

RAD Lab Research and Education Highlights: Using Statistical Machine Learning to Help Operate Cloud-Scale Datacenters

Armando Fox

Updated May 2011

The RAD Lab (Reliable Adaptive Distributed Systems Laboratory) was launched in January 2006 as a five-year effort with the mission statement: “Enable one person to develop, deploy and operate a next-generation Internet application at scale, without first having to build a company on the scale of eBay or Google to do so.” Given the scale of contemporary Internet services (thousands of machines, tens of millions of users) and the timescales on which management actions would be required to maintain the high responsiveness users have come to expect on a 24x7 basis, a key element of the RAD Lab vision is the use of Statistical Machine Learning (SML) techniques to facilitate all aspects of developing, deploying and operating such applications. The interdisciplinary faculty represented expertise in systems, networks, and machine learning. Most of my PhD students were advised jointly with Dave Patterson, Mike Jordan or Mike Franklin, and much of the work was done in close collaboration with one or more industrial partners of the RAD Lab.

The five-year meta-contributions made by this subgroup of the lab can be summarized as:

- Bring statistically sound, state-of-the-art SML techniques to bear on the above problems, yet “package” them so as to be readily usable by non-experts in SML.
- Produce a generation of reserachers cross-trained in systems and SML, publishing in both sets of venues and gaining visibility for contributions to both communities, to bolster our position that SML should become just as important a tool in the system builder’s toolbox as, say, queueing theory.
- Use SaaS and cloud computing as a vehicle to improve undergraduate software education.

The following project highlights capture specific research and educational contributions.

AWE: performance prediction as correlation analysis

Automatic Workload Evaluation focused on the use of a less-than-decade-old SML technique, kernel canonical correlation analysis (KCCA) [3], as a different approach to problems involving performance prediction. The insight is that both offered workload and measured performance can be captured as multi-dimensional vectors; for example, database queries can be characterized by a vector capturing the number and types of operations in a query plan, and performance of a query by a multidimensional vector whose elements include running time, number of interprocessor messages on parallel hardware, number of disk I/Os, etc. A similarity kernel function can then be defined over each dataset; this is necessary because simple geometric differencing of the vectors may fail to capture information the system designer believes is important in measuring the similarity between, say, the structure of given pair of queries. It then remains to find a basis of correlation between these two sets of differences, which the KCCA algorithm does. We applied this method to predict the running time and resource consumption of long-running database queries [10], achieving accuracies beyond those of the built-in query planner. This success led us to apply a similar technique to identify the most promising autotuning parameters for compiling code on multicore processors [9] and for predicting the performance of large Hadoop (map/reduce) cluster jobs based on the sampled performance of a smaller problem [8].

Core parts of the AWE work were done in conjunction with HP Labs.

Combining SML and program analysis for console log mining

Free-text console logs (`fprintf(STDERR,...)` or `console.write(...)`) are ubiquitous in large software systems, yet they are not machine-friendly because of their free-text format and they are not human-friendly because the volume of log messages is so large that finding “interesting” patterns manually is impossible. A better approach than ad-hoc shell scripts is Console Log Mining: combining techniques from SML (data mining, anomaly detection, principal component analysis), information retrieval (bag of words analysis), and static program analysis (basic type inference) to distill millions of lines of free-text unstructured logs to concise operator-friendly visualizations of operational problems or interesting conditions.

The key idea behind the work is to look not for specific events, but sequences of events that are unusual. Principal component analysis (PCA) is used for this step. However, PCA requires features to operate on, and in programs, the features of interest tend to be entities such as disk block numbers, file descriptor values, integers or strings indicating progress through a state machine (`file_opened`, `file_changed`, `file_closed`, ...), and so on, and in console logs, these

identifiers are buried in free-text log messages in unsystematic ways. To recover the underlying “schema” of the logs and successfully extract these features, source code parsing is used to infer all messages that could appear in the log, and type inference and abstract syntax tree traversal on the log printing statements is used to determine which parts of the message correspond to identifiers that could become a feature. This comprehensive parsing technique finds identifiers that are missed by existing analysis methods and without which specific problems would not be revealed by the subsequent PCA analysis. The final step is to present a digest of the results in an operator-friendly decision tree indicating the correlations between the presence or absence of certain sets of messages and the existence of an “unusual” condition.

We developed both an offline method [12] and later an online method based on Frequent Pattern Mining [13], validating all results on millions of lines of open-source applications (Hadoop, Darkstar) using cloud computing and finding both new bugs and existing inconsistencies between the log messages and actual operation that had been the subject of much confusion on the developer blogs for those applications.

The techniques were developed in collaboration with Intel Research, and have recently been applied to a production application at Google [14]. The work is also the subject of an invited presentation at ICML 2010.

Understanding spikes

The huge number of users now online (500 million for Facebook alone; 3.5 billion who have access to a cell phone) means unprecedented potential not just for workload spikes, but data hotspots. On the day of Michael Jackson’s death, so many Google searches were directed to Jackson-related pages that Google thought it was experiencing a denial-of-service attack. On that same day, over 20% of all “tweets” (messages on the Twitter social-messaging site) were related to Michael Jackson.

Dealing with such spikes and hotspots requires sound yet tractable and easy-to-work-with models of such behavior. We have brought our machine learning and statistics expertise to bear in creating such models based on a number of real spikes observed in the wild [5]. Our seven-parameter model captures not only spike volume and steepness, but skewness of distribution to data accesses (i.e. it can model hotspots). In particular, we observed that contrary to widespread belief, hotspot distributions do follow a power law, but the Zipf distribution does not capture them accurately. One component of our spike model uses two techniques from SML—the Chinese Restaurant Process and the stick-breaking process—to more accurately capture the hotspot behavior of real observed spikes. We also provide a workload generator that can be used to generate workloads in accordance with our model, and validate the generator against the real spikes we observed.

Elastic Structured Storage for Datacenter-Scale Applications

A common tale of fast-growing Web sites, including eBay, Facebook and others, is the need to rebuild the storage layer of the site as the site grows. Relational databases have long been the staple technology for persistent data in Web services, but the *performance opacity* of declarative query languages such as SQL complicates scaling: it is very easy to write an unaffordably-expensive SQL query that should not be allowed in an interactive-response setting, because it is too slow or too resource-intensive. Yet the alternative is hardly better: directly coding database query plans in terms of key/value operations makes performance explicit, but undoes the decades of progress of declarative relational queries.

The SCADS system (Scalable Consistency-Adjustable Data Storage) [1] attempts to provide the best of both worlds. By using compile-time analysis on queries to be performed, automatically analyzing which indices are needed to perform updates, and using machine learning techniques to estimate the cost of performing the overall query based on the underlying query plan, SCADS aims to provide an elastic yet SLO-compliant solution for structured data storage that “looks and feels” much like traditional relational databases. The Performance-Inightful Query Language, PIQL [2], allows only performance-safe queries to be expressed: those queries whose cost per user will not increase as the number of users increases. SCADS relies on an underlying key/value storage layer that can maintain relatively stable performance for low-level operations such as Get and Put; while many are available, no existing ones are designed to take advantage of the elasticity of cloud computing by gracefully scaling down as well as up. To that end, we are also working on an elastic storage layer that allows the storage system to scale up and down while complying with strict service-level objectives [11]. The idea is to use machine learning to model the costs of scale-up and scale-down operations and use model-predictive control to drive a policy that decides when to do this. Early results suggest that a single set of mechanisms can be used to harness elasticity both to deal with spikes (generating workload spikes using the techniques described above) and to save money during regular diurnal variation in usage, all while maintaining SLO compliance at a level of two nines or better over typical time intervals (e.g. 99% of requests complete within a tight latency bound during any 10-minute interval). We expect to report on this work in Fall 2010.

Fingerprinting the Datacenter

Contemporary datacenters comprise hundreds or thousands of machines running applications requiring high availability and responsiveness. Although a performance crisis is easily *detected* by monitoring key end-to-end performance indicators (KPIs) such as response latency or request throughput, the variety of conditions that can lead to KPI degradation makes it difficult to select appropri-

ate recovery actions. We proposed and evaluated a methodology for automatic classification and identification of crises, and in particular for detecting whether a given crisis has been seen before, so that a known solution may be immediately applied. Our approach [7] is based on a new and efficient representation of the datacenter's state called a *fingerprint*, constructed by statistical selection and summarization of the hundreds of performance metrics typically collected on such systems. This work was a major refinement of much earlier work on metric selection [15] and benefited from a fresh look at metric selection techniques for such scenarios [6], results that were already in use at Microsoft to decide what forensic telemetry to store permanently. Our evaluation uses 4 months of trouble-ticket data from a production datacenter at Microsoft with hundreds of machines running a 24x7 enterprise-class user-facing application. In experiments in a realistic and rigorous operational setting, our approach provides operators the information necessary to initiate recovery actions with 80% correctness in an average of 10 minutes, which is 50 minutes earlier than the deadline provided to us by the operators. To the best of our knowledge this is the first rigorous evaluation of any such approach on a large-scale production installation.

This work was performed in collaboration with Microsoft and Microsoft Research Silicon Valley.

Education

Early in the project, we decided we needed to train undergraduates to create interesting applications to showcase our infrastructure. Based on talking to our research and professional contacts, we quickly settled on Ruby on Rails as the “datacenter programming framework” that we could adapt to talk to our infrastructure. Through four iterations of teaching this course, not only were we amazed at the quality level of Web applications that undergraduates could produce in eight weeks (even with no prior Rails or web programming experience), but we also realized this was a great vehicle for teaching good software development practices such as user stories, behavior-driven design, test-first development, testing discipline, and effort estimation. We have since evolved the course into the upper-division Software Engineering course, where it has been enriched with software engineering fundamentals such as design patterns and discussion of software architecture and continues to be successful. We are packaging the course materials for technology transfer to other schools. I have been leading this effort and am likely to lead the creation of a textbook around this version of the course.

Final Demo

In February 2011, a successful integrated demonstration of all the RAD Lab artifacts was presented (http://radlab.cs.berkeley.edu/wiki/End_of_Project_Symposium).

Among other demonstrations, The RAIN workload generator [4] (now an open source project at <http://github.com/yungsters/rain-workload-toolkit>) was used to ramp up workload to an equivalent of 6000 Web requests per second harnessing hundreds of nodes on Amazon’s Elastic Compute Cloud, scaling up and down under the control of the Director. The applications serving the workload were developed by undergraduates using a modified version of Rails backed by SCADS with PIQL queries rather than ActiveRecord with SQL queries. In keeping with the RAD Lab’s original mission statement “allow a single developer to prototype an idea over a long weekend and scale it up to millions of users,” one of the applications was indeed developed over the Presidents’ Day weekend in 2011 (18 hours of work) by one undergraduate who was an alumnus of our course (above).

References

- [1] Michael Armbrust, Armando Fox, David A. Patterson, Nick Lanham, Beth Trushkowsky, Jesse Trutna, and Haruki Oh. SCADS: Scale-independent storage for social computing applications. In *CIDR 2009*, 2009.
- [2] Michael Armbrust, Nick Lanham, Stephen Tu, Armando Fox, Michael J. Franklin, and David A. Patterson. The case for PIQL: A performance insightful query language. In *First ACM Symposium on Cloud Computing (SOCC 2010)*, Indianapolis, IN, June 2010.
- [3] Francis Bach and Michael Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.
- [4] Aaron Beitch, Brandon Liu, Timothy Yung, Rean Griffith, Armando Fox, and David A. Patterson. Rain: A workload generation toolkit for cloud computing applications. Technical Report EECS-2010-14, University of California, Berkeley, Berkeley, CA, Feb 2010.
- [5] Peter Bodik, Armando Fox, Michael Franklin, Michael Jordan, and David Patterson. Characterizing, modeling and generating workload spikes for stateful services. In *First ACM Symposium on Cloud Computing (SOCC 2010)*, Indianapolis, IN, June 2010.
- [6] Peter Bodik, Moises Goldszmidt, and Armando Fox. Highlighter: Automatically building robust signatures of performance behavior for small- and large-scale systems. In *Third Workshop on Tackling Computer Systems Problems with Machine Learning (SysML 2008)*, San Diego, CA, Dec 2008.
- [7] Peter Bodik, Moises Goldszmidt, Dawn Woodard, Hans Andersen, and Armando Fox. Fingerprinting the datacenter: Automated classification of performance crises. In *SIGOPS European Conference on Computer Systems (EuroSys)*, Paris, France, Apr 2010.
- [8] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. Statistics-driven workload modeling for the cloud. In *Workshop on Self-Managing Database Systems (SMDB 2010)*, Long Beach, CA, March 2010.
- [9] Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson. Using machine learning to auto-tune a stencil code on a multicore architecture. In *1st*

Workshop on Hot Topics in Parallelism (HotPar 2009), Berkeley, CA, March 2009.

- [10] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting multiple performance metrics for queries: Better decisions enabled by machine learning. In *International Conference on Data Engineering (ICDE 2009)*, Shanghai, China, March 2009.
- [11] Beth Trushkowsky, Peter Bodik, Michael J. Franklin, and Armando Fox. Stateful scaling in the cloud. In preparation.
- [12] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Large-scale system problem detection by mining console logs. In *22nd ACM Symposium on Operating Systems Principles (SOSP 2009)*, Big Sky, Montana, October 2009.
- [13] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Online system problem detection by mining patterns of console logs. In *2009 International Conference on Data Mining (ICDM 09)*, Dec 2009.
- [14] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Experience mining google's production console logs. In *USENIX 2010 Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML '10)*, Vancouver, BC, Canada, October 2010. Submitted for publication.
- [15] Steve Zhang, Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. In *Proc. 20th ACM Symposium on Operating Systems Principles*, Cambridge, UK, 2005.