Towards a framework for automated robustness evaluation of distributed services

David Oppenheimer, Vitaliy Vatkovskiy, and David A. Patterson
University of California, Berkeley
{davidopp, vatkov, pattrsn}@cs.berkeley.edu

1. Introduction

Large-scale distributed services such as content distribution networks, overlay networks, peer-to-peer storage, distributed hash tables, and scientific and business Grid applications, have received substantial interest from researchers and industry over the past few years. But little attention has been paid to developing standard tools, techniques, and workloads for evaluating key properties of these systems, particularly robustness. As a result, prototype implementations are often subjected to the evaluations that are easiest to conduct, such as measuring system performance under benign operating conditions, while leaving robustness evaluation to limited scenarios or to simulations that may abstract away important details. Moreover, the lack of a standard robustness evaluation framework requires each team of developers to create their own infrastructure and makes it difficult to evaluate design tradeoffs across multiple implementations of the same service. We believe that distributed service robustness can be enhanced significantly if prototype implementations can be easily and carefully evaluated in a standard way.

Network emulation environments such as Emulab [22] and Modelnet [20], and real-world testbeds such as PlanetLab [3], have enabled application evaluation on realistic network topologies. Our vision is to extend the scope of such platforms into the application software stack, by adding a framework to automatically apply workloads, to perturb, and to measure complete distributed services, based on a user's specification. Such a framework allows services to be easily and comprehensively evaluated both under normal conditions and in the face of a flexible set of "stressful" events, thereby enabling a high degree of rigor and consistency in evaluating system robustness. Along with this framework we envision domain-specific standardized *robustness benchmarks.*

In this paper we discuss the research issues we have uncovered in building a prototype of a such a framework (ACME) [11] that runs on Emulab, Modelnet, and PlanetLab; some details of the framework itself; and some discoveries we have made in using it. We define *robustness* as the Quality of Service that a system provides during and after disruptive events that we call *perturbations* (a generalization of the traditional concept of a *fault*). Such measures are sometimes also called *performability.* We do not consider security issues, although some of the evaluation techniques that we describe apply to that aspect of system dependability as well.

2. Research issues

Building a reusable infrastructure for evaluating large-scale distributed services raises a number of challenging research questions.

2.1. Defining robustness benchmarks

The systems community has developed benchmarks for a number of domains, including CPUs [17], web servers [17], and databases [19]. But no widely accepted benchmarks exist for emerging distributed services such as those mentioned earlier. And although performance evaluations of distributed service prototypes are common, thorough robustness evaluations of such implementations are relatively rare in the literature and are not standardized. This is surprising because robustness is one of the principal features of many distributed services. Moreover, while the absence of robustness benchmarks may be acceptable for applications that usually run in a managed single-site datacenter, such an omission is less acceptable for distributed applications that run across failure-prone wide-area networks and that may use end-user machines, where much can go wrong. To remedy this situation, we advocate distributed service robustness benchmarks.

Robustness benchmarks have four components: the workload, perturbation load, metrics, and test environment. The goal is to measure the Quality of Service a system provides during and after perturbations. Benchmark *workloads* must be defined on a per-application-class basis. For example, we have used ACME to build a prototype benchmark for structured peer-to-peer overlay networks, in which the workload is requests to discover which node owns randomly-generated distributed hash table (DHT) keys. The *test environment* for a distributed service includes the number and type of nodes used, and the network topology and link characteristics interconnecting those nodes. In our benchmark, we used 150 nodes running on an emulated transit-stub network. Some *metrics* are generic across most services--for example, in our benchmark we measured end-to-end request latency, percentage of requests that completed, and amount of network bandwidth used. Other metrics are service-specific, such as consistency of the responses returned when multiple nodes make the same request at the same time (a metric we used in our benchmark) and correctness of the results returned. Finally, some *perturbations* are generic to any networked system, while others are service-specific. Examples of the former that we included in our benchmark include nodes joining or leaving the network temporarily (as in a node failure, network partition, or an end-user turning off her computer for the night) or permanently (as in membership "churn"), possibly at a rapid pace and/or many at a time. Other examples of generic perturbations include network congestion and correlated node failures. Examples of service-specific perturbations include sudden load spikes, configuration errors, and bogus network messages due to programmer error. Creating robustness benchmarks for the wide variety of emerging "planetary

scale" services is very much an open research question.

Defining robustness benchmarks raises not just a standardization issue, but also an acceptance one. A question that we have faced in our work has been: What degree of robustness should prototypes of robust systems be required to attain, in order to "prove" that they are robust? On the one hand, prototypes by their very nature will have bugs and implementation short-cuts, and some of these may hurt the prototype's robustness. On the other hand, there is a danger that researchers will overlook genuinely important aspects of robustness when designing their systems if they do not strive for a high degree of robustness even in their prototypes. Standard benchmarks help to ensure that researchers focus on all aspects of robustness that the community deems important, not just those aspects that a system's authors have chosen to showcase.

2.2. Standard monitoring and control interfaces

Harnessing a wide range of metrics and perturbation injection points within a single evaluation framework requires standard interfaces between the framework and the entities that are measured and perturbed. In building ACME, we found that the sensor and actuator abstractions [15] [21] for collecting data and invoking actions worked well. A user describes her experiment scenario using XML, and ACME translates this description into the appropriate sensor and actuator invocations at the right times. This mapping requires a translation layer between user-friendly abstractions (*e.g.,* "load") and the syntax of those invocations, which are specific to each sensor and actuator. The translation layer also enables the same experiment to run on multiple platforms (*e.g.,* Emulab, Modelnet, and PlanetLab). Such a layer could potentially leverage industry standards such as CIM [8], which provides schemas for describing a wide variety of system components. ACME currently uses an *ad hoc,* platform-specific translation layer for the three platforms on which it runs.

ACME includes sensors that measure for four peer-to-peer overlay networks (Chord [18], FreePastry [16], Tapestry [25], and Bamboo [13]) the benchmark metrics described in Section 2.1, and for Tapestry also application routing table contents and the number of application-level messages routed. ACME also interfaces to sensors written by others that collect operating system metrics such as CPU load and memory consumption. ACME includes actuators that allow experimenters to insert the benchmark perturbations described in Section 2.1 and to start and kill nodes at the beginning and end of the experiment, to copy logfiles to a central node for analysis, to reboot physical nodes, to modify the emulated network topology and link characteristics (latency, bandwidth, and loss rate) on Emulab, to cause application-level message loss, and to modify the rate at which the workload generator attached to each overlay network node instance issues lookup requests.

Thus although we do not believe it is possible to build a universal workload generator or perturbation injector, we have found that the extensibility afforded by standard interfaces to such components has allowed easy integration of data sources, perturbations, and application-specific workloads into our framework.

2.3. Scalable and robust data collection, coordination, and control

Applying a repeatable workload and perturbation load to a large-scale distributed system is challenging. One approach is simply to pre-load every workload-generating node with a time-based workload model and every service-providing node with a time-based perturbation model. Assuming that any randomness in the generators is configurable (*e.g.,* by seeding a random number generator) and that the evaluation infrastructure provides a distributed "barrier" operation so that all nodes can start the experiment at the same time, this approach may be sufficient to enable reasonably repeatable and flexible experiments. But this approach does not permit perturbations that are described as a function of system conditions other than elapsed time. For example, injecting a fault whenever a node detects that a node in its application-level routing table has died, requires coordination between the sensor that provides routing table information and the actuator that injects the fault. Even more challenging are perturbations that trigger on global system conditions, *e.g.,* "keep increasing the workload request rate until the average load across all nodes doubles, and then kill half of the nodes." Executing such rules require a robust event dissemination mechanism to invoke the action and a resilient distributed monitoring facility to detect the specified condition. Note that this monitoring facility must perform double-duty: it writes the bulk of its measurements to local disk for post-mortem analysis (to avoid perturbing the network during the evaluation), and it disseminates the metrics used in user-defined rules as needed to trigger perturbations.

To implement such user-defined experiment control rules, ACME contains a simple distributed query processor that queries sensors for the necessary metrics and aggregates the results as they are routed through a spanning tree overlay network to an "experiment control node." (Metrics not required to evaluate user rules remain where they are collected until post-mortem analysis.) That node also runs ACME's trigger engine that uses the data streams supplied by the query processor, in combination with the user's XML experiment description file, to decide when to invoke the appropriate actuators. (We are currently working on distributing the trigger logic, so that node-local conditions can be detected and responded to autonomously.) To achieve scalability and robustness, ACME uses the Tapestry overlay network for routing, and in-network aggregation of sensor values. Details of ACME's current architecture, implementation, performance, and rule language can be found in [11].

Query processing facilities and event dissemination mechanisms that can meet the scalability and robustness challenges of not just cluster testbeds but also planetary-scale testbeds are active topics of research [4] [5] [9] [12]

[21]. Identifying the properties of such systems that are most important when they are used to monitor and control large-scale system evaluations, and how to specialize them for such uses, remain open questions.

2.4. Turning information into knowledge

Producing informative metrics from large-scale experiments is not a trivial task. If an evaluator's goal is simply to compute end-to-end performance perceived from one vantage point, then only simple analysis facilities are needed. But more complicated metrics for benchmarking and debugging distributed services, such as the number of redundant copies of an application-level data unit, load balance, and consistency of responses, require correlating data from multiple observation points. Additionally, some metrics require correlating measurements across system layers, such as average network bandwidth consumption per application request, or historically across system versions, as when performing regression tests. Ideally the evaluation framework offers a query language that allows users to easily specify such metrics as part of both runtime rules and post-mortem analyses. ACME users can conduct such analyses, excluding historical ones, at runtime using the XML query/rule language with sensors as the data sources, and at the experiment's end using Perl scripts that process log files. We are currently integrating these two facilities. Exactly what an experiment control and analysis language should look like is an open question. Such a language might facilitate not just computation of single metrics, but also visualization and complex statistical analyses [1] [2] [6]. Scaling such analyses to very large systems and distributing them are interesting open problems.

2.5. Network topology and node mapping

To facilitate research and development in wide-area networked services, universities and companies have traditionally constructed off-the-shelf PC clusters outfitted with special network emulation software. Although hardware costs are constantly declining, system management costs are not. This fact, combined with the large number of open source developers unaffiliated with large organizations, lead us to believe that shared testbed infrastructures such as PlanetLab, which distribute management responsibility (and therefore costs) across many sites, will become increasingly popular as evaluation platforms. Because the single network topology of such a platform cannot match the varied topologies of interest to an experimenter, the experimenter will want to select the subset of available nodes that most closely matches the topology of interest for the experiment. This "topology embedding" problem [7] becomes even more interesting when the experimenter adds per-node constraints, as when exploring the robustness of a service to node heterogeneity and/or jobs competing for CPU and memory resources Thus we believe that scalable, robust resource discovery is an essential part of a distributed service evaluation framework. We are beginning to address this in ACME by combining existing monitoring data sources with an optimization framework that matches a user's desired "virtual testbed" to an appropriate subset of the testbed resources that are available.

3. Results from a peer-to-peer overlay network benchmark

To validate the design and usefulness of ACME, we used an early version of it to run the structured peer-to-peer overlay robustness benchmark described in Section 2.1. From this, we discovered a number of interesting properties of the early versions of Tapestry, Chord, and FreePastry that were written, and made available on the web, by those systems' authors. We found, and were able to quantify, the following effects. One of the systems' communication layers was too bandwidth-intensive to start up large (150-node) networks on our emulated topology. All of the remaining systems consumed too much steady-state network bandwidth to run over dialup modems. One of the systems was unable to recover from loss of 20% of the nodes in a 150-node network due to a vicious cycle of background maintenance probes and routing table repair traffic causing congestion that led nodes to believe that nodes that were still alive were actually dead. One system's proactive rebuilding of the routing table upon detection of failed neighbors significantly improved recovery time compared to another system's lazy recovery, but at the expense of higher bandwidth usage. Finally, under high churn, one of the systems experienced a substantial number of timeouts that grew as churn continued.

We do not indicate here which systems showed which of these behaviors, because many of these behaviors have been modified by the authors in the current versions of their software. But note that these benchmarks did not just reveal poorly-set "knobs"--they also provided insights into how to design for higher robustness. For example, they helped to motivate the Bamboo design for resilience under high churn, particularly its congestion-aware recovery and its actively probing neighbors to determine good timeout values for inter-node heartbeats [13]. Some results from this type of evaluation can also be found in [11] and [25].

4. Conclusion

In this paper we have argued that robustness evaluation of wide-area distributed services is important, and that there are challenging research issues in building a reusable infrastructure for such experiments and in defining standard evaluations in the form of benchmarks. Our work in building ACME and defining a structured peer-to-peer overlay network benchmark is but a small step in this direction. We believe that standardized platforms and evaluation criteria for large-scale distributed services will lay the groundwork for high distributed service dependability, while offering researchers from a number of computer science disciplines a host of challenging research questions worthy of their attention.

References

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. *Proceedings of 19th ACM Symposium on Operating Systems Principles*, 2003.

[2] P. Barham, R. Isaacs, R. Motier, and D. Narayanan. Magpie: real-time modelling and performance-aware systems. *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX),* 2003.

[3] A. Bavier, L. Peterson, M. Wawrzoniak, S. Karlin, T. Spalink, T. Roscoe, D. Culler, B. Chun, and M. Bowman. Operating systems support for planetary-scale network services. *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation,* 2004.

[4] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: achieving a global event notification service. *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII),*2001.

[5] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron. SCRIBE: a large-scale and decentralised application-level multicast infrastructure, *IEEE Journal on Selected Areas in Communications*, 20(8), 2002.

[6] M. Chen, A. Accardi, E. Kiciman, D. Patterson, A. Fox, and E. Brewer. Path-based failure and evolution management. *Proceedings of the 1st Symposium on Networked System Design and Implementation,* 2004.

[7] J. Considine, J. W. Byers, and K. Meyer-Patel. A constraint satisfaction approach to testbed embedding services. *Proceedings of HotNets-II*, 2003.

[8] Distributed Management Task Force. Common information model (CIM). http://www.dmtf.org/standards/cim

[9] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. *VLDB*, 2003

[10] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems-MASCOTS,* 2001.

[11] D. Oppenheimer, V. Vatkovskiy, H. Weatherspoon, J. Lee, D. A. Patterson, and J. Kubiatowicz. Monitoring, analyzing, and controlling internet-scale systems with ACME. UC Berkeley Technical Report UCB-CSD-03-1276, 2003.

[12] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *Third International Workshop on Networked Group Communication*, 2001.

[13] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. U.C. Berkeley Technical Report, UCB// CSD-03-1299, 2003.

[14] S. Rhea, T. Roscoe, and J. Kubiatowicz. Structured peer-to-peer overlays need application-driven benchmarks. *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03),* 2003.

[15] T. Roscoe, L. Peterson, S. Karlin, and M. Wawrzoniak. A simple common sensor interface for PlanetLab. PlanetLab Design Node PDN-03-010, 2003.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[17] Standard Performance Evaluation Corporation. http://www.specbench.org/

[18] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM*, 2001.

[19] Transaction Processing Performance Council. http://www.tpc.org/

[20] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *Proceedings of 5th Symposium on Operating Systems Design and Implementation,* 2002.

[21] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: an information plane for networked systems." *Proceedings of HotNets-II*, 2003.

[22] 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. *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[23] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. University of Michigan Technical Report CSE-TR-456-02, 2002.

[24] W. W. Zegura, K. Calvert and S. Bhattacharjee. How to model an internetwork. *Proceedings of IEEE Infocom,* 1996.

[25] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J.Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.