Meaningful Availability

Tamás Hauer, Philipp Hoffmann, John Lunney, Dan Ardelean, and Amer Diwan, *Google*

https://www.usenix.org/conference/nsdi20/presentation/hauer

This paper is included in the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)

February 25–27, 2020 • Santa Clara, CA, USA

978-1-939133-13-7

Open access to the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20) is sponsored by

Meaningful Availability

Tamás Hauer *Google*
Philipp Hoffmann *Google*
John Lunney *Google*
Dan Ardelean *Google*
Amer Diwan *Google*

Abstract

High availability is a critical requirement for cloud applications: if a sytem does not have high availability, users cannot count on it for their critical work. Having a metric that meaningfully captures availability is useful for both users and system developers. It informs users what they should expect of the availability of the application. It informs developers what they should focus on to improve user-experienced availability. This paper presents and evaluates, in the context of Google's G Suite, a novel availability metric: windowed user-uptime. This metric has two main components. First, it directly models user-perceived availability and avoids the bias in commonly used availability metrics. Second, by simultaneously calculating the availability metric over many windows it can readily distinguish between many short periods of unavailability and fewer but longer periods of unavailability.

1 Introduction

Users rely on productivity suites and tools, such as G Suite, Office 365, or Slack, to get their work done. Lack of *availability* in these suites comes at a cost: lost productivity, lost revenue and negative press for both the service provider and the user [1, 3, 6]. System developers and maintainers use metrics to quantify service reliability [10, 11]. A good availability metric should be *meaningful*, *proportional*, and *actionable*. By "meaningful" we mean that it should capture what users experience. By "proportional" we mean that a change in the metric should be proportional to the change in user-perceived availability. By "actionable" we mean that the metric should give system owners insight into why availability for a period was low. This paper shows that none of the commonly used metrics satisfy these requirements and presents a new metric, *windowed user-uptime* that meets these requirements. We evaluate the metric in the context of Google's G Suite products, such as Google Drive and Gmail.

The two most commonly used approaches for quantifying availability are *success-ratio* and *incident-ratio*. Success-ratio is the fraction of the number of successful requests to total requests over a period of time (usually a month or a quarter) [5, 2, 9]. This metric has some important shortcomings. First, it is biased towards the most active users; G Suite's most active users are 1000x more active than its least active (yet still active) users. Second, it assumes that user behavior does not change during an outage, although it often does: e.g., a user may give up and stop submitting requests during an outage which can make the measured impact appear smaller than it actually is. Incident-ratio is the ratio of "up minutes" to "total minutes", and it determines "up minutes" based on the duration of known incidents. This metric is inappropriate for large-scale distributed systems since they are almost never completely down or up.

Our approach, *windowed user-uptime* has two components. First, user-uptime analyzes fine-grained user request logs to determine the up and down minutes for each user and aggregates these into an overall metric. By considering the failures that our users experience and weighing each user equally, this metric is meaningful and proportional. Second, windowed user-uptime simultaneously quantifies availability at all time windows, which enables us to distinguish many short outages from fewer longer ones; thus it enables our metric to be actionable.

We evaluate windowed user-uptime in the context of Google's G Suite applications and compare it to success-ratio. We show, using data from a production deployment of G Suite, that the above-mentioned bias is a real shortcoming of success-ratio and that windowing is an invaluable tool for identifying brief, but significant outages. Our teams systematically track down the root cause of these brief outages and address them before they trigger larger incidents.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 motivates the need for *windowed user-uptime* Section 4 describes user-uptime. Section 5 extends user-uptime with windowed user-uptime. Section 6 evaluates our approach and Section 8 concludes the paper.

2 Related Work

Understanding and improving availability of computer systems has been a goal for decades [22] with early work aiming at developing stochastic models based on failure characteristics of individual hardware and software components [30, 20, 27, 21] . A more abstract approach [28] uses mean time to failure (MTTF) and mean time to recovery (MTTR), which are well suited to describe a system that shifts between binary up/down states. An extension to this approach that differentiates between transient failures and short-term downtime can be found in Shao *et al.* [26].

When describing complex distributed systems, however, a binary up/down state is usually not accurate. Brown *et al.* [13] focus on success-ratio instead, describing the availability of a system as the percentage of requests served successfully and understanding it as a spectrum between up and down.

Brevik *et al.*[12] show how to compute confidence bounds for future performance based on collected past performance data. Treynor *et al.* [29] describe error budgets connected to an availability goal and how availability of individual components affects the whole system.

In taking a user-centric approach, Dell'Amico *et al.*[18] explore problems similar to those presented here, but with a focus on prediction and future performance. In a sequel[17], a probabilistic model is used to suggest optimizations to a network application.

A common application for availability and error budgets are contracts for cloud service providers [5, 2, 9]. Patel *et al.* [25] discuss these Service Level Agreements (SLAs) focusing on a framework to define and monitor metrics and goals associated with SLAs while Endo *et al.* [19] describe issues encountered when trying to achieve high availability in cloud computing.

Microsoft Cloud services (such as Office 365) compute and report uptime as the time when the overall service is up divided by total time. [4] As a metric based on time, it is immediately meaningful (e.g., everyone interprets 99.9% uptime as "system will be down for a total of 1 day every 1000 days on average").

Before the work in this paper, Gmail computed and reported availability as the percentage of successful interactive user requests (irrespective of which user the request comes from). Google Cloud Platform computes and reports "downtime" as the error rate and downtime period as consecutive minutes of downtime.

Slack's status page reports that it is "a distributed platform and during any given incident it is rare for all Slack teams to be affected. For this reason, we report our uptime as an average derived from the number of affected users." [7]

Amazon Web Services computes and reports "Error rate" as the percentage of requests that result in errors in a 5 minute interval. The average of these five minute calculations is reported to customers, as "Monthly uptime percentage" - the average of all the 5-minute error rates. [2]

In the following sections, we will look at three desirable properties of an availability metric: meaningful, proportional and actionable. The surveyed systems each satisfy some but not all of these three properties.

3 Motivation

Availability is the ability of a service to perform its required function at an agreed instant or over an agreed period of time [24]. At a high level, all availability metrics have the following form:

$$\text{availability} = \frac{\text{good service}}{\text{total demanded service}} \tag{1}$$

Availability metrics are invaluable to both users and developers [10].

For users, they tell them whether or not a service is suitable for their use case. For some use cases, unavailability translates into lost productivity and a *meaningful* measure can help quantify that.

For developers, availability metrics help prioritize their work to improve their system. For this to be the case, the measure should be *proportional* and *actionable*. A proportional metric enables developers to quantify the benefit of an incremental improvement. An actionable metric enables them to zero in on episodes of worst availability and thus find problems that they need to address.

Broadly speaking, availability metrics fall in two categories: *time based* and *count based*, according to how they quantify the "good service" and "total demanded service" in Eq. (1).

3.1 Time-based availability metrics

Toeroe *et al.* [28] define availability as $\frac{\text{MTTF}}{\text{MTTF+MTTR}}$ where MTTF is the mean-time-to-failure and MTTR is mean-time-to-recovery. This measure is based on the duration when the system is up or down; these concepts are meaningful to users. Thus it is no surprise that many cloud providers, e.g. Microsoft's Office 365 [4], use it. The time between failures is uptime and the time to recover from a failure is downtime, thus we can express this metric as Eq. (1) once we identify uptime and downtime as the measure of good and bad service, respectively [29]:

$$\text{availability} = \frac{\text{uptime}}{\text{uptime} + \text{downtime}}. \tag{2}$$

At a deeper level, this measure depends in a complex way on the precise meanings of "failure" and "recovery". At one extreme we could consider a system to have a failure only

when there is an outage that affects *all* users. At the other extreme, we could consider a system to have a failure when there is an outage that affects even a *single* user.

Neither of these definitions are adequate. The first is unsatisfactory because the design and deployment of cloud systems, such as Azure [14], Dynamo [16], or Gmail, actively avoid single points of failure by sharding data across many machines and using replication [23] with failover when problems occur. Consequently, these systems rarely have an outage that affects all users [29]. The other extreme is not suitable, particularly for systems with hundreds of millions of users, because there will always be some that are experiencing failures. The system may not even be at fault for such outages. For example, we once debugged a situation where a user had authorized access to their Gmail account to many third-party services, each of which was repeatedly making extremely large requests against the user's account, inadvertently and collectively causing a denial of service attack, and in turn an outage for that user. It would be unreasonable to consider this as an outage of Gmail.

Consequently, availability in terms of up and down states either require manual labeling of up and down times (e.g., the incident ratio metric) or the use of thresholds, e.g., the G Suite SLA defines downtime as "if there is more than a five percent user error rate" [5]. While such thresholds avoid the extremes above, they do so arbitrarily. Furthermore, availability metrics that use such thresholds are not proportional. For example, the G Suite definition treats a system with 5% error rate the same as a system with 0.0001% error rate; and it treats a system with 5.1% error rate the same as a system with 99% error rate.

Thus, commonly used time-based availability metrics:

- are not proportional to the severity of the system's unavailability (a downtime with 100% failure rate weighs as much as one with 10%).
- are not proportional to the number of affected users (a downtime at night has the same weight as a downtime during peak period).
- are not actionable because they do not, in themselves, provide developers guidance into the source of failures.
- are not meaningful in that they rely on arbitrary thresholds or manual judgments (e.g., incident-ratio).

3.2 Count-based availability metrics

While the most common definitions of availability (including the ones discussed above) are in terms of time, some systems use "success-ratio" instead. Success-ratio is the ratio of successful requests to total requests. Since success-ratio does not use a threshold, it is more proportional than commonly-used time-based metrics.

Success-ratio as an availability measure is popular because it is easy to implement and is a reasonable measure: it accurately characterizes a service, whose (un-)reliability stems solely from a stochastic distribution of failures. Because the computation is based on user requests, it approximates user perception better than if we used some internal instrumentation to reason about the service's health. It is, however, prone to bias: the most active users are often 1000x more active than the least active users and thus are 1000x over-represented in the metric.

Even if we disregard the bias with respect to different users' activity, this metric can fail to proportionally capture changes in availability. Consider, for example, that a service is down for 3 hours and then up for 3 hours. While the system is up, users are active on the system and thus make many (successful) requests to the system. While the system is down, they will periodically probe the system to check if is still down but the inter-arrival time of the requests is likely lower than if the system is in active use. For certain systems, particularly ones that provide a service to other systems, the opposite situation may hold: a client service may flood our system with retries thus inflating failure count. Users ultimately care about time, thus a count-based measure - in contrast to a time-based one - can misrepresent the magnitude of an outage.

In summary, count-based (success-ratio) availability metrics:

- are not meaningful in that they are not based on time.
- are biased by highly active users.
- are biased because of different client behavior during outages.

3.3 Probes

Using synthetic probes may mitigate some of the shortcomings of success-ratio. For example, probing the system automatically at regular intervals can avoid bias. This approach works well for low-level systems with modest business logic, for example, a network which "just" sends and receives packets. In contrast, a cloud application may have hundreds of types of operations and the work required by an operation type depends on the type and the user's corpus (e.g., a search operation for a user with a few documents is fundamentally different, in terms of the work involved, from a search for a user with millions of documents). Consequently, despite years of effort, we have been unable to fully represent real workloads on our system with synthetic probes [8]. Thus, for cloud applications, a metric which uses synthetic probes may not be representative of real-user experience.

In summary, availability metrics based on synthetic probes:

- are not representative of user activity.
- are not proportional to what users experience.

3.4 Actionable metrics

The availability metrics that we have described so far represent different points in the tradeoffs for "proportional" and "meaningful". All of them, however, have a similar weakness when it comes to being "actionable": a single number associated with a reporting period does not allow enough insight into the source or the shape of unavailability.

Indeed, none of the existing metrics can distinguish between 10 seconds of poor availability at the top of every hour or 6 hours of poor availability during peak usage time once a quarter. The first, while annoying, is a relatively minor nuisance because while it causes user-visible failures, users (or their clients) can retry to get a successful response. In contrast the second is a major outage that prevents users from getting work done for nearly a full day every quarter.

In the following section, we describe a new availability measure, user-uptime that is meaningful and proportional. Afterwards, we'll introduce windowed user-uptime, which augments it to be actionable.

4 Proportional and meaningful availability: user-uptime

As discussed in Section 3, prior metrics for availability are not meaningful or proportional. A seemingly straightforward variant of Eq. (2) satisfies both proportionality and meaningfulness:

$$\text{user-uptime} = \frac{\sum_{u \in \text{users}} \text{uptime}(u)}{\sum_{u \in \text{users}} \text{uptime}(u) + \text{downtime}(u)}, \quad (3)$$

Since this metric is in terms of time, it is meaningful to users. Since it is free from arbitrary thresholds, it is (inversely) proportional to the duration and magnitude of the unavailability. The calculation of this metric, however, is not straightforward as it requires a definition of uptime(u) and downtime(u) *per user*. The remainder of this section describes how we calculate these.

4.1 Events and durations

An obvious approach to uptime and downtime would be to introduce evenly spaced synthetic probes for *each* user (Figure 1) and count the successful and failing probes. In Figure 1 the light green circles are successful events and dark red diamonds are failed ones from a single user's perspective. The green horizontal line at the top marks the period when the user perceives the system to be up (i.e., its length represents uptime) while the red one on the bottom marks the period when the user perceives the system to be down. As discussed in Section 3.1, however, synthetic probes fail to

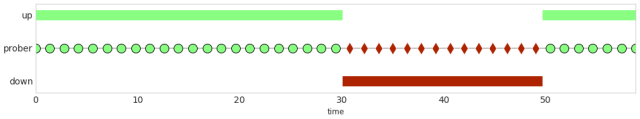

Figure 1: System availability as seen through evenly-spaced prober requests. success-ratio=67%, user-uptime=67%

mimic true user behavior. For example, we may probe certain operations but users may experience unavailability for other operations. In a recent Gmail outage, for example, the failure of the system serving attachments impacted only operations that accessed attachments; but other operations were largely unaffected.

Our key insight, therefore, is to use user requests as probes. A user's perception of a system being up or down depends on the response they get when they interact with it: if the response is successful (unsuccessful) they perceive the system as being up (down). For example, as long as the user can successfully edit a document, she perceives the system to be up; once she experiences a failing operation she perceives the system to be down.

Success versus failure of individual events is not always obvious to the user: for example, if a device's request to synchronize the contents of the user's mailbox fails, the user may not notice that the system is down. Despite this, we want our availability metric to be conservative: if there is *any* chance that a user may perceive a failure (even one as subtle as a longer-than-usual delay in the notification for a new email) we consider it as a failure.

Fig. 2 illustrates this approach. In contrast to Fig. 1, availability is not necessarily the same as success-ratio. Section 4.2 discusses some variations of this approach and the consequences of our choices.

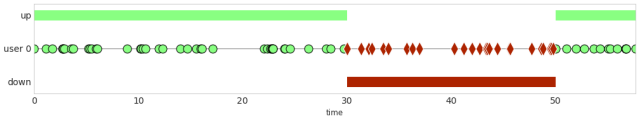

Figure 2: Availability as seen through unevenly spaced requests from a single user. success-ratio=68%, user-uptime=65%

Fig. 3 illustrates a system with four users. Each user experiences an outage of a different duration and generates requests at a different rate from the other users. Thus if we use success-ratio across the users, it will skew availability towards the most prolific user. User-uptime does not suffer from this bias.

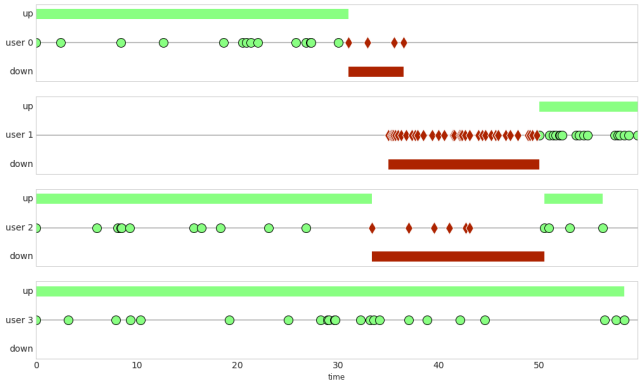

Figure 3: An outage which affects users selectively.

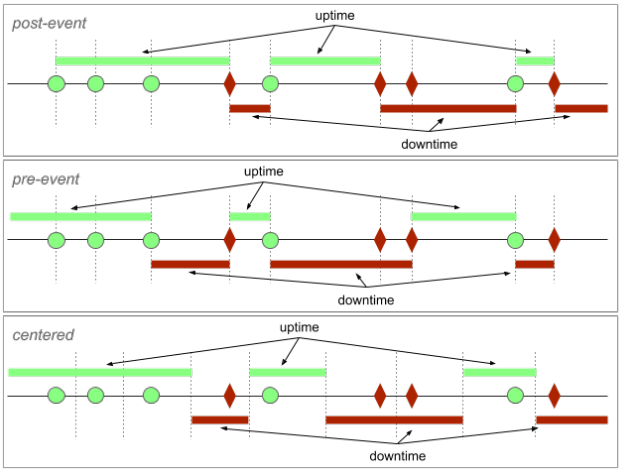

Figure 4: Three choices to extrapolate uptime or downtime from neighboring events

4.2 Challenges with user uptime

While we have illustrated user-uptime using examples, there are two main questions that we have glossed over: (i) how do we label a duration as up or down; and (ii) how do we address users that are inactive. The remainder of this section addresses these challenges.

4.2.1 Labeling durations

As long as back-to-back events are both failures or successes, it is obvious how to label the duration between the two events. But if back-to-back events are different (i.e., one is a failure and one is a success) there are three choices (Fig. 4):

- After a successful (or failing) operation, assume that the system is up (or down) until the user sees evidence to the contrary.
- Before a successful (failing) operation, assume that the system is up (down) until the previous event. Intuitively, this option takes the position that the user request probes the state (up or down) of the system up to the previous operation.
- Split the duration between events (i.e., half the time is uptime and half is downtime).

Assuming that transitions (from up to down or vice versa) occur at random points between the events, the differences between the above three options will be negligible. This is not always the case when the client aggressively retries failing operations. For simplicity, and because it captures user intuition of system availability, we use the first option.

4.2.2 Active and inactive periods

If a user stops all activity for an extended period (e.g., goes on a vacation) they have no perception of the system being up or down: assuming that the system is up or down continuously after the last seen request does not make sense and may even optimistically bias the data: e.g., if a user has an unsuccessful request they may retry until they get success and thus the last request before a vacation is disproportionately likely to be successful.

To this end, we introduce a *cutoff* duration; if a user has been inactive for more than this duration, we consider the user as being inactive on the system and thus do not count uptime or downtime. We pick the cutoff duration as the 99th percentile of the interarrival time of user requests. For Gmail this is 30 minutes. Experiments with multiples of this duration have shown that our availability metric does not change significantly with different values of the cutoff duration.

Now we can define how we label durations as uptime or downtime for each user:

Definition (uptime, downtime): A segment between two consecutive events originating from the same user is:

- *inactive* if the two events are further apart than `cutoff`, otherwise
- *uptime* if the first of the two events was *success*
- *downtime* if the first of the two events was *failure*.

Fig. 5 illustrates this definition.

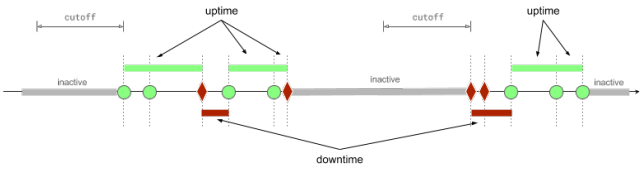

Figure 5: Definition of uptime and downtime segments

For each user u and a measurement period of interest, uptime(u) is the sum of the lengths of the uptime segments and downtime(u) is the sum of the lengths of the downtime

segments. We can calculate overall availability for the system by aggregating across all users as in Eq. (3).

4.3 Properties of user-uptime

We now study the properties of user-uptime and success-ratio in synthetic situations where we know the ground truth; later (Section 6) we evaluate them in a production system.

For the first example, we generated synthetic user requests for an hour such that all user requests within a contiguous 15 minute period failed. We simulated this thousands of times for 10,000 synthetic active users. Fig. 6 illustrates what this outage looks like for 2 users.

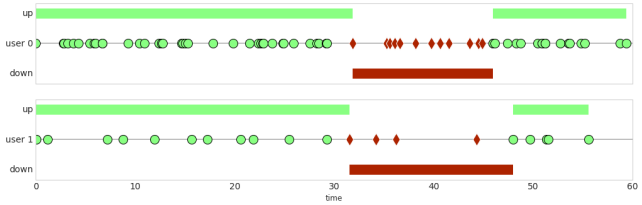

Figure 6: Two users' activity around a hard outage during $30 < t < 45$

Fig. 7 shows the distribution of the success-ratio and user-uptime metrics across the different simulations. We see that both metrics show an availability of around 0.75 but the standard deviation for user-uptime is much smaller indicating that user-uptime more precisely and consistently captures the outage.

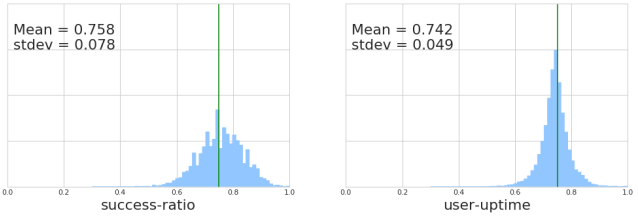

Figure 7: Normalized distribution of success-ratio and user-uptime measurements

For the second example, we incorporated retries: when a request fails, the user retries the request. Fig. 8 shows the distribution for success-ratio and user-uptime for this experiment. We see that user uptime is more accurate than success-ratio: identifying the availability of 0.75 with a tight distribution. Success-ratio, on the other hand, is affected by the retries and indicates a lower availability than what users actually experienced.

In summary, at least with synthetic examples we see that user-uptime better captures availability than success-ratio. Section 6 compares the two metrics using production data.

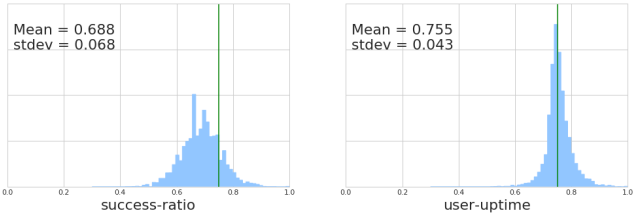

Figure 8: Normalized distribution of success-ratio and user-uptime measurements with automatic retries

5 Actionable availability: windowed user-uptime

Cloud productivity suites commonly define, track, and report monthly or quarterly availability. As Section 3.4 points out, monthly or quarterly availability data is often not actionable. Concretely, monthly availability does not distinguish between a flaky system that routinely fails a small percentage of the requests from a system whose failures are rare but when they happen the system is unavailable for an extended duration.

To distinguish long outages from flakiness, we must look at availability at the timescale of the outages: looking at a timescale of months may average away the unavailability and paints a rosier picture than reality. Our approach, *windowed user-uptime*, addresses this by combining information from all timescales simultaneously. The rest of this section describes windowed user-uptime and explores its properties.

5.1 Calculating windowed user-uptime

Windowed user-uptime iterates over all time windows fully included in the period of interest (e.g., month or quarter) and it computes an availability score for each window size. For example, to calculate windowed user-uptime over the month of January 2019, windowed user-uptime finds the worst period of each window size from 1 minute up to a month. The score for a window size w is the availability of the *worst* window of size w in our period of interest. We calculate the availability for a particular window from t_1 to t_2 as follows:

$$\mathrm{A}(t_1,t_2) = \frac{\text{good service between } t_1 \text{ and } t_2}{\text{total service between } t_1 \text{ and } t_2}$$

To obtain the score for a window of size w, we enumerate all windows of duration w, compute the availability for each of them, and take the lowest value. Thus, we end up with one score for each window size. We call this score the minimal cumulative ratio (MCR).

Formally, the *MCR* for a window size w in a period (T_1, T_2) is as follows:

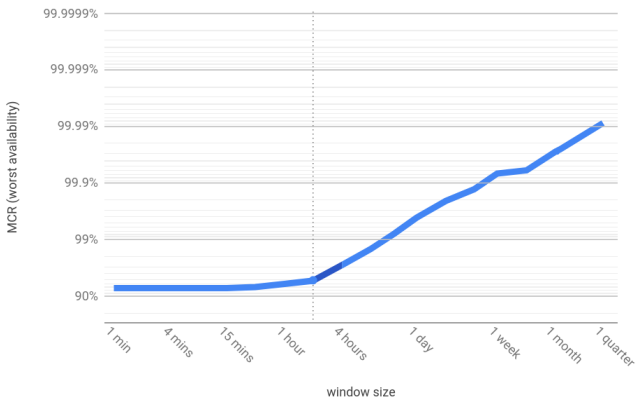

Figure 9: Windowed user-uptime:

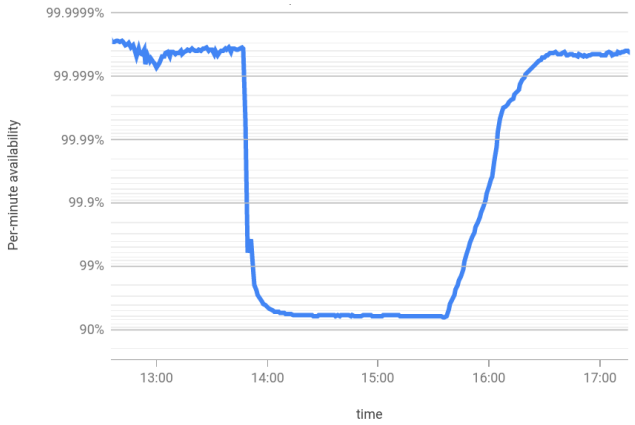

Figure 10: Per-minute availability over time

$$\mathrm{MCR}(w) \equiv \min_{T_1<t_1<t_2<T_2} \{A(t_1,t_2) | t_2 - t_1 = w\} \qquad (4)$$

MCR picks the worst availability for each window size because that is the window that had the most impact on overall availability. Prior work in evaluating real-time garbage collectors and specifically the Minimum-Mutator-Utilization metric [15] inspired our windowing approach.

Fig. 9 shows an example of windowed user-uptime over a period of one quarter; the x-axis gives the window size and the y-axis (in log scale) gives the corresponding MCR. Fig. 9 enables us to readily make the following observations:

- The overall availability for the quarter is 99.991%; this is the rightmost point in the curve.
- There was no window of one minute or longer whose availability was worse than 92%.
- Knees of the curve can give insight into the longest incidents and their distribution for the service. In the example, the knee at about 2 hours indicates that the episode that brought availability down to 92% lasted for two hours; availability rapidly improves for larger windows.

By maintaining a mapping from each window size to the start of the window, we can readily examine the worst window(s) of a particular size in detail (e.g., Fig. 10). We can automatically discover the knee by finding the peak of the 2nd derivative of the windowed graph.

In summary, windowed user-uptime provides us with a rich view into the availability of our services. This data is actionable: it tells us the windows that degrade our overall availability. Our engineers use these graphs to find and fix sources of unavailability.

5.2 Monotonicity with integer-multiple sized windows

Intuitively, we expect windowed user-uptime to be monotonically non-decreasing in the size of the window; i.e., we expect larger windows to have better availability. This section proves that this is the case as long as window sizes are integer multiples of smaller ones.

We use boldface to distinguish windows: $\mathbf{w} = [t_1, t_2]$ from window sizes: $w = t_2 - t_1$. For the remainder of this section a window $\mathbf{w}$ will always be a continuous interval. The availability of a window $\mathbf{w}$ is the ratio of uptime and total time[1] over that window:

$$A(\mathbf{w}) = \frac{u(\mathbf{w})}{t(\mathbf{w})}.$$

Given a period of interest, $[T_1, T_2]$, windowed user-uptime, or the *minimal cumulative ratio* is the least of the availabilities of all windows (Eq. (4)) of *size* w that are fully enclosed within $[T_1, T_2]$:

$$\mathrm{MCR}(w) \equiv \min_{\substack{\mathbf{w} \subseteq [T_1,T_2] \\ |\mathbf{w}|=w}} \left(\frac{u(\mathbf{w})}{t(\mathbf{w})} \right).$$

One expects $\mathrm{MCR}(w)$ to be a monotonic function. Indeed, the availability over a window $\mathbf{w}'$ of size w' is intuitively the mean of a fluctuating time series. Scanning the interval with windows of smaller size $w < w'$, we should find both higher and lower availability values and the minimum of these should be smaller than the mean over the whole window:

$$\mathrm{MCR}(w) \overset{?}{\leq} MCR(w') \qquad w \leq w'. \qquad (5)$$

[1] We use user-uptime terminology but windowing can be defined for other availability metrics. Substitute the corresponding concepts for good and total service, for example request count in case of success-ratio.

Eq. (5) indeed holds when we can cover a larger window fully using windows of the next smaller size without overlap. Consider $\mathbf{w}'$ of size w' covered with windows of size w as in Fig. 11

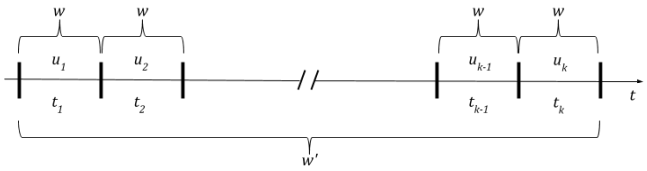

Figure 11: Covering a window with smaller ones: $w' = kw$

We denote the uptime and total time of the i^{th} window as u_i and t_i, respectively. Then the availability of $\mathbf{w}'$ can be written as:

$$A(\mathbf{w}') = \frac{u_1 + u_2 + \ldots + u_k}{t_1 + t_2 + \ldots + t_k}.$$

It is impossible that all of the ratios u_i/t_i are greater than $A(\mathbf{w}')$. Indeed, the above equation can be rearranged to

$$0 = \sum_{i=1}^{k} (u_i - A(\mathbf{w}')t_i)$$

and at least one term in the sum must be non-positive for the sum to yield zero: $u_j \leq A(\mathbf{w}')t_j$. Therefore there is at least one window of size w whose availability is at most that of $\mathbf{w}'$:

$$\frac{u_j}{t_j} \leq A(\mathbf{w}').$$

It follows that every window of length kw contains at least one window of size w with lower availability. Since there exists a window whose availability is $\mathrm{MCR}(kw)$, it follows that:

$$\mathrm{MCR}(w) \leq \mathrm{MCR}(kw) \qquad \forall k \in \mathbb{N} \qquad (6)$$

5.3 Monotonicity in the general case

For practical purposes, Eq. (6) is "enough": the worst availability of a day is always better than the worst availability of an hour or of a minute. Curiously, windowed user-uptime is not monotonically non-decreasing in the general case. The Appendix gives a proof that we can bound the extent of this apparent anomaly:

$$\frac{k}{k+1}\mathrm{MCR}(w) \leq \mathrm{MCR}(w') \qquad kw < w'. \qquad (7)$$

In practice, we can guard against the apparent anomaly and enforce strict monotonicity by restricting to integer multiples, e.g. compute windowed user-uptime for windows that are powers of two.

6 Evaluation

Since last year, we have used windowed user-uptime for all G Suite applications (Calendar, Docs, Drive, Gmail, etc.). This section presents case studies where this metric provided keen insight into availability especially compared to the success-ratio metric.

We present availability data from production Google servers; in doing so we ignore unavailability due to issues with user devices or mobile networks.

Since the actual availability data is Google proprietary, we linearly scale and shift all curves in our graphs. This transformation preserves the shape of the curves and enables meaningful comparison between them.

6.1 Availability due to hyper-active users

As we have discussed, bias (due to hyper-active users) can negatively affect success-ratio. This section shows a real-life example of this phenomenon.

Fig. 12 shows (scaled) user-uptime and success-ratio for a large user base. One observes a consistent and significant mismatch between success-ratio and user-uptime.

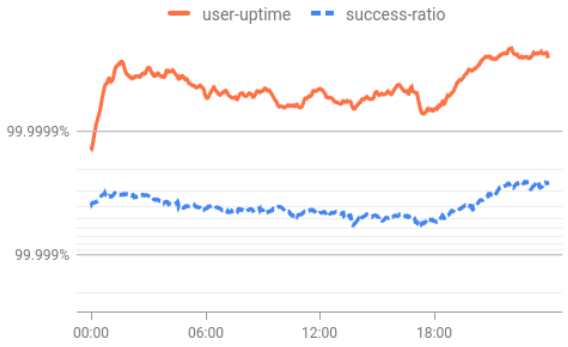

Figure 12: Uptime discrepancy due to mixed use-cases (log scale)

Upon diving into this discrepancy, we discovered that most (99%) active users contribute fewer than 100 events each in the duration of the graph. The remaining (1%) of the users are hyper-active and contribute 62% of all events.

Fig. 13 breaks down the availability data. The "Overall" curves give overall availability, the "Hyperactive users" curves give availability for the most active 1% of the users, and the "Typical users" curves give availability for the rest of them. Success-ratio appears heavily biased towards the availability for hyper-active users even though they make up only 1% of the user base.

Bias due to hyper-active users for success-ratio is, thus, not a theoretical possibility: we see this in production: indeed our most active users are 1000x more active than our

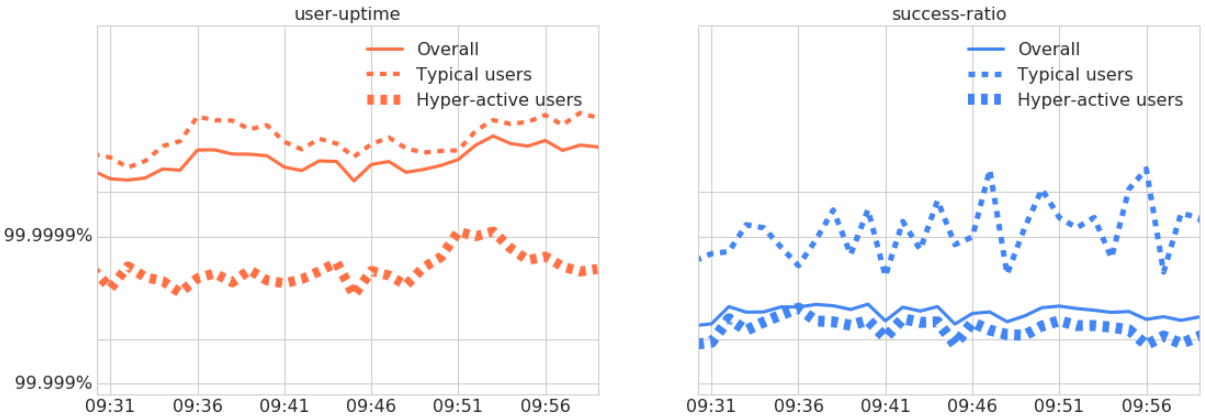

Figure 13: Left: user-uptime for the two clusters of events and the combined user-uptime. Right: success-ratio for the two clusters of events and the combined success-ratio

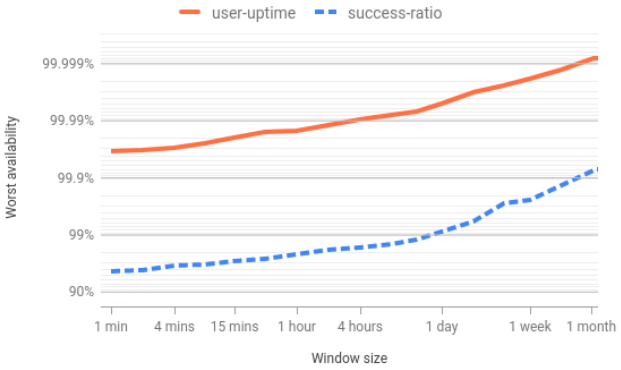

Figure 14: Monthly windowed uptime: success-ratio and user-uptime

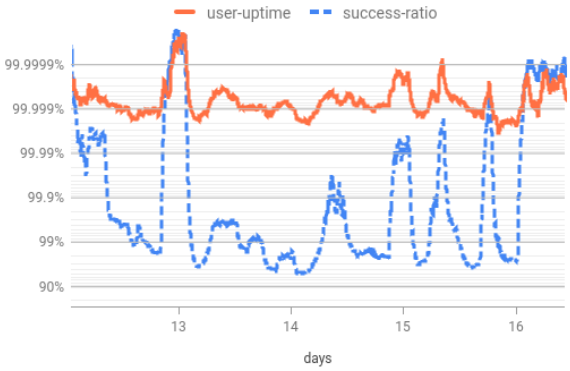

Figure 15: Effect of abusive users

"typical" users which makes this bias a common occurrence. When this bias occurs, success-ratio can mislead us towards thinking that an incident is much more or much less severe than it actually is.

6.2 Availability and hyper-active retries

While so far we have shown availability data for all customers of a G Suite application, we also separately measure availability for large customers since they often have their own unique patterns of usage and thus may experience different availability from other users. From this data we noticed that a particular large customer ($> 100,000$ users) had much poorer success-ratio than other users of Gmail; moreover we noticed a discrepancy between success-ratio and user uptime (Fig. 14). While the curves for user-uptime and success-ratio have the same shape, they are far apart with user-uptime indicating a better availability than success-ratio.

Fig. 15 shows the user-uptime and success-ratio over time. We can see that before and after the incident (i.e., the two ends of the graph) the two metrics were similar, but during the incident the two metrics diverged.

On investigation we uncovered that a small number of users had enabled a third-party application which synced their mailboxes; this external service was unable to handle large mailboxes and for these users it would retry the sync operation repeatedly and without exponential back-off. This generated bursts of failing requests. The event count of these failures drove the success-ratio availability down, even though this impacted only a handful of users and other requests for the users (e.g., to read an email) were successful. User-uptime did not suffer from the bias due to these retries. We were able to resolve this incident by communicating with the third-party vendor.

In summary, users (and the clients they use) can behave differently during incidents than during normal operations. Clients may make many more requests during incidents (our example above) or they may just decide to give up on the

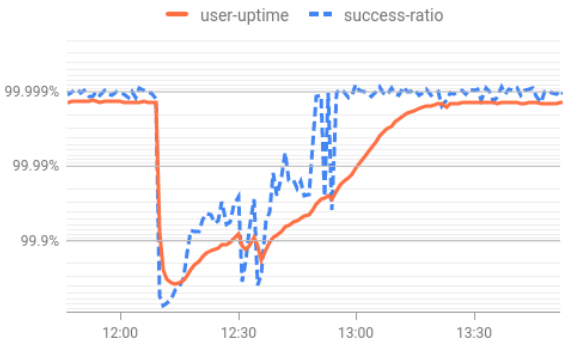

Figure 16: Impact assessment

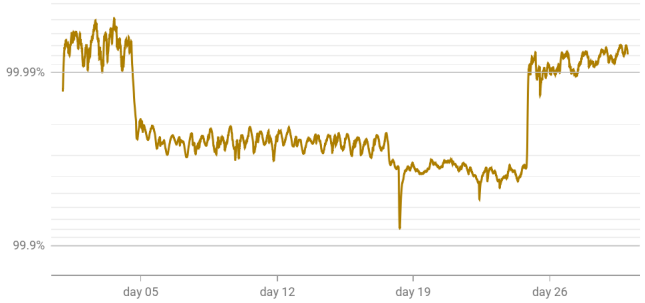

Figure 17: User-uptime analysis of Google Drive traffic, 30 days.

system and try few hours later. In both cases, success-ratio over- or under-estimates the magnitude of an outage while user-uptime matches user perception.

6.3 Quantifying impact of outages

We quantify the impact of every outage on our users. This impact advises the aftermath of the outage: for example, a really severe outage may result in a temporary freeze in deploying new versions until we address all the underlying issues while a less severe outage may require less extreme measures.

When using success-ratio, this quantification is difficult since the metric is not based on time. Concretely, since request patterns (and specifically retry behavior) changes during outages, using success-ratio to derive impact is not meaningful.

Consider, for example Fig. 16 which displays an outage from one of our products. Both success-ratio and user-uptime dip at 12:10 to around 97%, then recover over the course of an hour. What impact did this outage have on our users? Was it the case that $100\% - 97\% = 3\%$ of our users couldn't work for over an hour? Or that the impact was more evenly spread across our users and that automatic retries hid most of the negative impact?

The seconds of downtime, which we compute as part of user uptime, provides more insight. From there we see the minutes of downtime that our users experience. The large number we measured for this indicated that this was clearly a significant outage for some users and the retry behavior was unable to mask this outage from our users.

6.4 Combining user-uptime and success-ratio

The Drive team extensively uses windowed user-uptime to investigate, root-cause and fix sources of unavailability. The proportionality of user-uptime is critical for their use case: they need to be able to make changes and notice a proportional change in the availability metric. Sometimes, combining user-uptime with success-ratio yields valuable insights.

Fig. 17 illustrates this situation. A first analysis of the uptime graph shows a drop on the 4th day followed by a degradation that reached a plateau on day 18 then was finally fixed on day 26. Looking at success-ratio instead of user-uptime paints the same picture, as can be seen in Fig. 18. Looking at either of these two graphs, it is reasonable to assume that this incident has a single cause.

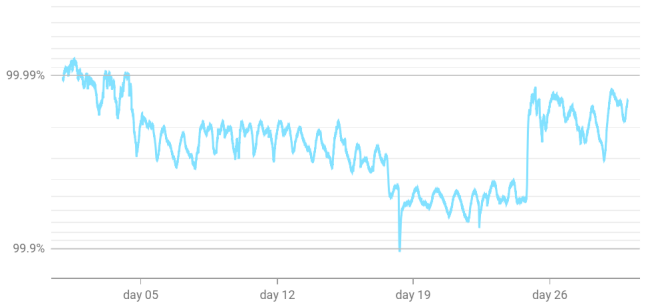

Figure 18: success-ratio of Google Drive, same 30 days as Fig. 17.

But if we plot success-ratio and user-uptime together as in Fig. 19, we see that they diverge on day 18. Indeed, an investigation showed that the period between day 18 and day 26 was caused by a different issue, introduced by an attempted fix for the first problem. It manifests differently in user-uptime compared to success-ratio due to different client behavior for this new issue. This divergence led the engineers to the correct path in fixing the new issue instead of trying to find out why the old issue was still ongoing, and made the path to resolution easier and quicker.

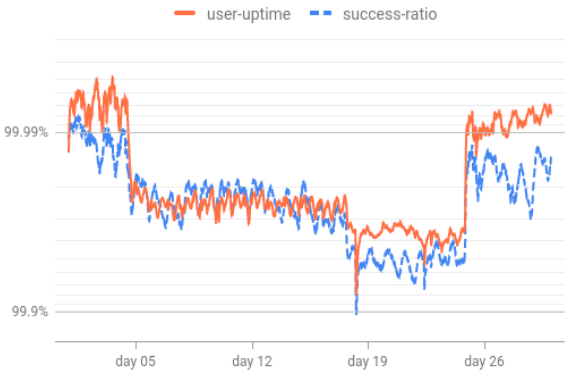

Figure 19: Divergence of user-uptime and success-ratio

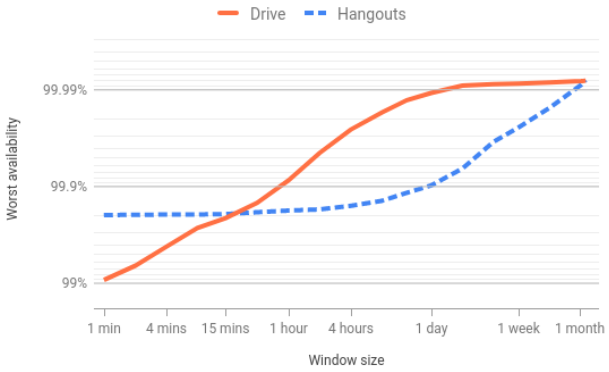

Figure 20: Monthly windowed user-uptime distinguishes between the nature of unavailability

6.5 Windowed uptime indicates burstiness of unavailability

When we look at availability metrics aggregated over a month or a quarter we cannot see short episodes of poor availability. Fig. 20 shows the windowed user-uptime of Drive and Hangouts. The two curves meet at the same point at the 1 month mark; thus they have the same scaled availability (99.972%) over the course of the month. Their windowed user-uptime graphs, however, tell different stories.

Windowed user-uptime reveals that it is a four hour episode (the knee of the curve) that held back Hangouts from having an even higher availability. In a different month without such an episode or if we fix the root cause behind such episodes, this product would have a higher availability.

For Drive, the windowed user-uptime curve has no pronounced knee; this means that rather than a single long episode, there are continuous short episodes which are holding back the service from having a higher availability. Thus, unless we fix their root cause, Drive will continue to suffer from these downtimes month after month.

By exposing shorter episodes of low availability (e.g., Hangouts' four hour episode), windowed user-uptime alerts us to problems that would otherwise be masked by the (commonly-used) monthly aggregation. Our teams consequently use windowed user-uptime to identify, root-cause and then fix the sources of these short episodes, improving overall availability.

7 Applicability of windowed user-uptime

To calculate windowed user-uptime we need fine-grained logs of individual user operation. These logs must include a key that enables us to chain together operations for each user, the timestamp of the operation and the status of each operation (success or failure). In some cases, additional information is invaluable: for example, (i) knowing the type of the operation enables us to determine if different operations have different availability and (ii) knowing the organization of a user enables us to determine if a particular organization is experiencing worse availability compared to other organizations.

In the simplest case we need to retain only the cumulative count of up and down minutes for each minute to calculate windowed user-uptime over any time duration. If we want to slice data along additional dimensions we must maintain the count of up and down minutes for each dimension (organization, operation type, etc.).

It took us about 1 year to deploy windowed user-uptime to all of the G Suite applications. This time included implementation (e.g., to normalize the different log formats so we can use the same pipeline across all of our applications and to build the pipeline for calculating the metric) but the bulk of the time was in determining which operations we should consider in windowed user-uptime and whether or not a given operation's availability could be visible to users. From this experience we are confident that windowed user-uptime is broadly applicable: any cloud service provider should be able to implement it.

8 Conclusion

We have introduced a novel availability metric, *user-uptime* which combines the advantages of per-user aggregation with those of using a time-based availability measure. We have shown that as a result user-uptime avoids multiple kinds of bias: hyper-active users contribute similarly to the metric as regular users, and even behavioral changes during an outage result in a proportional and meaningful measurement that in many cases is even more precise than success-ratio.

We have evaluated our metric against the commonly-used success-ratio metric using production data from G Suite services. We show that the bias in success-ratio is not an aca-

demic argument: we actually encounter it in production and that user-uptime avoids this bias.

We have introduced a visualization technique, windowed availability, that allows us to study multiple time-scales from single minutes to a full quarter in an easy to understand graph. Again using production data from G Suite services, we show that windowed user-uptime sheds invaluable and actionable insight into the availability of our services. Specifically, windowed user-uptime enables us to differentiate between many short and fewer but longer outages. This ability focuses our engineering efforts into improvements that will yield the most gains in user-perceived availability.

Acknowledgments

We are grateful for insighful contributions from André Prinsloo, Jesse Bickmore, Aaron Isotton, Andrés Martínez and Tony Scelfo, all of Google, and thoughtful comments from the anonymous reviewers.

References

[1] 5-minute outage costs google $545,000 in revenue. https://venturebeat.com/2013/08/16/3-minute-outage-costs-google-545000-in-revenue/.

[2] Amazon s3 service level agreement. https://aws.amazon.com/s3/sla/.

[3] Amazon.com goes down, loses $66,240 per minute. https://www.forbes.com/sites/kellyclay/2013/08/19/amazon-com-goes-down-loses-66240-per-minute/.

[4] Cloud services you can trust: Office 365 availability. https://www.microsoft.com/en-us/microsoft-365/blog/2013/08/08/cloud-services-you-can-trust-office-365-availability/.

[5] G suite service level agreement. https://gsuite.google.com/intl/en/terms/sla.html.

[6] Google's bad week: Youtube loses millions as advertising row reaches us. https://www.theguardian.com/technology/2017/mar/25/google-youtube-advertising-extremist-content-att-verizon.

[7] Slack system status: How is uptime calculated? https://status.slack.com/.

[8] Dan Ardelean, Amer Diwan, and Chandra Erdman. Performance analysis of cloud applications. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 405–417, Renton, WA, 2018. USENIX Association.

[9] Salman Abdul Baset. Cloud slas: present and future. *Operating Systems Review*, 46, 2012.

[10] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. " O'Reilly Media, Inc.", 2016.

[11] Betsy Beyer, Niall Richard Murphy, David K. Rensin Rensin, Stephen Thorne, and Kent Kawahara. *The Site Reliability Workbook: Practical Ways to Implement SRE*. " O'Reilly Media, Inc.", 2018.

[12] John Brevik, Daniel Nurmi, and Rich Wolski. Quantifying machine availability in networked and desktop grid systems. Technical report, CS2003-37, University of California at Santa Barbara, 2003.

[13] Aaron B. Brown and David A. Patterson. Towards availability benchmarks: A case study of software RAID systems. In *Proceedings of the General Track: 2000 USENIX Annual Technical Conference*, 2000.

[14] David Chappell. Introducing the windows azure platform. *David Chappell & Associates White Paper*, 2010.

[15] Perry Cheng and Guy E Blelloch. A parallel, real-time garbage collector. *ACM SIGPLAN Notices*, 36(5):125–136, 2001.

[16] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[17] Matteo Dell'Amico, Maurizio Filippone, Pietro Michiardi, and Yves Roudier. On user availability prediction and network applications. *CoRR*, abs/1404.7688, 2014.

[18] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Back to the future: On predicting user uptime. *CoRR*, abs/1010.0626, 2010.

[19] Patricia Takako Endo, Moisés Rodrigues, Glauco Estacio Gonçalves, Judith Kelner, Djamel Fawzi Hadj Sadok, and Calin Curescu. High availability in clouds: systematic review and research challenges. *J. Cloud Computing*, 5, 2016.

[20] A. L. Goel and J. Soenjoto. Models for hardware-software system operational-performance evaluation. *IEEE Transactions on Reliability*, R-30(3), 1981.

[21] A. Goyal and S. S. Lavenberg. Modeling and analysis of computer system availability. *IBM Journal of Research and Development*, 31(6), 1987.

[22] Jim Gray and Daniel P. Siewiorek. High-availability computer systems. *IEEE Computer*, 24(9), 1991.

[23] Rachid Guerraoui and André Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4), 1997.

[24] ISO Iso. iec/ieee international standard-systems and software engineering–vocabulary. Technical report, ISO/IEC/IEEE 24765: 2017 (E), 2017.

[25] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.

[26] Lingshuang Shao, Junfeng Zhao, Tao Xie, Lu Zhang, Bing Xie, and Hong Mei. User-perceived service availability: A metric and an estimation approach. In *2009 IEEE International Conference on Web Services*, 2009.

[27] Ushio Sumita and Yasushi Masuda. Analysis of software availability/reliability under the influence of hardware failures. *IEEE Trans. Software Eng.*, 12(1), 1986.

[28] Maria Toeroe and Francis Tam. *Service availability: principles and practice*. John Wiley & Sons, 2012.

[29] Ben Treynor, Mike Dahlin, Vivek Rau, and Betsy Beyer. The calculus of service availability. *Communications of the ACM*, 60(9), 2017.

[30] S. R. Welke, B. W. Johnson, and J. H. Aylor. Reliability modeling of hardware/software systems. *IEEE Transactions on Reliability*, 44(3), Sep. 1995.

Appendix: Monotonicity

We prove Eq. (7) which was stated in Section 5 without proof.

$$\frac{k}{k+1}\text{MCR}(w) \le \text{MCR}(w') \qquad kw < w'. \tag{7}$$

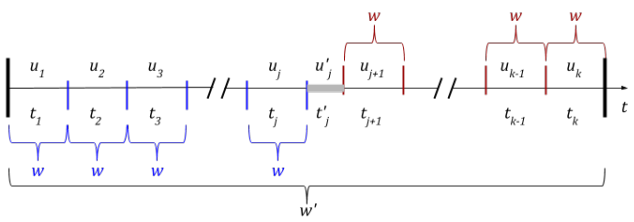

Figure 21: Embedding length-w windows in length-w' one. for $w' \neq kw$

To prove Eq. (7), we cover the larger window of size w' with windows of size w. Pick some $0 \le j \le k$ and fill the segment with j windows without a gap from the left and $k-j$ windows from the right, as in Fig. 21. As w' is not an integer multiple of w, this will leave a small segment uncovered in the middle. Like before, we write the availability of $\mathbf{w}'$ in terms of the uptime and total time of the segments as:

$$A(\mathbf{w}') = \frac{U}{T} = \frac{u_1 + \ldots + u_j + u'_j + u_{j+1} + \ldots + u_k}{t_1 + \ldots + t_j + t'_j + t_{j+1} + \ldots + t_k}.$$

The availability of each window of length w is bounded by $\text{MCR}(w)$:

$$\frac{u_i}{t_i} \ge \text{MCR}(w).$$

Substituting $u_i \ge \text{MCR}(w)t_i$ and $u'_j \ge 0$ yields:

$$A(\mathbf{w}') \ge \text{MCR}(w)\frac{t_1 + \ldots + t_j + t_{j+1} + \ldots + t_k}{t_1 + \ldots + t_j + t'_j + t_{j+1} + \ldots + t_k}$$

or

$$TA(\mathbf{w}') \ge \text{MCR}(w)(T - t'_j).$$

Depending on how many windows we add on the left or on the right, j can be any integer between 0 (all windows on the right) and k (all windows on the left). Each of these possibilities yields the same inequality as above, they differ only in the respective value of t'_j. Let's add all of these $k+1$ inequalities:

$$(k+1)TA(\mathbf{w}') \ge \text{MCR}(w)(kT + T - \sum_{j=0}^{k} t'_j).$$

Note, however, that

$$T - \sum_{j=0}^{k} t'_j \ge 0$$

because t'_j are total times of non-overlapping intervals, all part of T. Substituting this yields the promised bound:

$$A(\mathbf{w}') \ge \frac{k}{k+1}\text{MCR}(w).$$

We see, therefore, that every window of length $w' > kw$ contains at least one window of size w with lower availability. Since there exists a window whose availability is $\text{MCR}(w')$, Eq. (7) follows.