The Perfect Storm: Reliability Benchmarking for Global-Scale Services

Amin Vahdat and Jeff Chase
Department of Computer Science
Duke University

Mike Dahlin
Department of Computer Sciences
The University of Texas at Austin

Abstract

Technology trends are leading many to consider pushing applications and storage systems into the network infrastructure for transparent anywhere/anytime access to programs and data. These trends apply not only to low value services, but also business critical and mission critical services. Thus, a key research challenge is developing ways to deliver high-assurance network services across the commodity Internet. Meeting this challenge will require a large-scale coordinated effort spanning operating systems, distributed systems, and networking. This paper argues that a necessary early step in this effort is designing an experimental methodology to characterize the extent to which systems succeed in meeting this challenge. We outline one such methodology that envisions several sets of scenarios that provide progressively deeper understanding of network service behavior: Calm Days scenarios for understanding normal behavior; Red Skies scenarios for understanding behavior under unusual loads, failure patterns, or attacks; and Perfect Storm scenarios for understanding behavior under combinations of problems.

1 Introduction

Today, users are increasingly accessing services and storage from a variety of devices, for instance, a laptop, a home computer, a work computer, a PDA, and a cell phone. At the same time, maintaining these machines and storage is becoming increasingly burdensome and expensive. These trends are leading many to consider pushing traditional applications and storage systems into the network infrastructure for transparent anywhere/anytime access to programs and data. This infrastructure model can provide simplified administration, (potentially) higher reliability and security, and economic benefits resulting from scale.

Many of these same trends imply that, increasingly, mission-critical services, such as air-traffic control, medical, and military applications, will run over the public Internet. Just as Commercial-Off-The-Shelf (COTS) hardware and software enjoy such large cost and performance advantages over custom systems that they are increasingly used as a basis for mission-critical systems, the huge existing infrastructure and the rapid pace of technological innovation in networking and distributed systems means that "one-of" systems will face significant difficulties in keeping pace with the latest innovations in commodity systems.

In such an environment, key research questions for operating systems, distributed systems, and networking are: How do you build network services out of commodity pieces that deliver desired levels of performance and availability? How do we augment the current Internet infrastructure, at all layers, to support mission-critical systems? How does one compare the performance and availability of one system to another?

Clearly, an exhaustive discussion of how to answer these questions is beyond the scope of this paper. It is likely that solving this problem will require continued advances in emerging techniques such as network overlays [1, 21], adaptive service and data replication [26, 2, 23, 3, 28], continued operation across failures [5, 12], to name just a few. The hypothesis this paper is that a key missing piece is an experimental methodology and infrastructure to evaluate the reliability of a given service architecture and its realization. In particular, a goal of our work is to enable designers and vendors of network services to compete on availability, robustness, and guaranteed worst-case performance under a wide range of operating conditions, not just average performance under idealized conditions.

While the systems research community has decades of experience evaluating raw system performance, there is comparatively little experience with measuring system reliability. One difficulty in understanding reliability is that, by definition, the system must be evaluated under extraordinary circumstances. Typical trace-based evaluations may not be sufficient to understand the behavior of the system, for instance, when the goal is 99.999% availability. A second difficulty arises from the fact that network service reliability must be measured on an end-to-end per-client basis and must account for service-specific Quality of Service (QoS) metrics or Service Level Agreements (SLAs).

An infrastructure for evaluating mission-critical services must address at least three issues:

1. **Workload and models**. We must develop ways to translate measured common-case workloads and fault-loads to new workloads that allow us to evaluate systems under more challenging but foreseeable circumstances. In this paper, we argue for developing a controlled infrastructure for measuring the behavior of network services under Calm Day scenarios (e.g., normal access patterns, expected failure patterns), Red Skies scenarios (e.g., flash crowds, projected worst-case failure patterns, or deliberate attacks on the system), and Perfect Storm scenarios (e.g., simultaneous or correlated heavy load, heavy failures, and/or deliberate attack.)

2. **Metrics**. We must develop metrics that succinctly and accurately characterize the properties of the system. We argue that these metrics must appropriately characterize the distribution of performance and availability and data quality seen by different sets of clients under different network conditions.

3. **Experimental infrastructure**. It must be possible to experimentally measure a system and compare one system against another. We consider the systems questions associated with building a flexible test harness for subjecting COTS network services running on COTS operating systems to a variety of client and network characteristics.

In the rest of the paper, we first discuss the interplay of system workloads and faultloads. Then, we discuss metrics for characterizing systems. In Section 4 we outline how to integrate these issues into an experimental framework. Finally, Section 5 summarizes our conclusions and future directions.

2 Workload and environment

A service's behavior is affected both by its workload—the set of requests sent to it—and also by by the larger network environment where it operates.

- Common case workloads can be characterized by traces. But for highly-available services, other scenarios must be considered. For example, systems must be able to handle unusual load situations such as flash crowds. Given that systems are deployed on the commodity Internet, it may also be important that they resist deliberate attacks such as distributed denial of service attacks (DDoS) or targeted attacks to exploit bugs in the system.

- Environmental factors that affect services include hardware, software, and maintenance failures within the system as well as external network variability or failures.

Different systems will address these problems in different ways and to different degrees. To simplify understanding of this broad range of trade-offs, we organize scenarios into three basic groups that should be considered by systems seeking to provide progressively stronger service guarantees.

Calm day. Calm day scenarios represent workloads with typical access patterns and environments with typical failure patterns and no deliberate attacks.

Some techniques for evaluating calm day workloads are relatively well understood. Request traces have long been used to benchmark systems. And a number of studies have quantified environmental factors such as hardware, maintenance, and environmental failures [13], Internet failures [18, 15, 9, 1], and Internet performance variability [29]. Several recent studies have used fault-loads derived from such studies to examine end-to-end service availability [9, 28].

To deepen system understanding under calm day scenarios, additional research is needed. On the request-load side, for example, it may be important to consider the long-term evolution of the service. For instance, it would be important to consider the rate at which new content is introduced and how clients access different portions of the service as a function of time. On the environment side, although there is a growing body of publicly available data on network availability [18, 15, 9, 16], additional studies are needed to help Internet service designers understand the range of Internet behavior. In particular, we need traces spanning longer periods of time (e.g., months or years v. days or weeks), resolving finer time granularities (e.g., failure-duration resolutions of seconds rather than tens of minutes), including performance information (e.g., available bandwidth and latency data rather than just connectivity information), and correlating measured properties with network topology features (e.g., distinguish the difference, if any, in performance variability between nodes on the same ISP versus nodes on different ISPs). Emerging technologies such as PlanetLab [19] and ScriptRoute [22] may help enable these more detailed measurements.

Red skies. Red skies scenarios reflect stressful scenarios that are unusual but nonetheless common enough that they must be considered in assessing business-critical or mission-critical systems. These scenarios can be organized into three groups.

Natural disasters (RS1): These are scenarios that stress Internet services but that are not deliberately initiated. For example, flash crowds can subject a service to request loads many times those normally seen. Similarly, studies suggest that Internet routing interruption durations are heavy-tailed, meaning that long interruptions are rare but account for a significant fraction of overall interruption time [15, 9]. Other examples of rare but stressful events that should be considered include power outages [13], system upgrades and maintenance [13, 25, 4], and internal hardware and software failures.

Deliberate "black box" attacks (RS2): These are scenarios where an external adversary deliberately attempts to impede service delivery by accessing the service by sending either a large number of "normal" requests to the system or by sending "generic abnormal" requests. One example of an attack based on "normal" requests is a distributed denial of service (DDoS) attack where many nodes simultaneously and repeatedly attempt to download a large file from a site. "Generic abnormal" requests are requests designed to challenge system implementations but that do not target service-specific known bugs. For example, an attacker might initiate requests to a cgi program with randomly generated arguments with the hope of triggering a bug.

Deliberate "vulnerability-targeted" attacks (RS3): These are scenarios where an external adversary deliberately attempts to impede service delivery by exploiting service-specific vulnerabilities. A notorious example of this sort of attack are the various sendmail bugs used to compromise many Internet systems.

The above taxonomy generally orders the scenarios by increasing complexity of execution and by increasing depth of coverage. Note that the line between "black box" and "vulnerability-targeted" attacks may not be sharp. Qualitatively, we intend "black box" attacks to be attacks that can be included in a toolkit of "generic" attacks that are specific to a wide range of services while "vulnerability-targeted" attacks may have to be specifically crafted for each service. A key research challenge is developing workloads for systematic Red Skies testing.

RS 1 scenarios could be modeled by observing existing systems to characterize the frequency, severity, duration, and other key parameters to develop synthetic models of events of interest such as flash crowds, network failures, hardware failures, or maintenance failures. Given the heavy-tailed distribution of some of these properties, it may be necessary to extrapolate beyond the events observed in finite-length traces to project the severity of the worst-case event likely to be seen. For example, we should be able to estimate the severity of the "100-year Internet failure event" – the most severe failure that has at least a 1% chance of occurring per year – so that just as a civil engineer can design buildings to tolerate a 100-year storm event, service designers could construct services to tolerate 100-year failure events.

Some RS 2 workloads could be modeled by tracing and extrapolating from observed events. For example, Moore et. al measured several important aspects of a wide collection of Internet denial of service attacks [17]. Other RS 2 workloads could be modeled on existing tools such as Ballista [11] (which systematically varies parameters to Unix system calls in ways designed to trigger unknown implementation bugs).

Developing RS 3 workloads is challenging. Short of formally proving the correctness of a service, it seems difficult to know all of the bugs that could be lurking in it. We believe that it is still possible to generate some broadly-useful RS 3 scenarios. For example, for a given package one could track the rate of new bug discovery and project the number of remaining latent bugs. Another approach may be to extend fault injection techniques designed to model common programmer errors [14, 6] to inject faults designed to model common exploitable network service bugs.

Perfect storm. As made famous by the book and movie of the same name, a 1991 storm in the Atlantic Ocean off the east coast of the United States has become known as a "Perfect Storm" because a combination of factors combined to create unusually vicious conditions. System engineers designing business critical and mission critical services must similarly consider how different factors listed above may interact, and workloads should be developed that capture important aspects of these interactions.

One source of interaction is chance. Given accurate Red Skies workloads, it should be possible to estimate the probability of multiple factors occurring simultaneously and to identify and model significantly likely interactions. In addition to random interactions, workloads must consider correlation among interactions. For example, an attacker might deliberately launch an attack during a flash crowd event or a widespread network outage.

3 Metrics

3.1 Performability

The overall goal of our work is to enable meaningful comparison of competing service architectures under a variety of network conditions. Thus, we must distill service behavior to a meaningful yet minimal set of metrics. There are three inter-related axes along which a service's behavior must be evaluated: performance, availability, and data quality. Considering the first two, we adapt earlier work in the performability community. We argue that service availability should be measured as a distribution of response times for individual client requests. This response time must account for end-to-end client-perceived performance, including WAN propagation time and congestion, DNS resolution, server processing time, etc. Certain requests may have an infinite response time, corresponding to the case where the service has suffered a true failure—the client cannot access the service at all. In other cases, the service may return a response, but so slowly that the service must effectively be considered unavailable.

Such a distribution of response times must be considered in light of individual service characteristics and client expectations. For instance, clients accessing a Web news service may have different expectations than those downloading a large software file or performing a complex query against a genome database. Further, this distribution of response time might be evaluated in the context of an SLA.

3.2 Data Quality

The third axes along which service behavior must be evaluated is the quality of the data returned to individual client requests. In general, quantifying data quality must be done in an application-specific manner. However, some potential, application-independent measures of data quality include:

- *Consistency:* For many services, small and bounded reductions in data consistency is tolerable to end users, especially in exchange for improved performability. Earlier work [28] shows that service consistency can be numerically quantified in a general, application-independent manner and that reducing consistency results in commensurate improvements in overall service performance and availability.

- *Transcoding:* Many Web services present rich multimedia content to end users. However, the content can often be effectively presented to end users with reduced multimedia fidelity.

- *Online Aggregation:* Database queries make up a significant portion of network service access. Recent work [20] shows that many aggregation queries (e.g., average employee salary or temperature) can be answered with high accuracy by sampling the underlying data.

- *Harvest/Yield:* Related to both consistency and online aggregation, many web search services may not access the entire inverted index in returning client results. This reduction in the quality of returned search results is typically imperceptible to end users, but results in significantly improved performability relative to waiting for all responses in a cluster environment.

These measures of data quality are interesting because services can often trade decreased data quality for improved performability. Earlier work [28, 20, 7] shows that moderate decreases in data quality can result in disproportionately large improvements in overall service throughput.

3.3 Discussion

Given the above definitions of performance, availability, and data quality, the principal challenge becomes blending these metrics into simple measures of service behavior as a function of global network characteristics (client access patterns, network conditions, attacks, failures, etc.).

We will have to develop metrics that trade the completeness of benchmark results against their understandability. For instance, the system could return the performance and data quality for each individual request. While such results are complete, it is difficult to compare two services based on this raw data. Distilling these results to cumulative distribution functions of performance and data quality is more understandable, though potentially useful information is lost. From the perspective of simplicity, we envision a numeric continuum of failure characteristics, ranging from Calm Day to the Perfect Storm. The metric for evaluating overall service behavior then becomes the maximum level of failure for which a given architecture can still satisfy a target SLA, where the SLA specifies performability and data quality requirements. Thus, a service architecture can be judged to be superior to another if it can satisfy a given SLA further along the failure spectrum for the same cost.

4 Putting it All Together

Given workloads and metrics, we want to make it relatively easy to experimentally evaluate systems under Calm Day, Red Skies, and Perfect Storm conditions.

A testbed for Internet services should allow users to test unmodified software prototypes – including user-specified operating systems and application software – in a configurable Internet-like environment including realistic topologies and switch behavior. Existing systems such as Emulab/Netbed [27], and ModelNet [24] provide a basic framework of this sort. The ORCHESTRA system [10] provides an environment for evaluating distributed systems that is specifically designed to simplify fault injection, but it focuses on cluster network models and restricts operating system choice.

We envision a two-pronged research effort to develop an experimental service benchmarking system. First, we must develop "packages" of Calm Day, Red Skies, and Perfect Storm network conditions under which systems can be tested. Some of these packages may specify generic, service-independent conditions such as a package that specifies network topology, performance, and availability on a "normal network day," "1-year bad network day," and "100-year bad network day." Other packages may be service dependent, but be generated automatically in a service-independent way. For example, given the interface to a service, one could automatically generate a “black box” attack on the system that probes for vulnerabilities using techniques similar to Ballista [11]; or, one could develop a self-scaling benchmark [8] that automatically characterizes a range of Calm Day performance characteristics based on an input set of traces or that automatically generates a Red Skies denial of service attack workload by identifying expensive queries for the system to process. Beyond these standard packages, of course, the system must still support evaluation under service-dependent workloads.

Second, we plan to extend ModelNet by adding interfaces to accept these packages. The idea is to run target services within ModelNet, subject to specified access patterns and failure conditions. Failures may consist of WAN failures—preventing a subset of clients from accessing the service or a subset of its replicas—or internal service failures—reducing available service throughput at a given site, reducing data quality, or making a portion of the service unavailable. To capture such complex interactions, we will extend ModelNet to not only inject faults, but to capture the impact of various failures on end-to-end behavior. For instance, we must capture the behavior of BGP to determine the impact of WAN router or wide-area replica failure and of OSPF to capture the effects of internal service failures. ModelNet can already capture the complex interactions of multi-tiered services, for instance, accounting for databases, web servers, application servers, storage, switches, and routers. Because ModelNet runs unmodified application code and operating systems, the effect of individual failures on overall service availability and data quality can be measured directly from individual client perspectives.

5 Conclusions and future directions

Given technology trends, a key research challenge is developing ways to deliver high-assurance network services across the commodity Internet. Meeting this challenge will require a large-scale coordinated effort spanning operating systems, distributed systems, and networking. This paper outlines a necessary early step in this effort: designing an experimental methodology to characterize the extent to which systems succeed in meeting this challenge.

References

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *SOSP*, 2001.

[2] A. Awadallah and M. Rosenblum. The vMatrix: A network of virtual machine monitors for dynamic content distribution. In ”*Web Caching and Content Distribution Workshop*, August 2002.

[3] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and Ad. Vahdat. Opus: an overlay peer utility service. In *OPENARCH*, June 2002.

[4] E. Brewer. Lessons from giant-scale services.

[5] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*, February 1999.

[6] S. Chandra and P. Chen. How fail-stop are faulty programs? In *FTCS*, June 1998.

[7] Surendar Chandra, Ashish Gehani, Carla Schlatter Ellis, and Amin Vahdat. Transcoding characteristics of web images. In Martin Kienzle and Wu chi Feng, editors, *Multimedia Computing and Networking (MMCN’01)*, volume 4312, pages 135–149, San Jose, CA, January 2001.

[8] P. Chen and D. Patterson. A New Approach to I/O Performance Evaluation–Self-Scaling I/O Benchmarks, Predicted I/O Performance. In *SIGMETRICS*, pages 1–12, May 1993.

[9] M. Dahlin, B. Chandra, L. Gao, and A. Nayate. End-to-end wan service availability. *IEEE/ACM Transactions on Networking*, 2003. To appear.

[10] Scott Dawson, Farnam Jahanian, and Todd Mitton. ORCHESTRA: A fault injection environment for distributed systems. Technical Report CSE-TR-318-96, University of Michigan, 26 1996.

[11] J. DeVale, P. Koopman, and D. Guttendorf. The ballista software robustness testing service. In *Proceedings of TCS99*, 1999.

[12] A. Fox. Towards recovery-oriented computing. In *VLDB*, August 2002.

[13] J. Gray. A Census of Tandem System Availability Between 1985 and 1990. *IEEE Transactions on Reliability*, 39(4):409–418, October 1990.

[14] W. Kao, R. Iyer, and D. Tang. FINE: A fault injection and monitoring environment for tracing the unix system behavior under faults. *IEEE Transactions on Software Engineering*, 19(11):1105–1118, November 1993.

[15] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Backbone Failures. In *FTCS99*, June 1999.

[16] Isp ratings. http://special.matrixnetsystems.com/ratings/goratings.asp, January 2003.

[17] D. Moore, G. Voelker, and S. Savage. Inferring internet denial of service activity. In *USENIX Security Symposium*, August 2001.

[18] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, April 1997.

[19] L. Peterson, D. Culler, and T. Anderson. Planetlab: A testbed for developing and deploying network services. http://www.planet-lab.org/pubs/vision.pdf, June 2002.

[20] V. Raman and J. Hellerstein. Partial results for online query processing. In *SIGMOD*, 2002.

[21] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-end Effects of Internet Path Selection. In *SIGCOMM*, pages 289–299, September 1999.

[22] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility. In *USITS*, 2003.

[23] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active Naming: Flexible Location and Transport of Wide-Area Resources. In *USITS*, October 1999.

[24] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI*, pages 270–284, December 2002.

[25] M. Wagner. Microsoft IM outage hits millions. *InternetWeek*, January 2003.

[26] A. Whitaker, M. Shaw, and S. Gribble. Scale and performance in the denali isolation kernel. In *OSDI*, December 2002.

[27] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, pages 255–270, December 2002.

[28] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems*, 20(3), August 2002.

[29] Y. Zhang, V. Paxson, and S. Shenkar. The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. Technical report, AT&T Center for Internet Research at ICSI, http://www.aciri.org/, May 2000.