Addressing Software Dependability With Statistical & Machine Learning: Successes, Challenges, and an Agenda

ICSE 2005 State-of-the-Art Keynote

by Armando Fox and a cast of tens

v1.5, 18-May-2005

Outline

- Background: Building on a "ROC foundation" (Recovery-Oriented Computing)
- Example-driven overview: using machine learning and statistical induction to attack dependability problems
- Research agenda, fundamental challenges, pitfalls
- Architectural decisions and recoverability

Software problems in production Internet services

- Complexity & churn breed high impact "Heisenbugs"
  - majority of SW bugs in live systems are environment-dependent
  - application bugs result in 28% of non-operator-related downtime for large Internet sites
  - >90% of typical corporate IT budget is maintenance/operations
- Fast detection & rapid recovery are main concern
  - Some Heisenbugs cause *user-visible* application failures *before* they are detected by site monitors (eg Tellme Networks)
  - Gross site metrics track only the *delayed* effect of bugs
  - Management cost dominates TCO => interest in "autonomic"
- Our ability to build & deploy complex systems appears to exceed our ability to understand how they work

History: Recovery-Oriented Computing

- ROC philosophy ("Peres's Law"):

  "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"

  Israeli foreign minister Shimon Peres

  - Failures (hardware, software, operator-induced) are a fact; recovery is how we cope with them over time
  - Availability = MTTF/MTBF= MTTF / (MTTF + MTTR)
  - Making MTTR<<MTTF is just as valuable as increasing MTTF

- Major research areas
  - Fast, generic failure detection and diagnosis (Pinpoint)
  - Fast recovery techniques and design-for-recovery (microrebooting) - prototyped in J2EE
  - System-wide Undo for operators - prototyped in IMAP server

Lesson: other uses for fast recovery

- Fast repair tolerates false positives
  - Keep MTTR below "human perception threshold"
  - Example: microrebooting - if can serve a request in <8sec, user doesn't see the failure
  - *Can be tried even if not sure it's necessary, since cost is so low*
- Human operators are both a major cause of failures and a major agent of recovery for *non-transient* failures
  - Lack of data is *not* the problem: "driving a car by looking through a magnifying glass" effect
  - Rapidly recognizing and recovering from mistakes; intuition/experience about when something's not right with the system
  - *Tools for operators should leverage humans' strengths* to make sense of all this data

Lesson: power of statistical techniques

- Want to talk about "self-*" system goals at high level of abstraction ("response time less than N seconds", etc)
- But these high-level properties are *emergent* from collections of low-level, directly measurable behaviors

Statistical/Machine Learning techniques can help:

- You have lots of raw data
- You have reason to believe the raw data is related to some high-level effect you're interested in
- You lack a model of what that relationship is

SLT applied to problem detection/localization

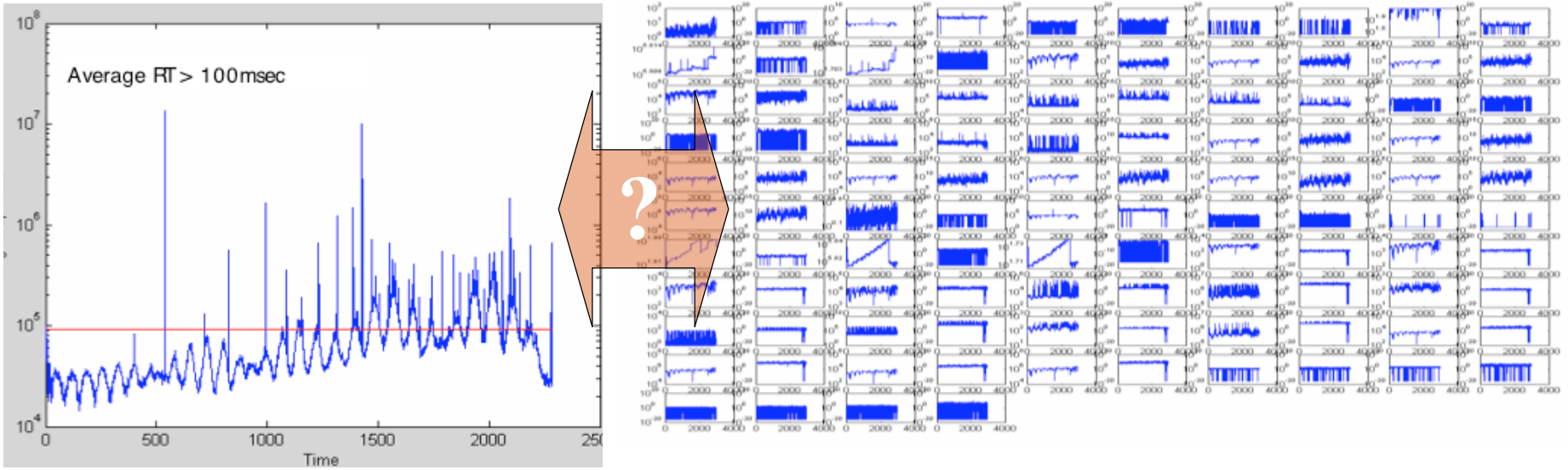
- What kinds of pattern-finding models are possible?
  - Attribution: what low-level metrics correlate with a high-level behavior?
    - Assumption: correlations may indicate root causes
    - Assumption: all required metrics are captured, and model is capable of finding sophisticated correlations
  - Clustering: group items that are "similar" according to some distance metric
    - Assumption: items in same cluster share some "semantic" similarity
  - Anomaly detection: find outliers according to some socring function of anomalousness
    - Assumption: anomalous may indicate abnormal/bad behavior
- A template for applying SLT to problem detection/localization
  - What *directly measurable and relevant* "sensors" do we have?
  - What kind of manipulation on the sensor values (classification, clustering, etc.) might expose the pattern?
  - Tune thresholds/parameters, learn what you did wrong
  - Repeat till publication deadline

Caveats

- Correlation != Causation
  - But it can help a lot, and sometimes the best we can do
- "All models are wrong, but some models are useful"
  - What assumptions are embedded in mapping model to system?
- Without operator's trust and assistance, we are lost

Example: Metric attribution*

- System operator's concern: keep response time below some service-level objective (SLO)
  - If SLO violated, find out why, and fix the problem
- Insight: SLO violation is probably a function of several "low-level" directly measurable metrics
  - But which ones??



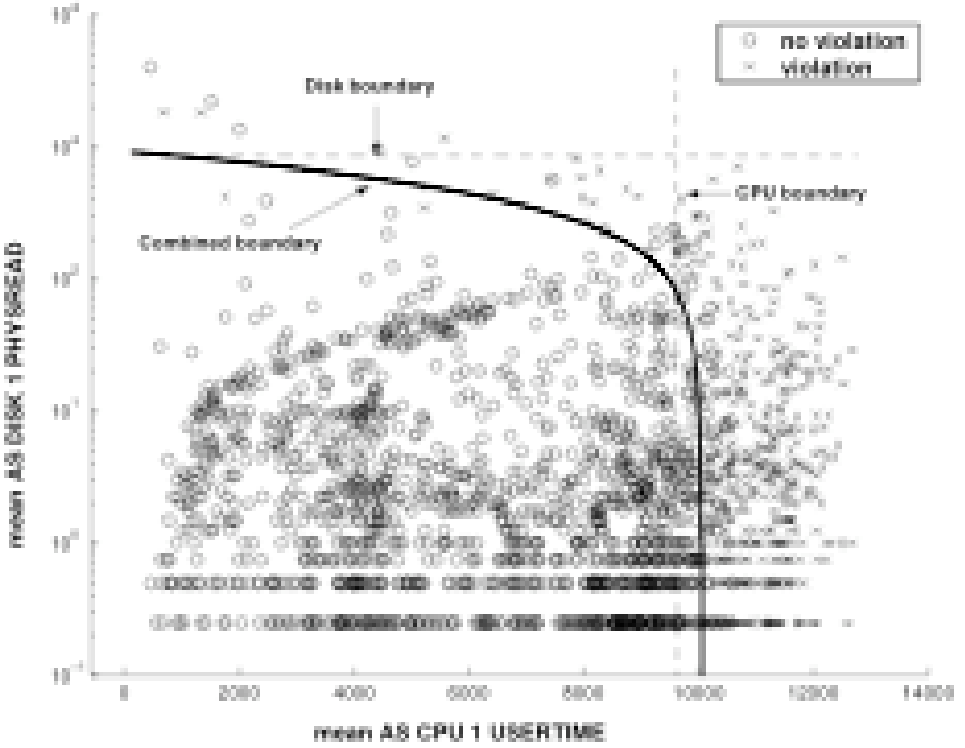
*** S. Zhang, I. Cohen, M. Goldszmidt, T. Kelly, J. Symons (HP Labs), A. Fox, DSN 2005.**

Binary Classification & Bayesian networks

- Goal: given low-level sensor measurement vector **M,** correctly predict whether system will be in compliance or non-compliance with SLO
  - Binary classification is easier than predicting actual latency from metrics!
  - Training the network is *supervised learning* since we know (can directly measure) the correct value of S corresponding to current **M**
- Use a *Bayesian network* to represent joint probability distribution P(S,**M**) (S is either s+ or s-)
  - Because a joint distribution can be *inverted* using Bayes's rule to obtain P(**M**|S), or $P(m_i|m_1,m_2,...m_k,S)$
- A sensor value *m* is "implicated" in a violation if P(*m*|s-) > P(*m*|s+)
  - High classification accuracy increases confidence in whether attribution is "meaningful"

Results & pitfalls

- How well did it work?
  - No single metric discriminates SLO violations very well
  - But collections of 3-8 metrics do very well
  - Balanced accuracy of 90-95% workload

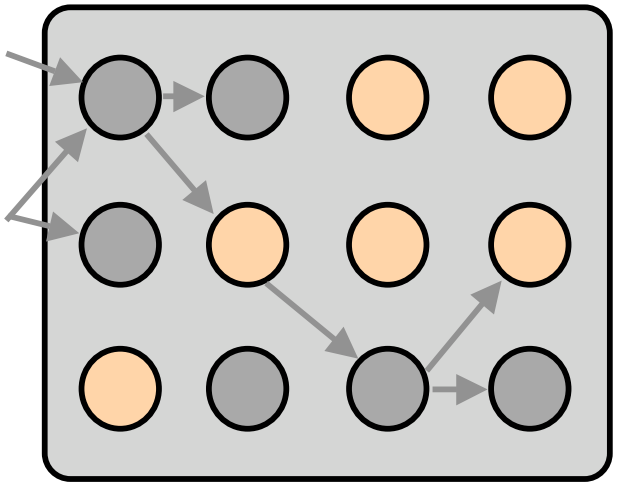


- Training: ~80 data points each of compliance and non-compliance (seconds of operation)
- Assumes we're capturing superset of required metrics
- Attribution is hard to verify empirically

*** Figure: from I. Cohen, M. Goldszmidt et al., OSDI 2004**

Failure detection as anomaly detection*

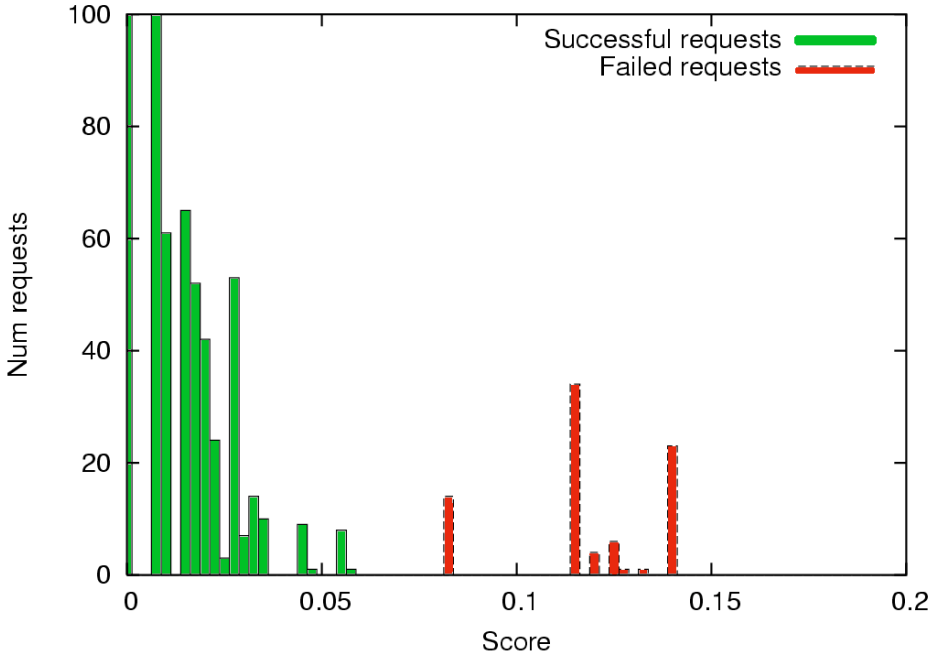
- Problem: detect non-failstop application errors, e.g. shopping cart broken
- Insight: if problems are rare, we can learn "normal" behavior; "anomalous" behavior may mean problem
- *Unsupervised* learning, since goal is to *infer* problems we can't detect directly
- Approach: represent *path* of each request through application modules
  - a parse tree in a probabilistic grammar that has a certain probability of generating any given "sentence"
  - Rarely-generated sentences anomalous



* E. Kiciman and A. Fox, IEEE Trans. Neural Networks, to appear 2005

Results and pitfalls

- Detect 107 out of 122 injected failures, vs. 88 for existing generic techniques (15.5% better, but real impact is on downtime)
  - Supervised learning w/recall and precision used for evaluation
- Impact of false positives
  - Really 2 kinds: algorithmic and semantic
  - Implication: cost of acting on false positive must be low (e.g. microreboot)



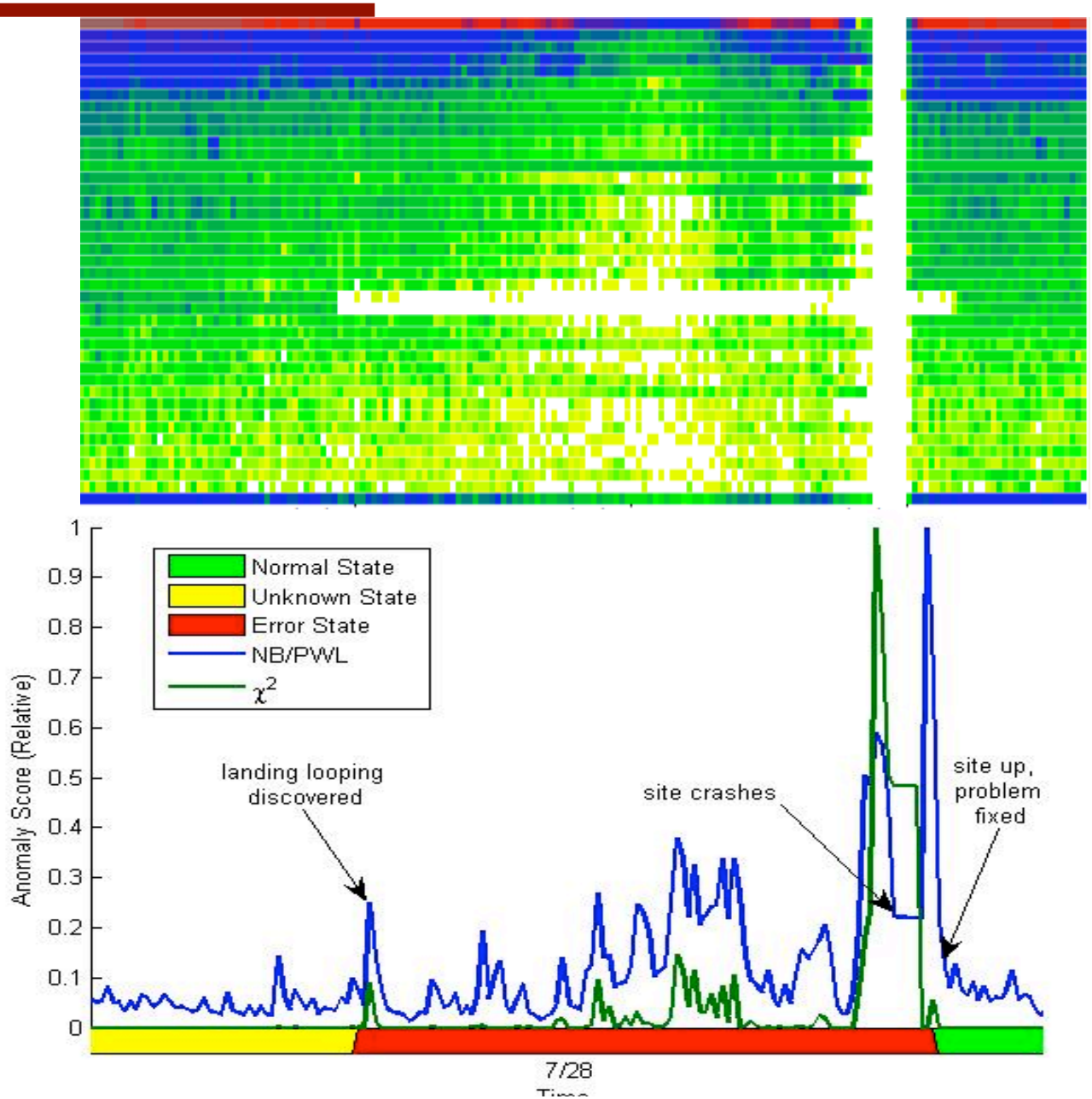
- Assumption: "most things work right most of the time"
  - Dealing with rare-but-legitimate behavior as false positive
- Done entirely in middleware

Visualizing & Mining User Behavior During Site Failures*

- Idea: when site misbehaves, users notice, and change *their* behaviors; use this as a "failure detector"
  - Quiz: what kind of learning problem is this?
- 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 χ^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*

* P. Bodik, G. Friedman, H.T. Levine (Ebates.com), A. Fox, et al. In Proc. ICAC 2005.

Example problem with page looping

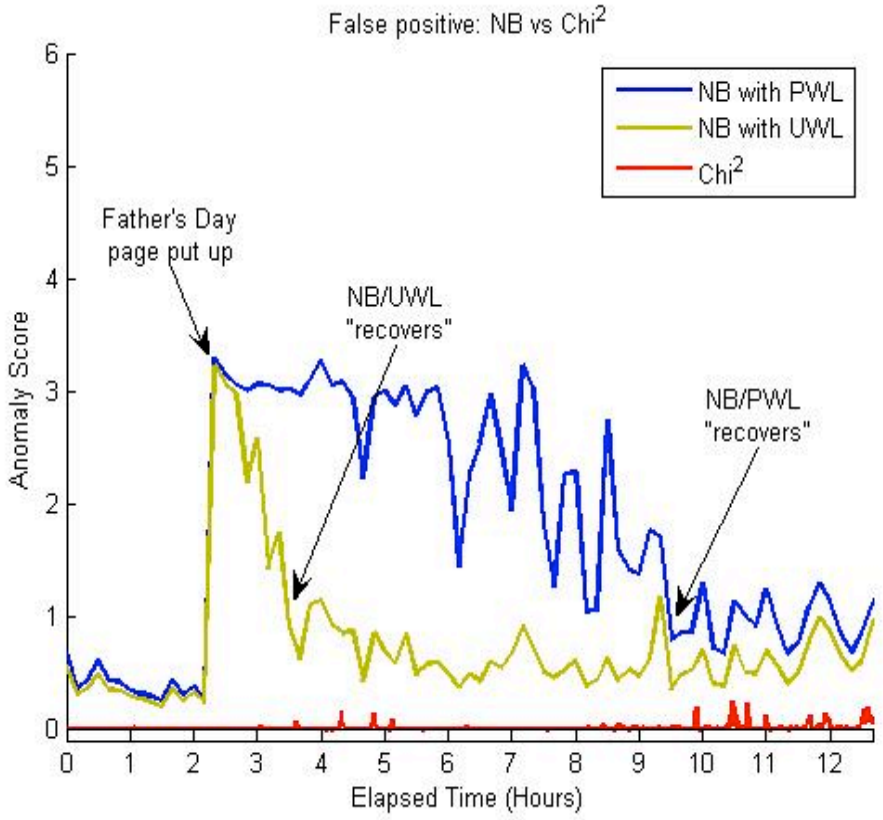


Potential Impact: Gaining Operator Trust

- Combining SLT 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 SLT
- Increasing operators' *trust* in automated techniques

Results and pitfalls

- Detected all anomalies in logs provided by real site, usually hours before administrators detected them
  - Including some that administrators *never* detected
  - *Ground truth determination presents a methodological challenge in real systems*
- "Eager" vs. "careful" learning
  - A long-lived anomaly or a new steady-state?
  - Fundamental challenge of *interpretability* of models
  - Another case for human intervention!



Fundamental Challenges

- Arise from *application* of SLT to systems, not techniques themselves
- **Validity** of induced models in dynamic settings
  - Models are being used to make inferences over unseen and *dynamically changing* data...how to evaluate their validity?
  - How many observations are required to induce new models?
  - Are thresholding, scoring, distance, etc. functions meaningful?
  - Supervised or unsupervised learning?
  - False positives/negatives will always be a fact of life
- Interaction with the **human operator**
  - Interpretability: mapping model findings onto real system elements
  - False positives: reduce cost through visualization and cheap recovery
  - Build trust of operator by combining visualization with SLT
- **Real data:** toward an "open source" failures & workloads database

ROC as Enabling Technology for SLT/ML

- Microreboot exemplifies "Repair as local adaptation"
  - Invariant: repair actions are *safe* and *low-cost*
  - safety achieved by state separation
  - State storage abstraction makes guarantees tuned to the needs of Web application state, and is itself crash-only
- Why the Web "works"
  - Web workloads help: request-reply means failure propagation distances are shorter (cf. *Failure-Oblivious Computing,* Rinard et al., OSDI 2004)
  - Web legacy helps: stateless protocol forced early developers to do separate state management
  - Separation of state => separation of recovery concerns (process vs. data)

What makes a 'good' architectural decision?

- Proposal: recoverability drives architectural decisions
  - Favor component decoupling over performance
  - Favor state segregation over performance
  - Favor more and narrower abstractions for state management
  - Favor machine-readable instrumentation and flexible control (not just "-verbose" mode)
- Architectures & frameworks can help enforce architectural decisions
  - Goal: separate "process recovery" from "data recovery"
  - J2EE gets some of these right, somewhat by accident

"The only problem of dependability is state management. All other problems inherit from it."

Hope

- Commercial frameworks *are* providing instrumentation hooks
  - e.g. developer API's in IBM Websphere XD, J2EE instrumentation API's
  - Opportunity for software architecture practitioners to move commercial frameworks in the right directions
  - **No excuse for academics to do research on "toy" platforms**
- Performance impact *is* tolerable
  - 10-30%, not "factor of X" degradations, in our experiments to date
  - We already accept other tradeoffs, why not performance for dependability/manageability?
  - Requires cultural change for *both developers and site operators*
  - **No excuse for avoiding adoption on the basis of performance arguments**
- Programmers are smart -- but they make human errors -- like a "commodity resource". **We have experience making allowances for commodity resources.**

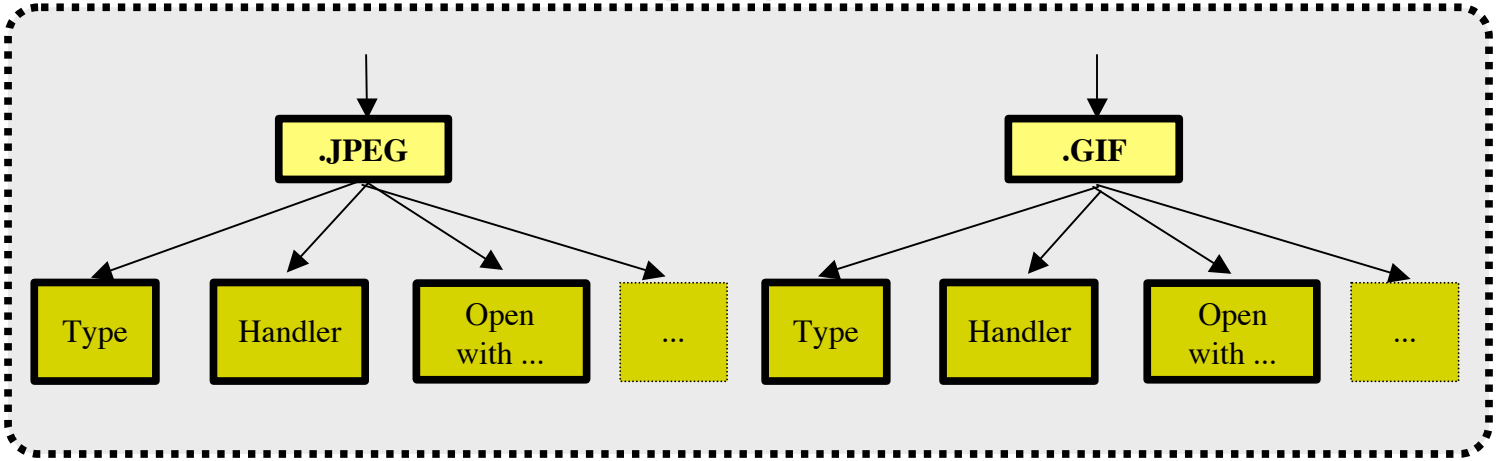
Acknowledgments

Papers at http://www.cs.stanford.edu/~fox

- Luke Biewald, George Candea, Greg Friedman, Emre Kiciman, Steve Zhang (Stanford)
- Peter Bodik, Michael Jordan, Gert Lanckriet, Dave Patterson, Wei Xu (UC Berkeley)
- Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons (HP Labs)
- Aaron Brown (IBM Research), Joe Hellerstein (IBM Research), HT Levine (Ebates.com), Yi-Min Wang (Microsoft Research)
- Research supported by NSF CAREER award, Microsoft, IBM, HP Labs, NSF ITR award, and NSF Fellowships

Example: finding Registry configuration errors*

- Idea: Many registry classes share common substructure
- Approach: use data clustering to learn these classes
  - Distance metric: number of common subkeys
- Then look for invariants over members in each class
  - Ex: "For a DLL registration, the only legal values for the DLLTYPE attribute are '16bit' and '32bit'"
  - Can be brute force for a "clean" registry, else thresholded



*E. Kiciman, Y.M. Wang et al., 2004 Intl. Conf. on Autonomic Computing