

# RADLab Technical Vision

Armando Fox\*, Michael Jordan, Randy Katz, David Patterson, Scott Shenker, Ion Stoica  
*fox@cs.stanford.edu, {jordan,randy,patterson,istoica,shenker}@cs.berkeley.edu*  
Updated: 12/6/2005 8:16:00 AM

## 1 The Fortune One Million

Over a long weekend in 1995, Pierre Omidyar created the initial version of what is now eBay. This is but one of the many large-scale Internet services—such as Amazon, Google, Mapquest and Craigslist—that have revolutionized what we can do on the Web.

However, the cost of that revolution is it takes an eBay- or Google-sized organization to turn a prototype or idea into a robust distributed service that can be relied on by millions. Innovation is fastest when it can leverage well-encapsulated prior building blocks as well as lessons, but today, the complexities of developing, assessing, deploying, and operating such services are *not* encapsulated, *not* systematized, and the design lessons exist largely as “folklore” taking different forms in different organizations. The next Eric Brewer, Bill Gates, Bill Joy, Pierre Omidyar, Sergey Brin, or Larry Page will have to build the corporate apparatus that can invest in vast repeated work and suffer repeated difficulties and failures in order to realize their dreams.

Our vision is to *enable one person to invent and run the next revolutionary IT service*, operationally expressing a new business idea as a multi-million-user service over the course of a long weekend. To do this, we will systematize what has become the *de facto* standard process for developing, assessing, deploying, and operating (DADO<sup>1</sup>) such services, by bringing to bear powerful techniques from statistical inference and machine learning as well as recent insights from networking and distributed systems. These new technologies and tools will enable an “open services ecology” where a talented individual can “instantly” design and deploy her own service.

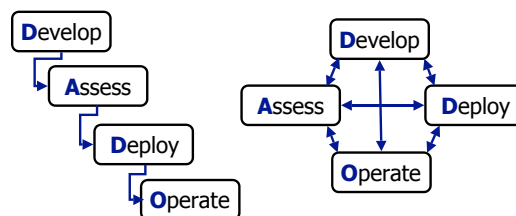
This vision is being realized in a new center in EECS at U.C. Berkeley called the RADLab, a laboratory to create core technology for Reliable, Adaptive, Distributed systems.

<sup>1</sup> Fittingly, in architecture a *dado* is the part of a pedestal that connects its cap to its base.

## 2 The Science: Systematizing DADO (Develop, Assess, Deploy, Operate)

The “waterfall” software development model is obsolete when applied to Internet services. Indeed, in most of today’s large services, the designers and developers share operational responsibility, and the operators must know something about the behaviors of the applications at large scale or under heavy workloads in order to do their jobs.

*What would it mean to create a “science” of developing, assessing, deploying and operating such services?* DADO, unlike the waterfall model, is an ongoing process by which the artifacts being created (applications and services) are measured and understood to improve their operational behavior, and this knowledge is being continuously fed back into the development process in the form of better tuning, new features, or bug/performance fixes.



This shortens the “distance” between a service’s users and its developers and allows for faster innovation and bug fixing. However, in today’s high-pressure development climate, the DADO steps and the connections between them are not always systematic. Our approach to “service science” will be to develop core technologies and competencies not only for each step but also for their interconnection. Our approach relies on two essential and complementary elements:

1. Applying statistical machine learning (SML) to close these loops, both by improving our understanding of developed artifacts and streamlining their deployment and operation.
2. Extending (and, in some cases, completely re-architecting) aspects of the underlying

Internet communication model. This will enable services to be built upon a base that is far more flexible and secure.

Both can be developed and applied in an incremental fashion to enable smooth evolution from today's deployed systems.

## 2.1 Develop

Service innovation is fastest when it can leverage highly sophisticated existing building blocks. Consider HousingMaps.com, a free Web service that shows rentable properties on an interactive map. HousingMaps consists of a modest amount of code that combines the highly-evolved Craigslist.org apartment listings and Google Maps API.

Alternatively, consider Google's internal MapReduce() primitive, which allows programmers to express implicit parallelism that not only exploits Google's scale and distributed resources, but also understands operational aspects of executing that implicit parallelism, such as proper load balancing for CPU scheduling and disk storage.

Both HousingMaps and MapReduce are components that enable an *ecology of services*, in which innovation can exploit existing building blocks that are operationally sophisticated and well developed. Instead of posting their code on Sourceforge, enterprising programmers should be able to deploy their *running service* instantly on a publicly available service platform. As their service's usage grows, its resource allocations would grow to match.

Mechanisms would be in place for understanding its pathologies under high load, its performance behaviors, and its resource needs, just as MapReduce's implementation knows about data locality, machine scheduling, and failure semantics. In a true service ecology, such facilities would allow other programmers to easily reuse or improve on a service and (re)deploy it.

Another example building block is a wide-area Distributed Hash Table (DHT), which is addressed using a *put/get* hashtable API but can be made resilient to denial-of-service and similar attacks and failures; operating DHTs as services would allow new services to "outsource" some of their storage in an attack-resilient way.

We believe middleware will also play a key role in enabling this ecology because it allows innovation beneath an important layer of

abstraction: commercial J2EE (Java 2 Enterprise Edition) application servers are differentiated by their diagnostic support, development environments, ease of integration with other Web standards, and so on. More importantly, componentized middleware imposes design constraints on application structure, such as separation of application logic from persistent and transient state in J2EE, that help us engineer faster recovery.

Our goal will be to create the middleware abstractions that expose API's appropriate for programming in an ecology-of-services model. One test for these abstractions is how much they simplify implementing a higher-level function like MapReduce or a higher-level service like HousingMaps. Although service-oriented middleware platforms such as NetBeans and .Net have taken important first steps in defining some of the relevant API's, creating a service ecology requires a vision that combines the development tools with improved assessment (finding bugs and performance problems), deployment (testing and simulation prior to release), and operations (keeping services running in a dynamic world).

## 2.2 Assess

In the areas of assessment and operations in particular, we see a new opportunity to apply machine learning and statistical learning theory (SML). The large scale and heavy usage of these systems means that not only do we have many data for SML algorithms, but since most systems work correctly most of the time, we have a basis for identifying and localizing anomalous behaviors relative to an observed baseline. Furthermore, advances in SML techniques, combined with Moore's Law, have resulted in algorithms fast enough to use in soft-real-time to monitor today's systems.

For example, Liblit *et al.* [4] showed how to do "statistical debugging" by instrumenting common programs with several "features" (for example, an assertion or comparison test in a particular line of code) and having each running copy report instrumentation for a random subset of features. Because the programs are so widely used, they were able to gather enough reports to correlate specific features with failed runs of the program (crashes), leading to the discovery of new bugs in such. Kiciman *et al.*[5] showed that probabilistic context-free grammars could be used to detect and localize "brownout" failures

in Internet services (errors that affect only a subset of users or requests and cannot usually be detected without special-case checks). His tool, Pinpoint, detected 20-30% more errors than existing techniques, without requiring any special-case checks or application-specific knowledge.

Middleware provides an excellent facility for separating instrumentation from application logic. We can change the instrumentation collected on a running system simply by changing the middleware, leaving applications unmodified. SML then helps us find the “needle in the haystack”—interesting patterns, buried in the instrumentation, that tell us something new about the behavior of the application.

In contrast, a main challenge of assessing the behavior of the wide-area network is a relative *lack* of observation and control points as compared to applications. We propose to develop and deploy a new type of network element, *inspection-and-action boxes (iBoxes)*, as the foundation of a comprehensive network infrastructure to provide such inspection and action points. Deployed at critical points such as network edges and Autonomous System boundaries, iBoxes collect data about packets and flows, compute statistics, and contribute observations to a network-wide formulation of models that discriminate between normal and abnormal network conditions [12][13].

Suppose that the network observations suggest a correlation between heavy link utilization at the server edge and high numbers of packets originating from outside the network. To determine causality, the network experiments by acting to delay external packets arriving from the Internet edge. The network then observes a reduction in server-edge link utilization. Conversely, slowing packets traversing to the servers has essentially no effect on the arrival rate of external packets. The network can then infer that large numbers of externally generated packets cause server link utilization to increase. In the future, whenever high server link utilization is detected, the network management algorithms can act to restrict the arrival of external packets in order to return server link utilization to desired levels.

Using techniques like this, we have been able to demonstrate “smart” load balancing of server requests by being analyzing the relationship between request type and impact on server load [11].

Whereas existing network data-collection tools such as Netflow focus on observation, iBoxes combine observation with analysis to enable management actions. For example, iBoxes can be instructed to drop selected packets, perform filtering, or take other actions to mitigate or prevent abuse of the network (“bad” traffic) while preferentially protecting “good” traffic. iBox analyses can even be forwarded to higher middleware layers in RADS application servers to allow coordinated application/network response to adverse network conditions.

The programmer-visible API to iBox functionality is a new *annotation layer (AL)* in the protocol stack that allows end-hosts, routers, and middleboxes to embed arbitrary information into data and control packets. annotations may be simple observations, such as a tag added by edge routers to distinguish internal from external packets, or they may be a label computed as the result of distributed packet analysis. Unlike proposals relying on extensive modification of existing IP header option bits (which often cause today’s routers to simply drop the packets), AL-encoded information is transparent to non-AL-aware entities.

For example, RADS applications will be able to tag their network traffic between local servers and the datacenter edges with special annotations that assist the network to distinguish their packets as “good.” These application packets receive protection in the form of preferential bandwidth and priority even when the network is under very heavy loading conditions.

iBoxes can be prototyped initially on software-based routers. They will ultimately be implemented on *programmable network elements (PNEs)*, an emerging class of extensible routers such as Layer 7 switches with hardware support for deep packet inspection and processing, able to operate at higher line speeds as local area networks evolve.

### 2.3 Deploy

Since deploying iBoxes at strategic points such as access links can be expensive, we rely on *delegation and indirection* mechanisms that allow both the sender and the receiver to specify explicitly iBoxes along the data path. For example, employees could specify that all traffic in and out of their machines be routed through their company’s virus filter, irrespective of her

location, giving them the same level of security at a public access point that she enjoys at work. In the context of RADS, this technique can be used to deploy new services without costly network reconfigurations, and assess the system behavior by redirecting the traffic to observation and analysis points.

Consider the latest generation of application accelerators like Riverbed's Steelhead, which use a combination of caching, remote-reference, and TCP optimization techniques to improve the application performance in wide area networks (WANs). Today, deploying such solutions require one to place the application accelerator boxes at the access links, a non-trivial and time-consuming process. In contrast, using indirection, one can place these boxes in the server room, and then redirect the applications' traffic through these boxes. Furthermore, this approach enables one to redirect the traffic on a per-application basis. This flexibility allows for incremental deployment, simplifies testing of new functionalities, and avoids overloading the boxes with traffic that doesn't need to be accelerated, such as web traffic. The delegation and indirection primitives can be supported through easy-to-deploy overlay networks such as i3 [14], which have been running on the Berkeley campus for the past 18 months.

How will distributed applications behave when deployed on hundreds or thousands of processors? How can we test our techniques for improving observability in the network, such as inspection-and-action boxes, without actually modifying routers *in situ*? A radical research vehicle, RAMP (Research Accelerator for Multiple Processors), may help us perform this parallel hardware and software research.

RAMP is a research-oriented, low-cost, highly-scalable system emulator supported by a multiuniversity cooperative and constructed at the Berkeley Wireless Research Center [Arvind et al]. The first generation boards contain enough FPGA's to emulate about 100 CPU's, each with 256MB DRAM and 20GB of disk. The second generation will double those parameters. RAMP can be built and operated cheaply (follows Moore's Law), the FPGAs make it easy to modify or trace any aspect of the simulated system, and it can host real instruction sets and OS's such as PowerPC, SPARC, Solaris, and so on. Although its slow clock rate (100-200Mhz) is no match for today's hottest CPUs, RAMP accurately tracks emulated time

to allow researchers to explore many parameters, including clock rate, memory latency, network bandwidth, and so forth.

Although RAMP is intended to spur architecture/software research in conventional parallel processing, we plan to use several RAMP boards to emulate a datacenter with hundreds of CPU's, or several datacenters connected by a network that includes inspection-and-action boxes. We believe that today's datacenters are near their apotheosis in terms of size and scale, and that it is time to consider alternative datacenter architectures.

For example, Akamai distributes their service over thousands of racks in hundreds of datacenters connected by multiple long-haul networks; in contrast, most of today's large-scale services operate a few massive datacenters containing thousands of machines and connected to one or two long-haul networks. Although Akamai's development processes, tools, and techniques do not necessarily generalize to building the "service ecology," we envision for rapid innovation, RAMP can serve as a "datacenter-in-a-box" research vehicle for exploring the performance and dependability ramifications of such alternatives and the new techniques and tools they would require.

We anticipate separate RAMPs for development and "production" work (i.e. leaving services deployed for public use over long timescales).

## 2.4 Operate

Middleware is also an important ingredient for our Operate strategy. We have successfully incorporated support for instrumentation and recovery based on statistical monitoring into industry-standard J2EE (Java 2 Enterprise Edition) middleware; existing applications enjoy decreases in recovery time by up to an order of magnitude and increases in availability of 20-30% when run on our improved middleware. Besides easing development by allowing the creation of reusable components, middleware serves to separate the operational concerns of monitoring and recovery from the mainline application logic.

We also see an opportunity for combining SML techniques with "operator-friendly" visualizations. The visualizations allow operators to crosscheck the results reported by SML algorithms and allow them to build trust in these techniques over time. We applied this

technique to real outage data at Ebates.com [1] and found that many of their actual failures could have been predicted up to hours in advance, and events that were misclassified as possible failures could have been rapidly identified as the result of a known system change.

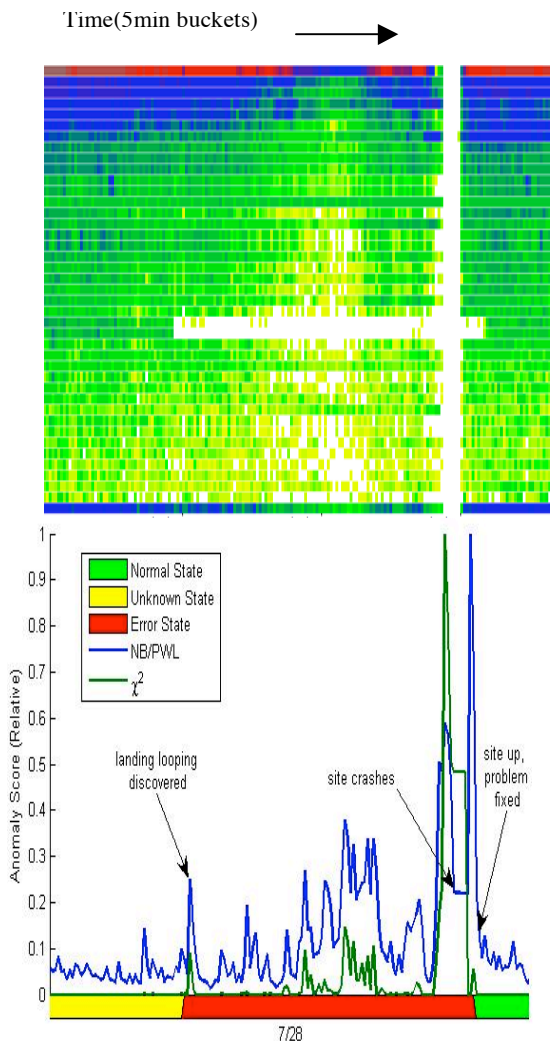
In the accompanying figure, visualizing the hit counts to popular pages (top) allows anomalies in the access frequencies to be easily spotted visually, and the corresponding anomaly scores from two different algorithms (bottom) can be used to crosscheck or drill down on a specific problem that was observed visually. By letting the computers crunch the data and the humans use their innate visual pattern recognition capabilities, we can provide *actionable* information that lets operators apply their extensive experience with the system in deciding how and whether to react.

Cohen, Zhang *et al.* have shown [6] that statistical inference can be used to correlate low-level system properties to high-level performance pathologies, and that a “fingerprint” of a system’s essential state can be meaningfully captured and used to identify

recurrent problems in the future [10]. Their method is especially useful when the data being used for modeling are incomplete or noisy, as has been the case in all production systems we have worked with. Therefore, another key contribution will be the creation of a publicly available repository of system traces and outage data, containing sanitized information that can be used to compare meaningfully different approaches and to sanity-check the realism of synthetic workloads and faultloads. We will provide tools to help companies sanitize their data automatically for inclusion in the database, such as filters to and from common formats.

As we have done in previous projects, courses will be used as “alpha testing” for our ideas. We envision offering a course each year to students across the Berkeley campus. Classes will test whether we have indeed “lowered the barrier to entry” for programmers without full CS backgrounds to realistically deploy large-scale applications.

We intend to make available and operate the most successful services developed in these courses. The courses and live services allow us to improve our system based on feedback



### 3 The RADLab

To pursue this research agenda, we are forming RADLab, an interdisciplinary center to create core technologies and develop core competencies to systematize the DADO process. The goal is to provide an atmosphere in which academic and industry researchers and engineers can collaborate on long-range pre-competitive technologies, and to train future IT leaders in multiple disciplines.

The Principal Investigators (PI’s) are leaders in systems, architecture, networking, and machine learning. Berkeley has a long record of accomplishment of successful technology transfer through close partnership with industry and liberal IP policy. Of the “Big 4” computer science research programs, the National Academy of Engineering identifies Berkeley research that led to 7 out of 19 \$1B+ industries, compared to 5 mentions for Stanford and MIT and 3 for CMU [3]. *US News & World Report* ranked Berkeley #1 in computer systems, above CMU, MIT, and Stanford [6]. In addition to technology innovation, Berkeley has long been one of the top suppliers of systems students to industry and academia.

The RADLab is exploring a new model of funding in which 80% comes from industry and 20% from government sources, rather than vice versa as in the past, and to partially match these donations via UC programs such as MICRO and Discovery. This strategy reflects the recent serious challenges in federal funding for IT [7][8].

Our plan is to find 3 or 4 foundation companies and 3 to 6 affiliate companies. In return for a donation of at least \$500,000 per year from each foundation partner, we will give preference to using foundation partners' technologies in the RADLab and involve company representatives in our design teams. (If foundation partners have competing technologies, we will make a technical decision as to which is the best match to the project, assuming that we cannot use all technologies.) We will invite members of both the foundation partners and affiliate companies (a donation of at least \$50,000 per year) to three-day offsite retreats twice per year. (We may also offer a developing company category at a much lower annual rate for startup companies.) The presentations made at those retreats will be available only to our partner companies for the first 6 months. The RADLab will make the software we develop openly available to the public. None of the Berkeley participants has ever filed a patent through the University of California, and we have no plans to start doing so now with the RADLab.

## 4 References

[1] Arvind, K. Asanovic, D. Chiou, J. Hoe, C. Kozyrakis, S. Lu, M. Oskin, D. Patterson, J. Rabaey, J. Wawrzynek. *RAMP: Research Accelerator for Multiple Processors - A Community Vision for a Shared Experimental Parallel*

*HW/SW Platform*, UC Berkeley CS Technical Report, UCB//CSD-05-1412, Sept. 2005.

[2] P. Bodik, G. Friedman, L. Biewald, H. Leving, G. Candea, A. Fox, D. Patterson, M. Jordan. *Combining visualization and statistical analysis to improve operator confidence*. Proc. 2005 Intl. Conf. on Autonomic Comp., Seattle, June 2005.

[3] *Innovation in Information Technology*, Computer Science Technologies Board, September 2003.

[4] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. *Scalable statistical bug isolation*. Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2005.

[5] E. Kiciman and A. Fox. *Detecting Application-Level Failures in Component-Based Internet Systems*. IEEE Trans. Neural Networks, invited article, to appear Spring 2005.

[6] *America's Best Graduate Schools*, US News & World Report, 2002.

[7] J. Markoff, *Pentagon Redirects Its Research Dollars*, April 2, 2005: New York Times.

[8] D. Patterson, *President's Letter: The State of Funding for New Initiatives in Computer Science and Engineering*, Communications of the ACM, April 2005.

[9] S. Zhang, I. Cohen, et al. *Ensembles of models for automated diagnosis of system performance problems*. Proc. DSN-2005, Yokohama, June 2005.

[10] S. Zhang, I. Cohen, et al. *Capturing, indexing, clustering and retrieving system history*. Proc. SOSP 2005, Brighton, UK, Oct. 2005.

[11] G. Porter, R. H. Katz, *Effective Web Service Load balancing through Statistical Monitoring*, IFIP/IEEE International Workshop on Self-Managed Systems and Services (SelfMan 2005), Nice, France, (May 2005).

[12] R. H. Katz, G. Porter, S. Shenker, I. Stoica, M. Tsai, *COPS: Quality of Service vs. Any Service at All*, Springer Verlag Lecture Notes in Computer Science, Proceedings International Workshop on Quality of Service (IWQOS 05), Passau, Germany, (June 2005). Invited Paper.

[13] M. Caesar, L. Subramanian, R. H. Katz, *The Case for an Internet Health Monitoring System*, First Workshop on Hot Topics in System Dependability (HotDep), Yokohama, Japan, (June 2005).

[14] Ion Stoica, Daniel Adkins, Shelley Zhaung, Scott Shenker, and Sonesh Surana, *Internet Indirection Infrastructure*, IEEE/ACM Transactions on Networking, Vol. 12, No. 2, pp. 205-218, April 2004.