Recovery-Oriented Computing

Statistical Analysis & Systems:
Retrospective and Going Forward

Emre Kıcıman

emrek@cs.stanford.edu

Software Infrastructures Group
Stanford University

This talk

- *Quick* motivation
  - We all want to ski.
- Pitfalls and observations
- Future work

Situation

- Large, complex systems, changing rapidly: translates to systems we don't understand
  - e.g., Internet services, wide-area distributed systems, Internet
- Solid engineering helps!
  - but still poorly understood emergent behavior at large scale
- Further complications
  - Frequent HW & SW changes
  - buggy software, HW failures, human error

Problem: Hard to Manage

- Issue: How to keep these systems running?
  - Adapt to changing environment, performance, workload
  - ... leading to failures, performance issues, etc.
- Car analogy: Driving with magnifying glass
  1. Overwhelmed by details
  2. Unable to focus on important stuff at a distance
- Concrete ex: Drop in searches/sec at search svc
  - No connection b/w symptoms of failure and cause
  - Wade through low-level details of components to find problem

What's needed?

- Techniques to bridge low-level behaviors & controls with high-level requirements
- Must scale to complexity, size & rate of change
- *Non-goal*: taking humans out of the loop
- *Goal:* Allow people to concentrate on the high-level
  - Minimize minutiae of systems
  - ... and automate micro-management

Approach

- Use statistical analysis & machine learning to bridge the gap
  - Basic assumption: there **is** a relationship between low- and high-level. We just don't know what it is.
- Combine
  1. Lots and lots of observations of the system
  2. Weak assumptions about system
  3. Various statistical analysis & machine learning alg.
- ... to translate low-level observations into high-level descriptions

How's that gone so far?

- Failure detection
  - Better & faster failure detection through anomaly detection
  - ... in JBoss / J2EE, and clustered hash tables, 2 Int. Svc's
- Failure diagnosis
  - Tracking symptoms to possible causes through correlation
  - ... in JBoss / J2EE
- Inferring extra structure to help understand sys.
  - Using data clustering to recognize patterns in system
  - ... finding 'complex types' in Windows registry
  - ... finding equivalent nodes, etc. within an internet service
- *Take-away lessons coming up...*

Obs #1: High probability != Good

- System dominated by steady-state behavior
  - At short time-scales, steady-state is “very probable”
- But, many events necessary for correctness are **rare**
  - E.g., system initialization, garbage collection
  - ... probability estimates won't notice missing rare events
- Monitor rate of all events (including rare events)
  - Take into account more context
  - Increase time-scale, calculate probability of whole period

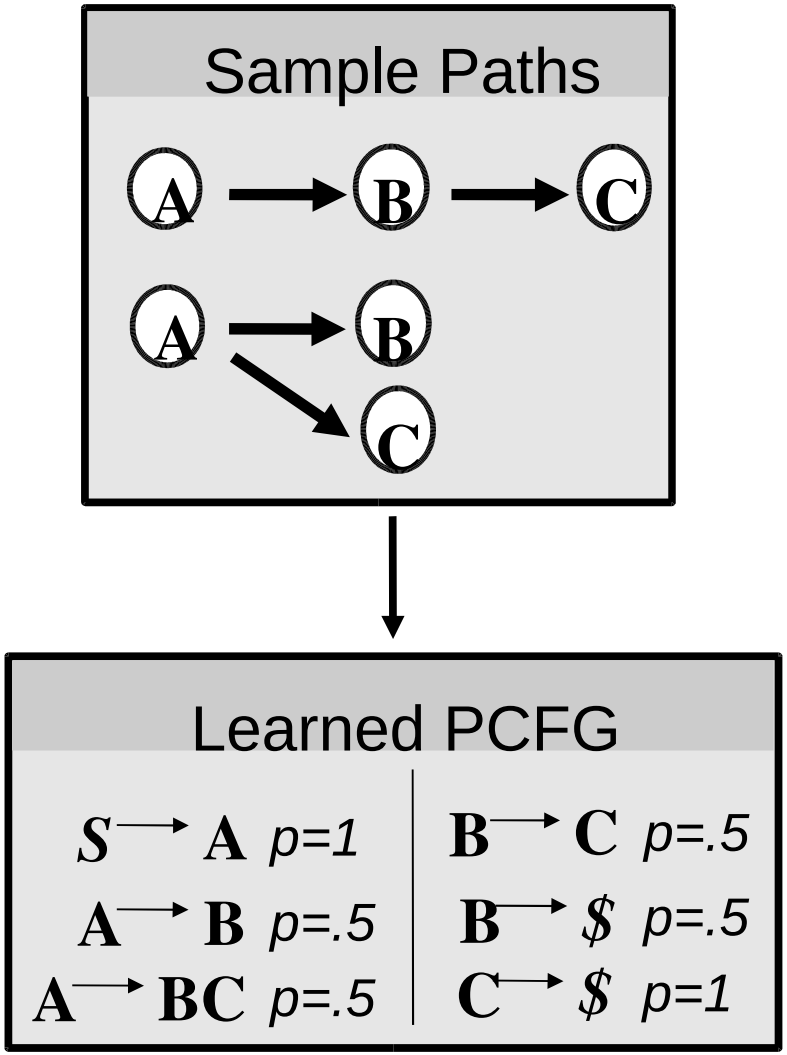
Obs #2: Low Probability != Bad

- Consider requests in a customizable Internet service
  - E.g., portals aggregate data from 10s or more of mini-services onto a single page
  - Many permutations of customizations
  - Most combinations are low-probability, but valid
- Probabilities of observed request behaviors
  - ... will correspond to user customization prefs
  - ... and not to system failures
- Split up request's behavior and analyze pieces
  - In each piece, discount contribution of low-probability events by expected probability

Obs #3: Watch your Biases (1)

- General principle, applies to SLT+Systems
- Algorithms have biases
  - What's appropriate in other domains, may not be appropriate in systems
- Example: We used probable context free grammar (PCFG) to model request behavior in system

Scoring w/PCFG: Take 1



- Probability-based scoring
- Score s of a new path is:

$$s = 1 - \Pi_{t \in transition} P(t)$$

s (A → B → C) =

$$1 - P(A \rightarrow B) * P(B \rightarrow C)$$

Scoring w/PCFG: Take 1 didn't work

s (O→O→O→O→O→O→O→O→O)=

1 - P(A→B)*P(B→C)*P(C→D) *
... * P(G→H)

... s approaches 1 with long paths

- Bias against long paths!
  - OK in natural language
  - Not ok in systems

Scoring w/PCFG: Take 2

- Another NLP technique: use geometric mean
  - avoids bias against long paths

s (O→O→O→O→O→O→O→O→O)=

$$1 - [\ P(A \rightarrow B)*P(B \rightarrow C)*P(C \rightarrow D)\ * \ldots * P(G \rightarrow H)\ ]^{1/9}$$

- But hides anomalies if they don't affect many transitions

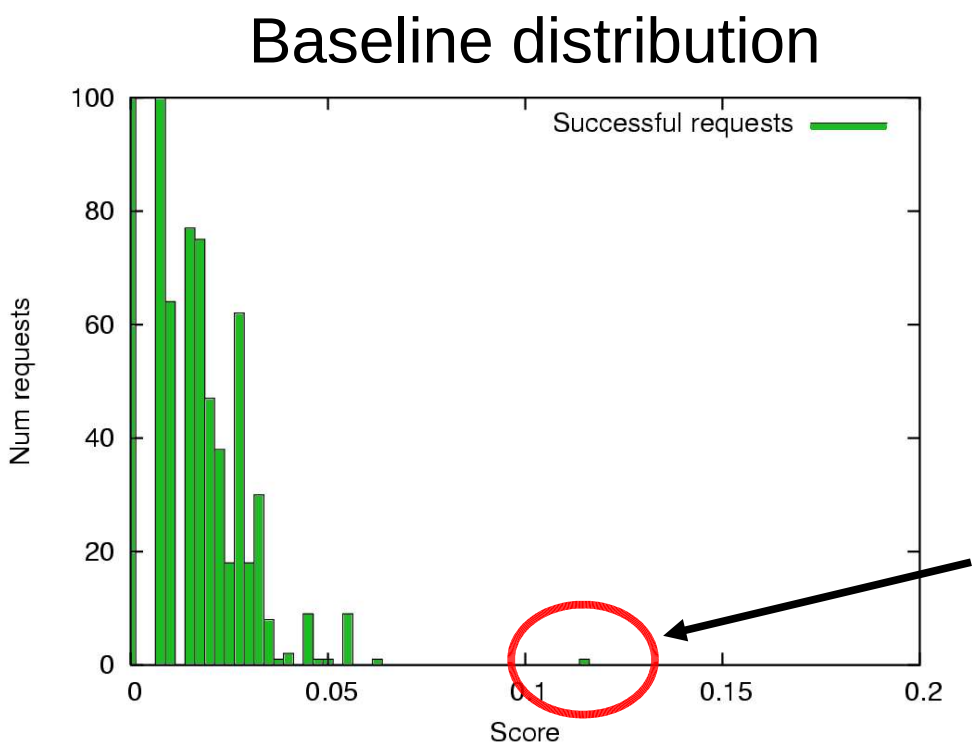
Scoring w/PCFG: 3rd time's the charm

- Sum the deviation between expected and observed probabilities at transitions
  - Sum instead of product to reduce bias against long paths
  - Uses “low-probability != bad” observation (pitfall #2)

$$s = \sum \min(0, 1/n_i - P(t_i))$$

  - n_i = # possible transitions from a component
  - $P(t_i)$ is the observed probability of a specific transition
  - Sum over all transitions in test path

Obs. #4: Imperfect system OK for training



- Learn "correct" behavior
  - but, anomalies always exist
- Will baseline anomalies mask problems?
  - Found: Only "most of system" must be correct
  - **not** "all of system"
- Rare path (1/1000) marked as anomalous
  - even though it is in baseline

Obs. #5: Prepare for imperfect analysis

- Analysis will sometimes be wrong
  - Algorithmic errors: statistics wrong some % of the time
  - Semantic errors: when assumptions don't hold

1. Cross-validate to filter out mistakes
   - e.g., in end-to-end autonomous recovery, we check that system is not already recovering
   - e.g., If **everything** is failing... maybe detector is wrong

2. Tolerate mistakes that slip through
   - Respond with a safe, fast action to reduce cost of mistakes [microreboots]

(if mistakes are OK, try being more aggressive)

Recap Take-aways

- In fault detection: high- & low-probability don't always correspond to good & bad
- Check algorithms for biases
- Imperfect system OK for training
- Prepare for imperfect analysis
  - Double-check assumptions and cross-check results
  - Trigger only cheap, safe responses

Looking Forward

- What's next?

1. Expand to more varieties of systems
   - Similar problems across wide variety of systems & networks
   - Capture commonalities, characterize differences
   - Collaborate and share solutions

2. Apply SLT to more systems management tasks
   - Automating micro-management (reinforcement learning)
   - App description / Policy specification

“Root cause” localization*

- Similar problem across at least 3 domains
  - Fault localization in Internet services
  - Root cause analysis of BGP dynamics
  - Bug isolation
- Abstract model 'localization' across these systems
  - Transformation from real system to model captures differences
  - Allows sharing of algorithmic approaches
  - ... comparison of trade-offs and assumptions
  - (hopefully) better collaboration across domains & w/SLT

* *Root cause localization in large scale systems*, Kıcıman & Subramanian. Tech report, 2004.

Policy Specification

- “we'll let [someone else] decide what policy makes sense”
- Just 1 example: where to deploy wide-area apps
  - e.g., SETI, dynamic Akamai, others
  - What resources are needed? (CPU/disk/mem/network)
  - Now: let developer specify needs... (but varies with workload & environment)
- How can we automatically find resource needs?
  - Deploy 100s across wide-area (randomly)
  - Measure performance & correlate to features of nodes
  - (Migrate poor performing nodes to better suited ones)
- Challenges
  - measure performance, factoring out workload, cheap migration, ...

Automated Micro-management

- Environment, workload change frequently
  - Constantly tuning system to adapt & maintain performance & correctness
  - (fault recovery process “just” one adaptation)
  - Low-level knobs far-removed from high-level goals
- Linear control theory & reinforcement learning
  - Based on model of system, predict how to tune knobs to improve performance
- Challenge: do no harm
  - Validate actions after-the-fact
  - Detect when outside of safe region, hand-off to operator

Summary

- Statistical analysis + systems
  - Simplify, improve admin, reliability
  - Automatic analysis → handles complex systems
  - Fast training → scales to frequent system changes
  - First round of work promising, learned important lessons
- Plenty of future work
  - More systems, common problems, sharing solutions
  - More tasks requiring rapid adaptation, detailed understanding of system minutiae

Thanks

- Questions?

Emre Kıcıman, emrek@cs.stanford.edu