Empirical Characterization of User Reports about Cloud Failures

Sacheendra Talluri[1], Leon Overweel[2], Laurens Versluis[1], Animesh Trivedi[1], Alexandru Iosup[1]
[1]*Vrije Universiteit Amsterdam*, The Netherlands [2]*Dexter Energy*, The Netherlands Corresponding author: s.talluri@vu.nl

***Abstract*—Cloud services are important for healthcare, banking, communication, and other purposes. Inevitably, such services fail, harming the processes and disturbing the people that depend on them. Understanding failure in cloud services is challenging, but important to help preventing them. Much work has studied failure logs and reports provided by infrastructure operators. However, there is a paucity of information about how *users* perceive the failures of cloud services. In this work, we collect user-reported failures and characterize them empirically. We collect failures reported by users to the trusted aggregator Outage Report for 12 cloud services over 16 months spread across 2019 and 2020. We show evidence that user-reported failures not only capture major failures also self-reported by cloud operators, but also provide information about additional failures. We count and analyze time patterns in these reports. We make 6 main observations about how users perceive failure in cloud services. We find over 10x differences in request failure rates across microservice structures when using user reported traces compared to using a constant failure distribution. Overall, our study provides the first long-term characterization of user-reported cloud failures.**

***Index Terms*—failure, availability, cloud service, cloud, characterization, crowdsourcing**

I. Introduction

Cloud computing is one of the main computing paradigms for many business and other societal applications. We expect cloud services to embody the results of decades of research on building dependable distributed systems [1]–[3]. However, as the scale and diversity of cloud services increase, we expect the presence of failures in these services to also increase [4]. Thus, it is important to study the presence of failures in cloud services. This has implications for how client and server applications are designed, the guaranties users of cloud services can expect, and the direction of fault-tolerance research. Previous work focuses on failures self-reported or logged by cloud operators [5]–[7]. In contrast, in this work we conduct *the first long-term study on cloud failures as reported by users.*

Information about cloud-service failures is already considered important. Amazon, Microsoft, Google, Facebook, and others offer status updates for many of their cloud services. However, preliminary evidence from common users indicates cloud operators do not report all failures [8] and also choose to report some failures only when media takes notice [9].

Cloud operators have to balance two sets of incentives. On the one hand, cloud operators want to report all failures that reach the news, so they are perceived as competent and transparent. On the other hand, cloud service providers may not want to display many, possibly hundreds, of failures on their status page, as someone who is not an expert can misinterpret this to mean that the provider is unreliable. In practice, failure reporting on the status page is a manual process, and the employees who approve making a failure report public may be disincentivized to report failures and even get negative performance reviews [8].

In this study, rather than self-reports by cloud providers, we consider failure reports from the new vantage point of *user reports*. When a cloud service does not work, its users search the Internet to understand if this is a problem only for them or if it affects other people. The first search results in Google, Bing, and other major search engines, tend to be websites of large aggregators of user-reported failures, e.g., Outage Report, Down Detector, and Down Right Now. Some users further report the problem they observed to the same aggregators; this study focuses on these reports.

The crowdsourced aggregators of cloud-service failures provide an important vantage point, *a valuable source of data which complements what is reported by other sources*. For example, our dataset indicates a failure on 7th May 2019 with 952 user reports. The actual number of users affected is likely much higher, because the users who report are more than an order of magnitude fewer than the total number of visitors to the crowdsourcing website. There is no indication of this failure on the Facebook status page and general Internet search does not reveal more.

Collecting and characterizing user-reported failures in cloud services is challenging. First, the data collection is hampered by the *lack of persistence in data published by crowdsourced aggregators*; most of the data remains available for less than a day. Even if such data becomes available, to evaluate its quality we further need to compare the crowdsourced data to trusted sources. Second, *user reports do not necessarily amount to cloud-service failures*, and new methods are needed to derive failures from (sets of) user reports. Third, as a data-driven process, *the characteristics of user-reported failures are currently not known*. Finally, *we lack open-access data about user reports of cloud failures over long periods of time* which inhibits the study of failures from this new perspective and the use of user-reported failures for experiments.

In this work, we broaden our understanding of cloud failures by taking a unique, user-centric approach. We tackle the aforementioned challenges with a four-fold contribution:

1) We propose a novel method to understand how users perceive failures in client-facing cloud services (Section III). Our method leverages *crowdsourced* data, long-term, to

enable a longitudinal study. To assess the quality of user reports, we compare them to failure reports from cloud service status pages.

2) We analyze *when and how* user-reports occur (Sections IV). Our analysis, which proposes the first characterization of such failures based on crowdsourced data, leads to important main observations. Among the types of characterization, this study covers failure counts, symptoms, longitudinal trends.
3) We evaluate in simulation the impact of failures on latency (Section V). We use user-reported failure traces and compare with the constant failure rate used by prior work. We find that using a constant failure rate overestimates the number of retries by over 10 times.
4) We provide a unique dataset of crowdsourced failure data for 2019 and 2020, and the associated open-source software framework to study such data. It covers 12 popular SaaS services (e.g., Facebook, YouTube, and Skype), with data sampled with 20-minute granularity. The dataset and the framework are available online: https://github.com/atlarge-research/outage_report_characterization.

II. Operation and Failure of Cloud Services

Cloud services are the digital services running largely in datacenters. They cover a broad span of functionality, from user-fronting services that provide meaningful (e.g., business) logic, to deployment and orchestration of datacenter applications, to continuous monitoring and feedback, to back-end services that complete low-priority tasks.

In this work, *we focus on Outage Report, which is unique among crowdsourced failure aggregators* by simultaneously being popular, exposing detailed information about individual reports, and having a favorable scraping policy. We collect from it user reports for 12 services, chosen for the diverse interests and use cases they represent. Apple services—developer services provided by Apple, such as certificate validation—and Github are primarily software developer oriented services; they are used by a small niche of users and usually for serious purposes, software development-related. Skype and Gmail are typically used for mostly non-entertainment communication. Facebook Messenger and Whatsapp are mainly used for casual social communication. A unique feature of Facebook Messenger and Whatsapp is that their primary purpose is narrow and well defined—to send and receive messages. Snapchat, Facebook, Twitter, and Instagram are popular social media services. Users typically interact with social media services multiple times a day, each time for short highly engaging bursts. YouTube and Netflix are multimedia entertainment services. Interaction with these services depends on the length of the content.

A. Terms and Definitions

We define a *failure* as a period of time during which the service is not operating as expected. Failures can be of different types, such as unavailability of service, crash of client application, message transmission problems, etc. We analyze actual failures reported by users in Section IV.

TABLE I
Datasets and subsets summarized.

Use	Dataset/subset	# reports	Date range
§IV	Reports	783,253	14 Apr. 2019 - 31 Oct. 2020
§IV	Detailed reports	174,170	14 Apr. 2019 - 31 Oct. 2020
§III-B	Github comparison	2,974	14 Apr. 2019 - 31 Aug. 2020

A failure *report* is an explicit indication by the user to the monitoring service about a cloud service failure. A *detailed report* can include additional information such as the symptom of failure the user experienced, the location of the user, and time of reporting.

A failure *symptom* is the application behavior the user observed due to the failure of the cloud service. Symptoms include website not loading properly, mobile application crashing, messages not being sent, etc.

III. Method for Collecting User Reports

Understanding user reports begins with collecting them. In this section, we describe the process of collecting, preparing, and validating user reports about cloud service failures.

A. Data Collection and Extraction

In this work, we consider reports about multiple cloud services, collected (scraped) from the crowdsourced failure reporting website Outage Report. The scraped pages contain the number of reports for the last 24 hours, aggregated in 20-minute intervals (*quanta*), and detailed information about reports made in the last 20 minutes. To avoid stressing the website, we scrape it only once every hour, so the scrapes miss detailed reports added with between 20 minutes and an hour before our scraping time. However, the sampled dataset already includes a large number of samples (174,170 detailed reports). Our scraper is hosted in Ohio, US, by Amazon AWS.

Outage Report embeds the data we collect in the webpage, as JSON objects. We use regular expressions to extract the data, constructed manually after careful inspection of these webpages. We further compare the data we parse to the visualizations made by Outage Report, to verify if we parsed the data correctly.

We collect the data over a long period of time, from April 2019 to October 2020; for some services, such as Github, the period is only May 2019 to August 2020. This allows us to study long-term trends. The results of parsing this data are two datasets, Reports and Detailed reports, stored in tabular form using the Parquet file format and summarized in Table I.

B. Comparison between User- and Self-Reported Failures

To assess the usefulness of user reports in identifying failures, ideally, we would compare them to the ground truth of failures for all services we monitor. However, this is not possible, as the operators rarely report publicly all the failures experienced by their cloud services. Instead, we take as ground truth the set of official updates about failures currently reported on each cloud service's status page. We have such information from Github's status page. This allows us show evidence whether the failures reported by users are similar in nature to the failures Github self-reports. Furthermore, this gives insight

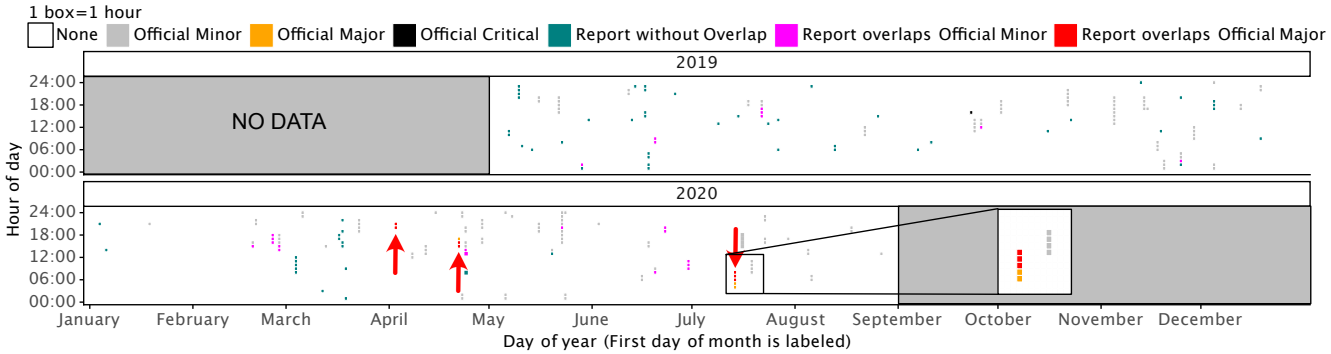

Fig. 1. Failures reported by Github on their official status page compared to failure reported by users to Outage Report. The 3 official major failures that overlap with user reported failures are indicated with red arrows.

into whether the metrics cloud services are tracking are what users experience as failures or do users experience something different. Our main findings are:

O-1: There is overlap between the *major* failures on the Github status page and those reported by users.
O-2: There is little overlap between *all* the failures self-reported by Github and crowd-reported by users.

Figure 1 depicts Github failures as (officially) self-reported by Github and as reported by users to Outage Report. The horizontal axis represents the days of the year. The first day of each month is labeled. The vertical axis represents the time of the day. A single box represents one hour of one day. There are 3 major and 1 critical issues officially reported by Github. All 3 major issues overlap user reports to Outage Report (**O-1**). This is seen in the two red-colored (bright) events in April 2020 and one in July 2020. The critical event does not have a match. Github describes the critical event as a problem with notifications. We conjecture that the short duration of the problem (20 minutes) and the subject of effect (notifications) led to our dataset not recording reports about it. We further find 13 official minor events that overlap with user reports. This gives evidence the overlap cannot occur randomly, so *user-reported failures are consistent with the ground truth*.

We also observe that many official minor reports have *no* overlap (**O-2**). This suggests little overlap between what users and GitHub perceive as important enough to report. The difference between the sets of official and of user reports stems from the different kind of failures they track. Minor events reported by Github range from degraded performance to problems with forking and accessing the website. In contrast, the most popular reason for events reported to Outage Report by far is "Website Down", which accounts for about two-thirds of the reports for Github. In line with similar conjectures about software-developing companies [10], we conjecture the official reports correspond to failures tracked by Github's *internal* metrics, whereas user-reports correspond to events with high visibility for the *(external) users*.

The difference between officially reported and user-reported failures suggests there is a gap in the definition of failures that are to be reported. On the one hand, as researchers and expert users, we want detailed information when a service we use is not functioning. On the other hand, a service cannot publish thousands of small events as failures, because this could fatigue the users and be misinterpreted as pervasive unreliability.

TABLE II
NUMBER OF REPORTS AND DETAILED REPORT, PER CLOUD SERVICE.

Rank	Cloud service	Report count	In detail	In detail, %
1	Apple	3,208	2,566	79.99%
2	Github	4,432	1,062	23.96%
3	Skype	4,799	2,029	42.28%
4	FB Messenger	13,788	6,836	49.58%
5	Gmail	16,471	9,427	57.23%
6	Whatsapp	34,653	9,426	27.20%
7	Snapchat	68,863	8,118	11.79%
8	Netflix	74,631	20,211	27.08%
9	Facebook	79,880	25,886	32.41%
10	Twitter	136,672	20,252	14.82%
11	YouTube	152,169	26,783	17.60%
12	Instagram	193,687	41,574	21.46%
	Total	783,253	174,170	22.24%

Implications: Our findings in this section motivate the need to revisit current failure-reporting methodologies, to make them identify and classify failures consistently with what the users experience. We recommend reporting both global failures and fine-grained, detailed failures per operational instance (shard) of each service; for example, the Salesforce status-page [11] offers already such reporting.

IV. Analysis of the User Reports

The number of failure reports and the symptoms behind reveal information about the nature of users reporting failures and the nature of the services themselves. In this section we analyze the counts, symptoms, and the pattern in time of user reports. Our main findings are:

O-3: Services with narrow functionality, such as Whatsapp, receive fewer reports compared to services with diverse functionality.
O-4: "Non-functional mobile application" and "Website not working" are the major symptoms that users report.

O-5: Users report more failures on specific days of the week and hours of the day. Most failures were reported during the evening.

A. Count of User Reports

We collected a total of 783,253 user reports. The number of reports for different services in the dataset is summarized in Table II. The 'Report count' column contains the total number of reports we have for a particular cloud service. The 'In detail' column contains the number of reports with additional information such as the location of the reporting user and the reason for the report.

The total number of reports range from 3,208 for Apple to 193,687 for Instagram. The large number of failure reports indicates that manual analysis of each report by engineers or support staff is not possible. Using individual user reports to help with failure identification and resolution requires automated systems able to deal with thousands of reports.

B. Number of User Reports

Different services show different patterns of failure reporting based on their use case. Table II indicates that a service like Instagram which has 500 million daily active users [12], also has a lot of reports. In contrast, relatively niche services like Github, which serve the software development community, have much lower number of reports. Surprisingly, Whatsapp and Facebook Messenger do not receive a lot of reports despite their popularity. Whatsapp, for example, also has over 500 million daily active users [13] but only received 34,653 reports, compared to Instagram's 193,687 (6x less). We conjecture that there are a couple of reasons for this. Whatsapp and Facebook Messenger both have a narrow purpose and well-defined feature set, to send and receive messages (**O-3**). Thus, in this sense there is less of a chance for things to go wrong. Another reason could be that most of the reports are from the USA, where Whatsapp and Messenger might not be the most popular applications.

Out of a total 783,253 reports, 174,170 (22%) reports have additional detail about the symptom that lead to the report and the location. For some services, such as Apple and Gmail, we observe a high fraction of reports (79.99% and 57.23% respectively) with detail. For others, such as Snapchat and Twitter, we observe a low fraction of reports (11.79% and 14.82% respectively) with detail. The number of detailed reports could reflect the temperament of users who use those services, and the intention with which they use the services. For example, we conjecture that frequent Gmail users use it with a deliberate mindset and might be more willing to report with additional information about the symptoms they observe. In contrast, we conjecture that while using Snapchat and Twitter, both of which involve short form content and photos, a user is usually looking for instant gratification. The user might not have the patience and motivation to detail the symptoms. The same user could have different mindsets while using Gmail and Twitter. An exception to this pattern is Github: users typically do use Github seriously, but are somehow reluctant to provide additional detail. A possible cause is that Git itself is a decentralized system, so not being able to open the website is not a problem.

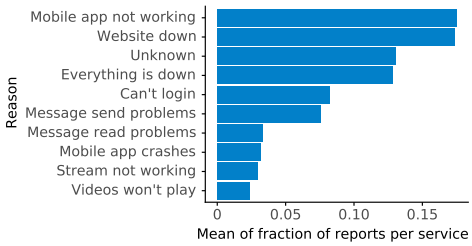

Fig. 2. Mean fraction of reports of a service, per symptom, across all services.

C. Pattern in Time Over a Week

We investigate the distribution of user reports over hours of the week. We intend to find if users are more likely to report failures during certain parts of a day and week than others. Figure 3 depicts the normalized number of reports per hour of week. The reports per hour are normalized by dividing them with the maximum number of reports per hour for that service. The maximum number of reports per hour for a particular service is displayed in the panel above each plot. The horizontal of each plot represents the hour of the week, with the first hour of each marked with the name of the day. The vertical axis represents the normalized reports per hour. Each point on the plot represents the normalized number of reports for a certain service in a certain hour of the week.

We observe a few well-defined peaks for all services (**O-5**). Some services such as Netflix exhibit a diurnal pattern with peaks at night in UTC. That would be evening in most of the USA where most of the reports are from. Evening is when people are likely to use Netflix. So, the failure reports correspond to the typical usage pattern of the service. Instagram also exhibits a slight diurnal trend from Tuesday to Thursday. Apple exhibits a slight diurnal pattern. It is interesting to note that peaks dominate the plots. **Implications:** The peaks either imply that failures occur often on the same day at the same time. They could also imply that a few failures receive such a high number of reports that they dwarf all others.

V. Using Traces of User Reports to Estimate the Number of Retries in Service-based Applications

In this section, we demonstrate how user-reported failure traces can be used in computer systems experiments, and how they impact the results. Our main finding is:

O-6: A real trace leads to more than $10\times$ less retries, but $100\times$ more failures compared to a constant failure probability for applications composed of a long chain of services.

Complex cloud applications such as e-commerce, social media and banking are important to daily life. Latency is an important metric for such applications. For example, increased

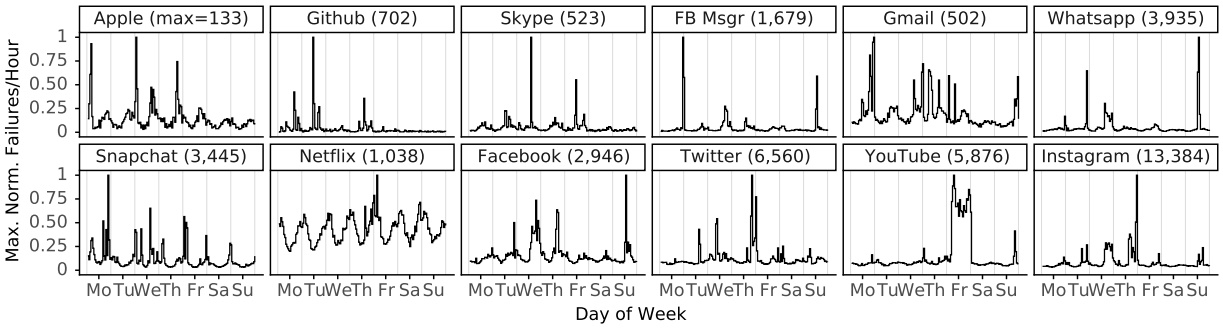

Fig. 3. Distribution of failure reports by hour of the week. Middle of each day is marked. Number of failures per hour is normalized by the maximum number of failures per hour of that service. Maximum number of failures per hour are displayed in the plot labels.

latency reduces the click-through rate of web search [14]. Complex cloud applications are composed of tens, or even hundreds of (micro-)services, working together to perform a task. The latency of the application depends on the latency of each of service. It also depends on how often the services fail. For example, additional latency appears when a service failure leads to a request being retried. Existing work [15], [16] considers *constant failure probability* or similar simple models to estimate the additional latency. We show that using user-reported failure traces produces different results. Specifically, most requests succeed with fewer retries than if a constant failure probability was assumed, but during failure periods, a lot more requests fails.

We conduct experiments with multi-service apps, using trace-based simulation. We interpreting the number of user reports of a service as the probability that a request to the service fails. We scale the number of reports by the number of users of the service at that time. As we do not know the actual number of users, we assume that the number of users follows a diurnal pattern with peak usage in the evening; we approximate this with a shifted sine wave. After scaling the trace, we normalize it into probabilities [0, 1] by dividing each data point by the maximum number of user reports in a single time period (20 minutes in our case). We thus obtain a failure probability associated with every 20-minute period in the trace.

We choose three app structures—for each, a number of microservices and their interdependencies. Many microservice applications are composed of simple structures [17] such as (1) monolith (a single microservice called by a client), (2) fanout (a client makes multiple requests to a microservice all of which need to succeed), and (3) long chain (a client calls a microservice, which calls another, and so in a long chain). We simulate failures in each of these structures and measure the impact on latency.

We use the metric number of retries as an indicator of the tail latency experienced by the service. Tail-latency is particularly important in cloud operations [18], where services are possibly used by millions of users daily. We assume each microservice and client comes with a retry oracle which knows when to retry. Hedging [15] and failure detection algorithms [19] have been studied extensively, and our results complement that work. We also assume that cost of retry dominates the latency, compared to the actual request-processing

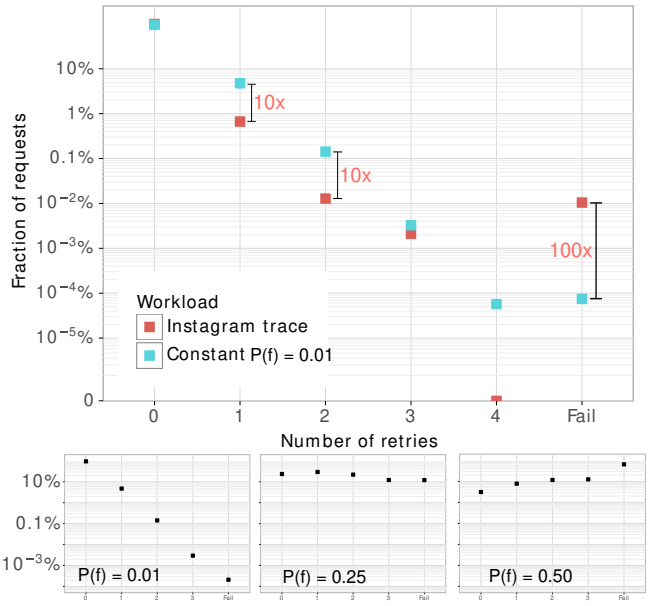

Fig. 4. Distribution of retries for a long chain chain microservice structure with 5 microservices. At the bottom are the distributions at different failure probabilities indicating why the distribution for the trace looks like it does. In particular, notice the large number of failed requests at high failure probability and for the trace. The vertical axis is logarithmic.

time; this holds for short-running apps [20].

We evaluate the number of retries experienced by a long chain workload structure comprised of 5 microservices. The evaluation for other structures is in the technical report. At each stage, we allow a maximum of 3 retries, which is common in practise. Reaching the maximum number of retries at a stage means that the previous stage has to retry again. When the retries of the first stage are exhausted, the request is considered to have *failed*. Figure 4 depicts the results. The horizontal axis represent the number of retries with a special marker for failed requests. The vertical axis is logarithmic axis depicting the fraction of requests. A point on the plot represents the fraction of requests that required a certain amount of retries to succeed. We run the experiment with two different workloads: (new) the Instagram use-reported failure trace and (traditional) a constant failure trace. The constant failure probability we chose is 1% (P(f) = 0.01). The arrival pattern is diurnal, peaking in the evening; we approximate this with a shifted sine wave.

We find that the number of requests that succeed is, in total across all possible retry-counts and also individually for retry-counts 1 and 2, more than $10\times$ lower for the Instagram trace compared to the constant failure probability (**O-6**). Requests which require 4 retries are non-existent for the Instagram trace, but they appear prominently with constant failure probability. Significant differences appear also in failed requests; $100\times$ more requests fail with the Instagram trace compared to the constant failure probability. The number of requests which succeed with 3 retries is very close for both the Instagram trace and constant failure probability. To understand the unique shape of the failure distribution exhibited when using the Instagram trace, we plot the retry distribution with different failure probabilities found in the trace at the bottom of Figure 4. Notice the flat line at 25% failure probability, which is the likely cause of the high number of requests that succeed with 3 retries for the Instagram trace. Similarly, at 50% failure probability, the number of failed requests becomes very high. Such periods with high failure probability lead the high number of failed requests for the Instagram trace.

VI. Related Work

Closest to our work, Gunawi et al. [21] analyze cloud failures over a multi-year period, using as input data news-reports. Complementary to our time-related analysis, they find that the distribution of failure duration in a bad year skews higher than in an average year. Other work focuses on hardware and software failure in clusters [5], HPC [6], and Enterprise [22].

VII. Conclusion

Cloud computing has become a backbone of our modern society, as the main computing infrastructure for many critical services. With increased use of cloud services in the past decade, it is important to understand their failures.

In this work, we proposed to focus on how clients perceive the failures of cloud services, and conducted a systematic study of failures reported by users over a period of 16 months. Our study collects, uses, and shares failure data from a unique data source, the crowdsourced failure aggregator Outage Report. In our study, we have: (i) identified challenges associated with failure data gathering and analysis, and addressed them through a method focusing on user-reported data; (ii) characterized patterns in how users perceive when and how cloud-services fail; (iii) demonstrated the impact of using failure traces on latency; and (iv) made available a unique long-term failure dataset and associated analysis-code. We have summarized our findings in 9 main observations.

The software and data artifacts are available online: https://github.com/atlarge-research/outage_report_characterization [1]

References

[1] Gärtner, "Fundamentals of fault-tolerant distributed computing in asynchronous environments," *ACM CSUR*, vol. 31, no. 1, 1999.

[2] Cachin *et al.*, *Introduction to Reliable and Secure Distributed Programming (2. ed.)* Springer, 2011.

[3] Dean, "Building software systems at Google and lessons learned," Distinguished Lecture Series (DLS), Stanford, accessed: 2020-04-15, 2010.

[4] Gunawi *et al.*, "What bugs live in the cloud? A study of 3000+ issues in cloud systems," in *SOCC*, 2014.

[5] El-Sayed *et al.*, "Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations," in *ICDCS*, 2017.

[6] Javadi *et al.*, "The failure trace archive: Enabling the comparison of failure measurements and models of distributed systems," *JPDC*, vol. 73, no. 8, 2013.

[7] Birke *et al.*, "Failure analysis of virtual and physical machines: Patterns, causes and characteristics," in *DSN*, 2014.

[8] Users of Hacker News, "Hacker news - users discussing employee incentives to report failure - top comment about a recent aws failure," https://news.ycombinator.com/item?id=25213817.

[9] T. Anderson, "Microsoft Office 365, Azure portals offline for many users in Europe," https://www.theregister.com/2015/12/03/office_365_goes_offline/, Accessed: 2020-04-15, December 3, 2015.

[10] Bouwers *et al.*, "Getting what you measure," *Commun. ACM*, vol. 55, no. 7, 2012.

[11] Salesforce, "Salesforce services status," https://status.salesforce.com/products/Salesforce_Services.

[12] O. Agency, "Instagram by the numbers: Stats, demographics & fun facts," https://www.omnicoreagency.com/instagram-statistics/.

[13] Statista, "Number of daily active WhatsApp status users, 2017–2019," https://www.statista.com/statistics/730306/whatsapp-status-dau/.

[14] Arapakis *et al.*, "Impact of response latency on user behavior in web search," in *SIGIR*, 2014.

[15] Primorac *et al.*, "When to hedge in interactive services," in *NSDI*, 2021.

[16] Vulimiri *et al.*, "Low latency via redundancy," in *CoNEXT*, 2013.

[17] Eismann *et al.*, "Serverless applications: Why, when, and how?" *IEEE Softw.*, vol. 38, no. 1, pp. 32–39, 2021.

[18] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013.

[19] Freiling *et al.*, "The failure detector abstraction," *ACM Comput. Surv.*, vol. 43, no. 2, 9:1–9:40, 2011.

[20] Shahrad *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *USENIX ATC*, 2020.

[21] Gunawi *et al.*, "Why does the cloud stop computing? lessons from hundreds of service outages," in *SOCC*, 2016.

[22] Yin *et al.*, "An empirical study on configuration errors in commercial and open source systems," in *SOSP*, 2011.

[1] https://doi.org/10.5281/zenodo.5153022