniques do not scale to large data sets
• Processing architectures struggle with increasing diversity of programming models and job types
• Adding people to a late project makes it later

Exactly Opposite of what we Expect and Need
Algorithms, Machines, People (AMP)

Adaptive/Active Machine Learning and Analytics

Massive and Diverse Data

CrowdSourcing → Cloud Computing

Team of experts in ML, Systems, & Crowds build ≈Real Time Big Data Analyzer
Extra Slides
Traditional Fault-Tolerance vs. ROC

• >30 years of Fault-Tolerance research
• FT greatest success in HW; ignores operator error?
  - ROC holistic, all failure sources: HW, SW, and operator
• Key FT approach: assumes accurate model of hardware and software, and ways HW and SW can fail
  - Models to design, evaluate availability
• Success areas for FT: airplanes, satellites, space shuttle, telecommunications, finance (Tandem)
  - Hardware, software often changes slowly
  - Where SW/HW changes more rapidly, less impact of FT research
• ROC compatible with SW Productivity Tools, SW Churn of Internet Sites
• Much of FT helps MTTF, ROC helps MTTR
  - Improving MTTF and MTTR synergistic (don’t want bad MTTF!)
Recovery benchmarking 101

- Recovery benchmarks quantify system behavior under failures, maintenance, recovery

- They require
  - A realistic workload for the system
  - Quality of service metrics and tools to measure them
  - Fault-injection to simulate failures
  - Human operators to perform repairs

Example: 1 fault in SW RAID

- Compares Linux and Solaris reconstruction
  - Linux: Small impact but longer vulnerability to 2nd fault
  - Solaris: large perf. impact but restores redundancy fast
  - Windows: did not auto-reconstruct!
Software RAID: QoS behavior

- Response to double-fault scenario
  - a double fault results in unrecoverable loss of data on the RAID volume
  - Linux: blocked access to volume
  - Windows: blocked access to volume
  - Solaris: silently continued using volume, delivering fabricated data to application!
    » clear violation of RAID availability semantics
    » resulted in corrupted file system and garbage data at the application level
    » this undocumented policy has serious availability implications for applications
REST (Representational State Transfer)

• Roy Fielding’s PhD thesis, 2000

• Wikipedia: “REST can be considered as a post hoc description of the features of the Web that made the Web successful”

• Idea: everything in your system (in this case, the Web) is a resource

• Requests specify one of a fixed set of operations on some representation of that resource
RESTful requirements

• Client-server, with client separated from server by a uniform interface

• **Stateless**: each request carries all necessary info for server to complete it

• **Cacheable**: responses must specify if representation of resource returned may be cached for future use

• **Layered**: intermediaries can pass on requests, transparently to client or server
REST and Web Services

• A RESTful web service advertises
  - a base URI
  - a way to name a specific resource, starting from that base URI
  - MIME types (JSON, XML, XHTML, ...) supported by available representation(s) of resource
  - a set of requests specifying what can be done to the resource
  - well-defined semantics for each request type
REST Summary

• A key difference between SaaS and SWS is how to encapsulate state
• REST forces you to think about this up front
• Highly recommended: Wikipedia article on REST