Research and Teaching Statement

Armando Fox

February 2008

Research Summary

My research focuses on both policy and mechanism for managing datacenter-scale installations (thousands of computers) of interactive-response, Internet-resident services. “Management” includes recovery from failures, on-demand scalability, dynamic resource allocation, and improved power efficiency. The *mechanism* for carrying out these tasks is to construct software building blocks in which common operations, such as failure recovery, scaling up/down, or reprovisioning, can be achieved by rebooting a machine (or its dual, adding a new machine and killing the faulty one). The *policy* is based on the use of statistical machine learning (SML) techniques to automatically identify and react to problems that would take too long for a human operator to diagnose manually. The ideal of 99.999% service availability corresponds to just 5 minutes of service downtime per year, which cannot be achieved if humans must participate in every operational decision [2].

Two main themes of my previous work on Recovery-Oriented Computing (ROC) have influenced the design of recent commercial and research systems. The first is the design stance of crash-only software [3]: since robust software must survive unexpected crashes anyway, the crash recovery code should be the *only* recovery code, and any non-crash problem (slowdown, anomalous behavior, etc.) observed during operation should be immediately coerced to a crash failure. This is a radical design simplification that allows focusing on optimizing the performance of the one and only recovery path. The second theme is exploiting this fast recovery by applying SML problem detection techniques that, while more sensitive than non-SML state-of-the-art methods, have nontrivial false positive rates: the observation is that because of the low cost of recovery, overall availability may still improve from using SML, despite false positives.

Engineers and researchers at Amazon, Oracle, eBay, Microsoft and Google have told us they were strongly influenced by the demonstration of these techniques, and Aster Data Systems (founded 2005) is designing its parallel clustered database as crash-only from the ground up. Hewlett-Packard is already putting some of the SML problem detection and diagnosis technology into its system monitoring products. The combination of SML for analyzing log data and visualization to draw the human operator’s attention to interesting patterns in the data was demonstrated on real failure log data from Ebates.com and remains an area of active research.

Previous Work: Lessons from Recovery-Oriented Computing

Recovery-Oriented Computing (ROC) [2] observes that a service whose mean time to failure is MTTF and whose mean time to recovery from failure is MTTR experiences an availability $A = \text{MTTF}/(\text{MTTF}+\text{MTTR})$, with $A = 1$ (i.e. $MTTF >> MTTR$) corresponding to the ideal of zero downtime. Reducing MTTR is just as effective as increasing MTTF to improve availability, and is an under-explored research approach despite being more consistent with the practical experience that failures and bugs will continue to be a “fact of life” rather than a problem that can be completely eliminated.

The ROC lesson was that a sufficient reduction in recovery time enabled the use of SML for problem determination in novel ways, detecting problems that do not generally lead to “hard failures” and are therefore often missed by traditional techniques. To demonstrate the potential of SML, we applied path-based analysis, a family of techniques from the natural language processing literature, to the detection of failures in Java enterprise (J2EE) applications. The technique required no source code changes to or other knowledge of the application. Path-based analysis was found to be 1.5× to 4× more sensitive than existing techniques [9, 5], but it exhibited false positive rates of up to 20%. However, we had equipped our J2EE application server with our *microreboot* capability [4], which allows restarting only certain parts of a failing Web application rather than the entire application, reducing recovery time by 1–2 orders of magnitude for many common transient failures. With recovery so inexpensive, the *overall* availability of the application in this scenario improved by 53%. Path-based analysis detected and recovered from problems that would have been missed by other techniques, and the cost of its false positive rate was outweighed by the benefit of extremely fast recovery (microrebooting).

After demonstrating the success of combining microrebooting with machine learning for stateless application servers, we next demonstrated its feasibility for persistent storage systems by building two special-purpose prototypes for storing Web application data [8, 10]. By using quorums and relaxing consistency, we were able to design these systems to tolerate crashes of any machine at any time with no data loss and minimal performance loss, and with all provisioning and maintenance operations recast as rebooting or adding/subtracting machines. We concluded that if recovery is sufficiently cheap, it leads to a qualitative change in thinking from "normal-mode vs. recovery-mode" to "always adapting, always recovering". In other words, while "Reduce recovery time to improve availability" and "build systems to be reboot-safe" may amount to codification of sound design practices, their *combination* has been instrumental in bringing Statistical Machine Learning techniques to bear on systems operational problems.

While path-based analysis was a first step in applying SML to systems problems, we next pushed the state of the art by attempting to reduce problem diagnosis to information retrieval [11, 12]. Our idea was to identify those specific measurable aspects of a running system that were highly correlated with undesirable system behavior, such as violation of its service-level performance agreement, over short time intervals. By capturing the most important measurements as a "signature" of 3 to 8 low-level metrics within each window, we could maintain a database of "signatures" of known problems and essentially model the system as going through a sequence of operational states captured by their respective signatures. When a new problem occurred, we would compute its signature and use classic information retrieval techniques and metrics to compare it to the signatures of known problems in our database. When tested on a real workload containing partially-labeled and some incorrectly-labeled training data, we found one real problem missed by weeks of human diagnosis, and corrected an expert's misdiagnosis of another problem in a similar system.

Ongoing & Future Work: Extending the Lessons of ROC

My current research agenda aims to extend and generalize the ROC lessons—combining fast recovery with SML-based analysis and detection [6]—in the design of datacenter-scale software subsystems. The crash-only approach, in which many common operations are recast in terms of crashing, removing or adding a machine, is an ideal fit for "hardware-as-a-service" environments such as Amazon's Elastic Compute Cloud (EC2), in which the incremental cost of leasing an additional (virtual) machine is near zero ($0.10–$0.80 per hour in 2008).

Applying SML to Systems Problems

Predicting resource utilization. My current work, also with HP, takes the problem of predicting resource utilization of long-running database queries using query workload features, and maps that problem onto an instance of a Kernel Canonical Correlation Analysis (KCCA) problem. While KCCA is a recent and fairly complex SML technology, we found that simpler SML methods such as regression do a poor job of prediction, motivating investigation of KCCA.

To our knowledge, no previous work attempts to predict the actual performance of a multi-query database workload. Using a real customer workload, our model predicts individual query running times to within 20% for over 85% of queries, outperforming a state-of-the-art commercial predictor and achieving $R^2 \geq 0.95$ in simultaneously predicting utilization of multiple resources [7]. On investigation, a main reason we outperform the commercial predictor is that the cardinality estimation errors that affect conventional predictors are "normalized out" by our SML-based prediction process. Thus we believe our approach represents a fundamental advance over other performance prediction methods. Ongoing work includes applying this approach to very large MapReduce workloads and interactive Web applications. In addition we plan to use the KCCA models to drive synthetic workload generators. This would allow researchers to use realistic workload data synthesized from the KCCA models of real commercial workloads, without having to obtain sensitive or proprietary workload data directly.

Power management. Datacenter operators are interested in saving power only if there is no risk of violating the SLA (e.g. due to slower performance from being in a lower-power mode). Our goal is therefore to construct

SML models that predict performance (i.e. SLA compliance) based on resource utilization *and* power state, allowing us to put parts of the system into a lower-power state without violating the SLA. Early results using nonlinear quantile regression [13] show that we can keep the CPU in a low-power state for a higher percentage of the time than the CPU's built-in power management policy (AMD PowerNow) while triggering few or none of the SLA violations caused by the built-in policy. Our eventual goal is that a collection of such analysis tools would inform the decisions of a "Datacenter Director" making global policy within the datacenter, in contrast to most current approaches in which components manage their own power and often end up working at cross-purposes.

SML as a Technology. In both previous and ongoing work, we consistently find that straightforward, naive approaches to problem detection gave poor results, motivating the investigation of more sophisticated SML techniques and algorithms. In the signatures work we demonstrated the need to use tree-augmented Bayes networks, which are more sophisticated representations of conditional distributions than Naive Bayes; we also found that a single simple model failed to capture the relationship between individual system performance metrics and overall SLA compliance/violationmanagement of models' lifecycles, model and process stationarity, thresholding/scoring/distance functions, dealing with false positives, and the challenges of combining supervised with unsupervised learning. The RAD Lab environment, mission and associated courses in progress are uniquely suited to train this next generation of "crossover" researchers, and working towards a collaboratively-built artifact such as the Datacenter Director is a way to keep focus on the relationships among the areas of research.

Scalable Crash-only Storage

While relational algebra created a revolution in data management and a new industry around relational database management systems (RDBMS's), there is little disagreement that today's Web applications have different needs. The ACID (atomicity, consistency, isolation, durability) guarantees provided by RDBMS's are stronger than needed for most Web applications, and fully-general relational queries are more expressive than needed by many Web applications; yet the engineering required to combine those properties in conventional RDBMS's means that they scale less and cost more than special-purpose storage systems that sacrifice one or more of the properties. For example, Amazon's Dynamo and Google's BigTable sacrifice one or more of these properties in order to achieve far greater scale and higher throughput than any existing RDBMS, and even our ROC storage system prototypes relaxed ACID to facilitate crash-only design and SML-based automated monitoring.

While many "one-off" specialized storage systems have been built, each requires rewriting the application to use the storage system, which explains the longevity of SQL as an implementation-independent abstraction for describing operations on stored data. With SCADS (Scalable Consistency-Adjustable Document Store), we are working with Facebook, the Internet Movie Database, Amazon, and eBay to capture use cases for their large-scale distributed databases, with the goal of developing both a formalism comparable to SQL for reasoning about such applications' storage needs and a prototype of a "SCADS engine" that can scale to 1,000 machines on Amazon EC2. We see an opportunity for a new abstraction with the advent of Ruby on Rails, whose Active Record middleware layer provides an object-graph model that fits the needs of many Web applications. Given the uptake of Ruby on Rails, a new abstraction that is near-compatible with Active Record would be much less disruptive than a completely new programming model. We believe a formalism is needed in which to ground this abstraction, both because it would facilitate the kinds of optimizations that today's query optimizers perform on SQL queries (by applying relational algebra transformations) and because it could provide an implementation-independent specification for building future consistency-adjustable storage systems.

Helping the Operator

As observed by ROC, in many systems the largest single contributor to downtime was human operator error or lack of proper tools to help diagnose a problem that could not be addressed automatically [1]. To create better tools, we used SML to pre-analyze data to draw the operator's attention to unusual patterns, combined with visualization techniques that exploit the built-in parallel processing of the human visual system. The combination helps operators quickly spot problems in large data sets and "grounds" their understanding of how the SML algorithms work, leading over time to increased trust in the automated algorithms. Ongoing work in this area includes combining text mining of applications' console logs, analysis of the source code, and visualization, to help spot rarely-occurring patterns or events in the logs that might be indicators of a failure or provide useful forensic evidence in tracking down intermittent failures. For our initial efforts we are using real console logs from a Java-based production search engine. In one instance our prototype helped identify the cause of one bug that took weeks of manual debugging. In another instance text mining of the logs would have focused human attention on the subsystem containing the actual bug, whereas in the absence of this information the operators' intuition had led him to focus attention on a different subsystem that turned out not to be faulty. We are in the process of applying this to other large-scale back-end services such as text search and extending the techniques to languages other than Java.

Teaching

I strive to inspire passion through teaching by connecting the material both to cutting-edge research and to the "real world". While I have taught core courses in systems, software and architecture, below are some courses I think have had particular impact.

For graduate students: Graduate-level project courses, including CS241 (Internet Services) at Stanford and various CS294's at Berkeley, have exposed graduate students to high-profile "technical advisors" for their research projects and have resulted in about a dozen refereed conference papers and at least three Ph.D. theses and numerous Masters' theses.

For non-CS undergraduates: My freshman seminars "History of Computing & Communication" (originally developed with Prof. Randy Katz) and "Digital Dilemmas", which explores the impact of technology on social policy issues such as digital rights management and e-voting, expose students to the larger impact of CS in the world and help to recruit curious students into the major. Successful seminar alumni who don't choose CS often enter political science, law or economics with a much deeper grasp of technology issues than their colleagues; recent examples include one Marshall Scholar and one serial entrepreneur who has become a respected "financial advice giver" to his generation of peers.

For CS undergraduates: In my pilot course in Web 2.0 development using Ruby on Rails, sophomores and juniors develop complete Web applications of their own design at an early stage of their careers guided by professional developers as well as faculty. Even though many students in the class have no prior Web development experience, most projects reach an impressive level of quality in just eight weeks, with many having outside customers. Besides leveraging undergraduates' natural enthusiasm for entrepreneurship and creativity as an opportunity for good CS pedagogy, the course exposes students to best practices and tools for professional software development, basic project planning, and team skills, making them highly sought after by industry (and by the RAD Lab—I have hired three alumni so far as undergraduate research assistants). A positive side effect is that the effort to provide a "realistic" computing environment for the course has led to collaboration with Berkeley IS&T and Shel Waggener's office that will ultimately propagate these improvements University-wide.

While I plan to package this course as a self-paced course and/or transition it to an extracurricular "programming club" similar to lecturer Dan Garcia's successful undergraduate clubs, I also plan to develop the course into a CS194 (with Paul Hilfinger) and ultimately a regular curriculum offering. Software technology trends in the last 10 years mean that CS undergraduates must be *conversant* with a broader range of technologies earlier in their careers, essentially gaining breadth earlier and with fewer courses. For example, whereas relational database technology previously was a specialty area in CS, today *all* software/systems students must be familiar with basic concepts and tools in this area. The same is true of networking, cluster computing, concurrent programming, and human-computer interaction. I believe the Ruby on Rails course can serve as the basis for a more in-depth Web software development course that ties these areas together.

Professional Responsibility

Academia is fun. We get the best students, and mentoring them through research, teaching, advising, and general life skills is not only rewarding but also probably the surest path to lasting impact. Having said that, academic professionals' responsibility (paraphrasing Paul Hilfinger) extends to every respect in which higher education can benefit humanity, but with a strong emphasis on research and mentoring in our individual fields. I believe Berkeley sets a good example in this regard, and I look forward to further developing those ancillary career areas beyond my University research and teaching.

References

[1] A. B. Brown and D. A. Patterson. To err is human. In *Proceedings of the 1st Workshop on Evaluating and Architecting System Dependability (EASY)*, Göteborg, Sweden, July 2001. IEEE Computer Society.

[2] G. Candea, A. Brown, A. Fox, and D. Patterson. Recovery-oriented computing—designing multi-tier dependability. *IEEE COMPUTER*, 2005. Invited submission; to appear.

[3] G. Candea and A. Fox. Crash-only software. In *Proc. 9th Workshop on Hot Topics in Operating Systems*, Lihue, HI, June 2003.

[4] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot—a technique for cheap recovery. In *Proc. 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)*, San Francisco, CA, Dec. 2004.

[5] M. Y. Chen, E. Kıcıman, A. Accardi, E. A. Brewer, D. Patterson, and A. Fox. Path-based failure and evolution management. In *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, March 2004.

[6] A. Fox, E. Kıcıman, D. Patterson, R. Katz, I. Stoica, and M. I. Jordan. Combining statistical monitoring and predictable recovery for self-management. In *2004 ACM SIGSOFT Workshop on Self-managed Systems (WOSS'04)*, Newport Beach, CA, October 2004.

[7] A. Ganapathi, J. Wiener, D. Patterson, M. I. Jordan, and A. Fox. Predicting query and workload performance for very large data warehouses. In submission, Feb 2008.

[8] A. C. Huang and A. Fox. Cheap recovery: A key to self-managing state. *ACM Transactions on Storage*, 1(1), 2004.

[9] E. Kıcıman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks (special issue on Adaptive Systems)*, Spring 2005.

[10] B. Ling, E. Kıcıman, and A. Fox. Session state: Beyond soft state. In *Proc. 1st Symposium on Networked Systems Design and Implementation (NDSI'04)*, San Francisco, CA, March 2004.

[11] S. Zhang, I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *Proc. 20th ACM Symposium on Operating Systems Principles*, Cambridge, UK, 2005.

[12] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. In *2005 Intl. Conf. on Dependable Systems and Networks (DSN 2005)*, Yokohama, Japan, June 2005.

[13] P. B. k, C. Sutton, A. Fox, D. Patterson, and M. Jordan. Response-time modeling for resource allocation and energy-informed slas. In *SysML'07*, Vancouver, BC, Dec 2007.