Nines are Not Enough: Meaningful Metrics for Clouds

Jeffrey C. Mogul
John Wilkes
Google Inc.
Mountain View, CA

Abstract

Cloud customers want strong, understandable promises (Service Level Objectives, or SLOs) that their applications will run reliably and with adequate performance, but cloud providers don't want to offer them, because they are technically hard to meet in the face of arbitrary customer behavior and the hidden interactions brought about by statistical multiplexing of shared resources. Existing cloud SLOs are more concerned with defending against corner cases than defining normal behavior. This and other tensions make SLOs surprisingly hard to define. We show that this problem shares some similarities with the challenges of applying statistics to make decisions based on sampled data. We argue that a mutually beneficial set of Service Level Expectations (SLEs) and Customer Behavior Expectations (CBEs) ameliorates many of the problems of today's SLOs by explicitly sharing risk between customer and service provider.

ACM Reference Format:
Jeffrey C. Mogul and John Wilkes. 2019. Nines are Not Enough: Meaningful Metrics for Clouds. In *Workshop on Hot Topics in Operating Systems (HotOS '19), May 13–15, 2019, Bertinoro, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3317550.3321432

1 Introduction

One way to meet several types of strict service-level guarantees is to provision sufficient resources to meet worst-case (peak) demands, and sufficient underlying redundancy to tolerate all conceivable faults. But most cloud customers want low-cost service, and the ability to rapidly adjust up, or down, the resources they are using/paying for. So to meet cost goals, providers cannot overprovision, and must multiplex customers onto more-or-less oversubscribed resources. This means that the interests of cloud customers and providers are inherently at odds (the "Principal-Agent Problem"), and we need contractual agreements, *Service Level Agreements* (SLAs), to re-align their interests.

An SLA is a promise by the provider that, in exchange for payment, it will meet certain customer-visible behaviors known as *Service Level Objectives* (SLOs) or make up for it – an SLA is an SLO plus consequences [3, Ch. 4], typically financial (e.g., refunds). Most commercial SLAs concern availability, and don't address the potential effects of resource under-provisioning, leaving the Principal-Agent Problem unresolved.

Creating an SLA *seems* simple: define one or more SLOs as predicates on clearly-defined measurements (*Service Level Indicators*, or SLIs), then have the business experts and lawyers agree on the consequences, and you have an SLA.

Sadly, in our experience, SLOs are insanely hard to specify. Customers want different things, and they typically cannot describe what they want in terms that can be measured and in ways that a provider can feasibly commit to promising. Real requirements are complicated; a single, one-size-fits-all SLO will likely be unsatisfactory to most customers: far removed from their desires, incomplete, or both. A basket of many simple SLOs could more faithfully reflect their needs, but customers and product managers want simple agreements, and prior work suggests that the "basket" approach could be gamed by an unscrupulous provider [2].

SLOs can also be a blunt tool. Providers want to limit SLA payouts on SLO violations to rare, egregious events; and because lawyers tend to think in binary terms, so are consequences: either a contracted service level was met, or it wasn't. But blunt tools are seldom the best way to manage a provider's internal processes, which involve lots of complex tradeoffs.

We can do better – by thinking less like lawyers and more like statisticians. Suppose we looked at SLOs as a tool to manage a complex set of risks, the same way the medical profession attempts to manage (say) the risks of a cancer treatment vs. its benefits. Lessons from statistics, and decision theory ("the science of making decisions under uncertainty" [6]) might provide us a path towards a principled approach to SLO definition.

In this paper, we will first explain why SLO definition is harder than it might seem. We will then describe an analogy between defining SLOs and doing statistical reasoning. Finally, we will speculate on how to build from that analogy

HotOS '19, May 13–15, 2019, Bertinoro, Italy

ACM ISBN 978-1-4503-6727-1/19/05.
https://doi.org/10.1145/3317550.3321432

towards a principled approach, based on managing risks instead of managing outcomes.

1.1 A little more terminology

To define an SLO, we start with one or more *Service Level Indicators* (SLIs). An SLI is something you can measure; an SLO is a predicate over a set of SLIs. A simple SLO might just be a fixed threshold on a single SLI; for example, that the "monthly uptime percentage" for a VM is at least 99.99%. Here, the uptime percentage is the SLI, and "≥99.99%" is the predicate.[1]

A lot of complexity hides within innocuous-looking phrases such as "monthly uptime percentage." At what granularity is it specified, measured, and aggregated (seconds? minutes?); calendar months or rolling 30-day periods? What does "up" mean: when the VM is provisioned, or actually running an OS?); does the VM have to be reachable from the Internet âĂŞ and how is this tested? Do performance brownouts during VM migrations count as outages? It's even more complicated to define something like uptime of a set of VMs across multiple availability zones in one region, because that depends on the customer's failover model: 2-way redundancy, or N+1?

In addition to customer-facing SLOs, where the consequences are financial penalties, a provider will also define *internal* SLOs, well-defined predicates used to tell if various internal aspects of their systems are functioning as intended. For example, a short-term internal SLO might trigger an alert for operators to take preventive action, long before customers see any impact. A long-term internal SLO could help detect when parts are failing sooner than expected. One can also think of internal SLAs, based on internal consequences: e.g., operators might be overwhelmed by alerts, or receive a bonus for managing the system particularly well.

It is not easy to define internal SLOs, either, or to align internal SLOs with customer-facing SLOs, so that an internal SLO violation will always happen in advance of customer-facing SLO violations.

2 Many flavors of SLOs

Often, technical people focus on the SLOs rather than the SLAs, and therefore treat all SLOs as variations on a single theme. However, in the real world, there are different kinds of consequences, and it is important to define SLOs that are matched to the right kind of consequences.

Previous work [5] suggested dividing SLOs into two categories, "tactical" (short-term, aimed what "what do we need to fix now?") and "strategic" (long-term, aimed at designers of dependent systems). We see a need for a larger set of "flavors," based on a categorization of consequences (in **bold**):

- Contractual SLOs, to support legal agreements, where the consequences are typically **financial penalties** – the traditional definition of an SLA. (Contractual SLAs also send signals to potential customers: "we believe enough in our service's SLO that we're willing to bet real money on it.") Because contractual SLOs are aimed at bad outcomes, they are often far from the expected outcomes, and few customers would be happy if a system only met its contractual SLOs.
- Competitive (or "customer satisfaction") SLOs address this gap, by defining "Service Level Expectations" (SLEs) aimed at keeping customers **happy**. Such SLOs might be tighter than the contractual ones (e.g., 5 nines instead of 4 nines), or cover properties that are too hard or too risky to put into contracts. As a result, SLEs are usually internal (not exposed to customers) – so customers cannot rely on them, and resort to behaviors such as benchmarking to learn what they can expect.
- Compositional uses of SLOs, which system designers can use to understand what they can **depend on**. E.g., an SLO that says that VM failures in two different "availability zones" are uncorrelated allows the designer of a higher-level system to design fault-tolerance mechanisms and to estimate the risk of failure for that system. (Sometimes these are contractual, sometimes they are internal.)
- Control loop SLOs, which allow a provider to **actively manage its systems** in order to meet other kinds of SLOs. E.g., a control-loop SLO violation could alert an operator to repair something before there are any violations of other SLOs. On an even shorter timescale, an SLO for network link utilization could trigger automated shedding of low-priority load. On a longer timescale, a capacity planner could look at increases in resource utilization as a signal to order more equipment. A vice president might look at an SLO for help-desk response times to decide when to hire more technicians – or when to replace their boss.

It is usually a mistake to try to support two or more kinds of flavors with the same SLO, because the appropriate SLIs and/or predicates differ (especially in timescale). The resulting compromise satisfies nobody; attempts to craft a compromise leads to long, fruitless discussions when the parties do not even realize they are trying to solve two distinct problems with one SLO.

While only the first flavor of SLOs ("contractual SLOs") is typically associated with formal, lawyer-defined SLAs, we suggest that it is still useful to think of all flavors in terms of informal consequences (e.g., "my pager won't go off" or "I won't get fired"), because that helps to focus on exactly what SLOs we need, and how carefully they need to be specified.

We have seen attempts to use SLOs to solve problems for which SLOs are inappropriate. For example, debugging

[1] These examples often involve quantities like 99.9% or 99.99%. People use the shorthand "3 nines" or "4 nines", respectively, to refer to those percentiles, or "4 and a half nines" to describe 99.995%. This shorthand can obscure the huge difference that "adding a nine" implies; going from 99.99% to 99.999% uptime means reducing downtime from 263 to 26.3 seconds per month, which, when sustained for a cloud-scale system over long periods, is an astounding feat of engineering.

should be based on SLIs, or other non-SLI measurements (such as packet traces), not SLOs, because the use of aggregation or thresholds in the definition of an SLO's predicate inherently loses most of the data from the SLIs.[2] If debuggers need predicates to filter out noise in SLIs, these should be treated as *sui generis* filters, not as an added requirement for a true SLO.

3 What makes SLO definitions hard?

In our own attempts to define meaningful, practical SLOs, we have learned several lessons. (Here, "meaningful" means that customers can understand and benefit from an SLO; "practical" means that one can accurately and cheaply collect the SLI data, and process it to compute the SLO's predicate.) These lessons include:

- We need SLOs because we want to protect against the consequences defined by SLAs. Even within one "flavor" as described above, there are many kinds of consequences. Therefore, we need lots of SLOs (often, multiple SLOs per flavor).
- However, SLOs are not free. We have to spend resources collecting and processing the data that allows us to compute an SLO's predicate, without significantly interfering with "real" work – and without compromising security or privacy of cloud customers and their own users. We often must aggregate data to reduce costs, which loses fidelity. (These costs are what we mean by the "feasibility" of a set of SLOs.)
- Customers (and operators) don't like dealing with too many SLOs, which creates pressure to limit the number of SLOs âĂŞ especially the externally published ones.
- Customers must also be able to understand how to make use of the SLOs they are offered. A "meaningful" compositional SLO should support clear decisions – for example, how much capacity a customer needs to purchase, or how many VM replicas are required to reach a desired application-level availability. A contractual SLO that involves complex predicates, although well-defined, might be too confusing to be truly meaningful.
- The cloud business is price-sensitive, so providers must typically multiplex their resources, creating a tension between predictability and cost (predictability is not free!), but this relationship is not easy to express in a closed form.
- Some SLOs depend on customer behavior âĂŞ e.g., anything involving a guarantee about network throughput relies on the customer running software capable of driving the network fast enough. (This is one reason why performance SLOs are surprisingly rare.)
- System availability as seen by end users depends on careful exploitation of redundancy (because the alternative – failure-proof systems – is either too expensive or entirely impossible). Redundancy only works when one can avoid correlated failures of the resources that a system depends on, but it is not always easy to understand these dependencies. Given the wide variety of possible correlations, writing SLOs about non-correlation is a complex art.

We have encountered a few issues specifically related to the use of "nines" to express availability SLOs:

- Using average uptime metrics over a period (e.g., the number of "bad minutes" per month) hides the distinction between many short outages and a few long ones. Yet many applications/users actually care about this distinction – distributed-systems algorithms can magnify the impact of a "short" outage (e.g., BGP convergence, or state-heavy services which have long initialization times), while humans (e.g., bank ATM customers) might better tolerate a frequently-but-slightly degraded overall experience (waiting an extra 15 seconds for their cash) rather than a long outage (making a second trip to the ATM).
- "Nines" treats outages as generic problems; one outage is the same as another if they last the same length of time. But for real-world applications, that's not always true. For example, for a retailer, an outage on Black Friday is a lot worse than the same outage on a boring day in March or planned downtime. Cloud providers know that if they want to retain their customers, they have to support stronger SLOs on certain dates, rather than treating a formal SLA as the only constraint.
- Even when a system is not formally meeting its SLO, we generally prefer "graceful degradation"; a system that is "almost within SLO" is usually better than nothing, but it is hard to formalize this concept within the "nines" approach.

In general, SLOs are not just single SLIs with a simple threshold; they are complex predicates over complex SLIs. For example, we might want to define a compositional SLO over a time series rather than a scalar; we might need statistical functions (mean, median, nth percentile, min, max) depending on the application.

A consequence of the Principal-Agent Problem is that customers should not trust a provider to report whether it is meeting its SLOs, and they should prefer SLOs that can be independently validated without specific assistance from the provider. This is especially difficult for SLOs, such as stored-data integrity, where consequences (loss of data) cannot be observed until years or decades after the fact [7].

4 Lessons from statistical thinking

A good statistician will look at what decision needs to be made, define hypotheses to test in order to make the decision, decide how to collect sufficient data without bias,

[2] For example, if you are using ML-based anomaly detector to spot unanticipated failures, its input data should include as much detail as possible, and should not be aggregated or thresholded based on your prior assumptions about what SLI values indicate problems.

often sampled from the underlying population while staying within a budget, choose an appropriate method to test the hypotheses against the sample. We call this the "professional statistician method."

"Choosing an appropriate method" is the part of statistics where one most needs deep training, careful reasoning, and honest creativity. Poorly-trained statisticians often fall back on the few methods they know, without understanding if these are appropriate to the task at hand.

This suggests a close analogy between SLO definition and statistics:

- Given that an SLA is an SLO with consequences, the *decision* we need to make is "should we invoke those consequences, or not?" Crisply defining this question is called "operationalizing" the decision (a term from the social sciences that made its way to statistics via psychology); in SLOs, the challenge is knowing exactly what is "meaningful" to the customer or provider.
- The problem of gathering accurate SLI data, while not using up too many resources, is analogous to *sampling* the underlying population. Neither in statistics nor SLOs do we have unconstrained freedom to gather all the data we might possibly want, or even to know if it might be biased by unexpected phenomena. Trained statisticians know how to avoid mistakes in this phase, such as biased sampling.
- What remains is to define the SLO as a predicate on the available SLIs, which is roughly analogous to choosing an inference method, the deepest expertise of the expert statistician – so, if this analogy is valid, it should not be surprising that defining SLOs is not as easy as it looks. It also requires direct use of statistical skills: using the appropriate aggregation mechanisms, and applying the appropriate statistical hypothesis tests to make a decision about something that is not directly measurable.

Perhaps the most important lesson we can learn from statisticians is to realize that we can never be fully confident when reasoning under uncertainty, and sometimes we must admit defeat. "Indeed, the very humility of statistics is its salvation." [1]

By analogy to the professional statistician method, a professional SLOgician's approach to defining SLOs would be to:[3]

- List the good outcomes you want and bad outcomes you want to avoid (e.g., lack of network capacity, correlated failures, stored-data loss, disk failure rates, mis-predicted demands, late capacity delivery, etc.).
- Agree with business decision-makers what the consequences should be (e.g., the size of a refund).
- Operationalize these outcomes – e.g., deciding what level of network capacity or stored-data loss you need to promise to customers, to attract their business.
- Decide what kind of data to collect in order to decide whether you are suffering from one of those bad outcomes, and what kinds of aggregation are needed/possible, in order to be able to test whether the SLO is being met (i.e., what are the SLIs needed to evaluate each SLO predicate?). Deciding how much data you need is analogous to what statisticians call "power analysis."
- Decide what specific predicate on that data tells you whether an outcome has happened – this is analogous to choosing the right method for hypothesis testing. Much of the challenge is in choosing the numeric thresholds, because these represent a negotiated tradeoff between costs and outcomes.
- Decide how much of the desired data you can collect, given your resource budget, and then deciding whether that is enough to actually compute the SLOs. For example, if you have an SLO that network throughput measured over a 1-minute interval will never drop below some threshold, but you can only afford to collect data in 5-minute aggregates, you have a problem.

Just as "statistician" and "data scientist" are distinct roles that share many, but not all, skills [4], "SLOgician" is also a distinct role with its own specific skills.

Suppose a provider wants to compute many SLOs, but does not have the measurement budget to support all of those at once; how can one resolve this conflict?

- Accept that fewer SLOs is OK, and use the consequences to prioritize the most important ones. Providing metrics instead of SLOs is often enough.
- Accept lower confidence in determining whether the SLOs are being met, when this can reduce measurement load – e.g., by reducing the sampling frequency or increasing the aggregation window.
- Dynamically lower a measurement rate if an SLO is not at risk of violation; raise the rate when risks are high.

5 Are we barking up the wrong tree?

Our experiences trying to define performance-oriented SLOs for networks, VMs, and storage systems have convinced us that such SLOs are hard to define, that many depend on decisions (such as the choice of congestion-control algorithm) outside the provider's control, and that we seem to have way too many of them. Perhaps this is a sign that we are solving the wrong problem.

Following our analogy with statistics, we could focus less on SLOs that guarantee outcomes (e.g., network throughput or query performance), and instead use SLOs as a tool for providers to provide structured guidance about making decisions that create or remove risk. That is, we should focus SLOs on risks that are entirely under the provider's control – a provider's job, with respect customers, is to mitigate risks and make them understandable, not to ensure complete happiness.

[3] Such people practice SLOgistics, of course.

If we could ignore resource sharing, we could focus SLOs on various risks that arise from poor engineering or operational practices, such as not repairing control-plane outages; SDN designs that allow short-term control-plane failure to disrupt the data plane; failover mechanisms that do not actually work; operational procedures that create correlated risks, such as simultaneous maintenance on two availability zones in a region; routing network packets along surprisingly long WAN paths.

It should be relatively simple to write SLOs addressing this kind of "sharing-agnostic" issue, because while they sometimes require reasoning about probabilities, they do not require reasoning about customer behavior.

If we could offer only sharing-agnostic SLOs, we could reason about system composition by assuming that virtual CPUs, networks, and storage behave similarly to real ones, but without the hubris of offering quantified "outcome SLOs." This would still require the provider to expose some aspects of the real hardware: our expectations for a rotating disk reasonably differ from our expectations for an SSD, and likewise we have reasonably different SLEs for LANs, MANs, and WANs.

However, we *cannot* ignore resource sharing, and thus the tension between predictability and cost, in the face of unpredictable customer behaviors. Sharing creates risks such as: unpredictable CPU performance due to a shared L3 cache or memory bus; unpredictable network throughput or packet loss due to link overutilization or queue overflow; or unpredictable storage performance due to disk-bandwidth or disk-seek overutilization. The analogy to statistics, the science of reasoning under uncertainty, is especially obvious here – we have uncertain knowledge of future workloads, and we also are unable to accurately model the performance of complex computer systems even if we did know the workloads.

5.1 Coping with customer behavior

Even if we address catastrophic failures by defining SLAs for sharing-agnostic risks, we still must manage the conflict between resource-sharing (for efficiency) and predictable outcomes. Uncertainty about customer behavior makes it hard for a provider to define such "sharing-dependent" SLOs – so, let's change the rules of the game to sidestep uncertainty.

Instead of focusing on outcomes, we should focus on expectations, and make these expectations bilateral: what service level the customer can expect from the provider (an SLE), and what the provider can expect from the customer (*Customer Behavior Expectations*, or CBEs). An SLE *only* applies if its related CBEs are met. Our view is that the customer and provider should each bear part of the risk of unpredictability, and use SLEs and CBEs to explicitly manage the sharing of risks.[4]

Providers already have some CBEs; e.g., they limit a VM's egress network bandwidth, or how many VMs a customer can run at once. But contractual SLOs are not typically predicated on customer behavior, and many customer behaviors that could lead to resource conflicts are inherently hard to enforce efficiently in real-time.

But after-the-fact detection of a CBE violation is often acceptable, especially for CBEs that are hard to enforce. For example, while we cannot prevent a customer VM from generating lots of L3 cache misses, CPU counters can tell us that this is happening. And while we cannot prevent a customer from generating a network incast, switch-based metrics can tell us when incast has happened, and which source(s) caused it.

Contractual SLAs could thus be structured with obligations in both directions: for example, the provider's obligation to limit LAN packet loss is removed whenever the customer seems to be generating incast traffic. And, one step further, customers could either agree to pay additional fees if they unexpectedly create bursts of traffic that interfere with other customers, or accept stricter rate limits in exchange for more predictable bills.

Then one could limit sharing-dependent SLOs to be only compositional, not contractual – that is, sharing-dependent SLOs are offered as guidance: the provider implicitly promises not to undermine well-accepted SLEs, but makes no enforceable promises (SLAs) about sharing-dependent outcomes.

SLEs cover likely behavior, rather than worst cases. Most commercial SLAs focus on the latter, which means that there are no clear, enforceable expectations for "normal behavior." CBEs can help here, too: by eliminating worst-case customer behavior (from the provider's point of view), they permit tighter bounds on average or median behavior. Providers can use CBEs to signal to consumers the behavior they can optimize for, rather than what they have to tolerate.

While they may feel like additional restrictions, CBEs and joint risk sharing can move customers and providers closer towards mutually-beneficial collaboration, and further away from lawyers. The downside is that we need more SLOs/CBEs to define behaviors. This should not be a surprise: real systems have immense complexity. We should not expect to describe them via just a few numbers without a huge loss of fidelity.

5.2 CBEs vs. burstable instances

Several cloud providers offer "burstable instances" for VMs, which are complementary to CBEs. A burstable instance provides a performance SLO equivalent to a partial CPU

[4] By analogy, a consumer using a medicine typically bears some of the risk of side effects ("do not operate heavy machinery") to themselves or others, as well as the risk that the medicine does not actually cure them, while the pharmaceutical company accepts the risks associated with manufacturing errors.

core, but offers a way for a customer to occasionally exploit the work-conserving nature of that shared resource, with a disincentive to take this extra capacity for granted. As far as we know, providers offers no guarantee that any extra capacity will be available, although they can optimize VM placement to benefit from statistical multiplexing of uncorrelated workloads.

6 Open questions

We list here a few more open questions for discussion:

- We argue for treating sharding-agnostic and sharing-dependent risks differently, but how do we define the boundary between these? – given that the decision about which resources to share is left to the provider, and can vary by cloud-product differentiation (e.g., spot instances vs. high-availability instances). SLO definitions must also evolve as novel network features (such as RDMA within distance-constrained neighborhoods) create new trade-offs.
- SLO definitions must evolve over time, as customer behavior and expectations change, and as providers improve their infrastructure or discover new failure modes. It may be impossible to define SLOs, especially their thresholds, purely by reasoning from first principles; rather these must be defined through an incremental-improvement process grounded in a deep understanding of the provider's systems [3, Ch. 4]. Or could there be a principle-driven method?
- Suppose customer X has unexpected L3 cache behavior that affects customer Y, and the provider fails to detect this CBE violation and the resulting interference. Clearly customer Y, in this situation, should be able to call "foul" and perhaps invoke the legal system as the final arbiter – but by what cost-effective mechanism could Y detect or prove the SLO violation? Perhaps we must require providers to include support for third-party audits [7] – while one would prefer to avoid legal proceedings, when they become necessary they must be based on solid data.
- Can we validate "SLO quality" by comparing an SLO's history to separately-measured evidence of system and customer well-being (e.g., outage history, support tickets, etc.)? How can we analyze our SLOs to ensure they correctly support the decisions we are trying to make with them?

7 Putting it all together

There are good times; there are bad events. Cloud customers need well-defined SLOs to avoid bad events, but the quest to create outcome-oriented SLOs, for application performance on multiplexed resources, has challenged many smart computer scientists and engineers in multiple fields – storage, networking, and computation. We do not pretend to have a complete solution to the problems of cloud-SLO definition, but we think such a solution could emerge from re-thinking our use of SLOs, and using the combination of SLEs and CBEs to create harmonious cooperation in normal times.

We hope this paper has exposed some of the issues underlying the difficulties of defining outcome-oriented SLOs for shared resources, and how this area may resemble the fields of statistics and decision science – suggesting that there is room for a new field of expertise within computer systems, analogous to the role of statisticians (with a similar need to create a structured corpus of knowledge and practice).

Perhaps the most important lesson we can learn from statistics, however, is humility – that the combination of unpredictable workloads, hard-to-model behavior of complex shared infrastructures, and the infeasibility of collecting all the necessary metrics means that certain kinds of SLOs are beyond our power to deliver, no matter how much we believe we need them.

Acknowledgments

We are grateful for insightful contributions from Cassie Kozyrkov, Steven Hand, and Yaniv Aknin, and thoughtful comments from the anonymous reviewers.

References

[1] The Function of Statistics. The Nation. [81:2094] pp. 137–138, August 17, 1905.

[2] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. The Price is Right: Towards Location-independent Costs in Datacenters. In *Proc. HotNets*, 2011.

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy, editors. *Site Reliability Engineering.* O'Reilly Media, Inc., 2016.

[4] Cassie Kozyrkov. Top 10 roles in AI and data science. https://hackernoon.com/top-10-roles-for-your-data-science-team-e7f05d90d961, 2018.

[5] Jeffrey C. Mogul, Rebecca Isaacs, and Brent Welch. Thinking About Availability in Large Service Infrastructures. In *Proc. HotOS*, 2017.

[6] Leonard J. Savage. *The Foundations of Statistics.* New York, 1954.

[7] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to Keep Online Storage Services Honest. In *Proc. HOTOS*, pages 11:1–11:6, 2007.