Reviewing Oppenheimer et al. - Why do Internet Services Fail, and What Can Be Done About It?

Matthias Beyer and Julian Ganz
Faculty of Computer Science
Hochschule Furtwangen
Robert-Gerwig-Platz 1
78120 Furtwangen

***Abstract*—In this paper, we review "Why Do Internet Services Fail, and What Can Be Done About It?" by Oppenheimer et al.**

I. Introduction

In 2003, Oppenheimer et al. published a paper titled "Why Do Internet Services Fail, and What Can Be Done About It?" [1] where they examine more than 500 user-visible service failures from three large-scale Internet services.

Note that, in this context, the services analyzed were probably large scale at the time. However, in 2003, nearly 15 years ago, the scale of services was probably rather low compared with today. For example, many services (e.g. "Facebook"[2] and "Youtube"[3]) did not exist back then.

Although they state that 100% availability can not be guaranteed, they conclude in five points how failure rates could be minimized. Oppenheimer et al. also suggest a world-readable failure database as well as more, industry-wide research in the field of system reliability as well as recovery benchmarking would greatly improve system availability and maintainability for everyone.

In this paper we review their approach and findings, how the failures were analyzed and whether their results are representative.

In the first part of this paper we summarize what the dataset Oppenheimer et al. used looked like, how their approach for examining the dataset was and what they found. In the second part we review their approach and findings. We conclude with an overall impression of their paper.

II. Oppenheimers' (et al.) data

Oppenheimer et al. worked with three services. These services are hosted in a geographically distributed manner. Each site is build from three tiers: One load-balancing tier, one stateless front-end tier and one backend tier. The back-end tier is responsible for persisting data and making it available to the front-end tier. The front-end tier may cache or queue data, but not persist it.

Each service was categorized into one of the following types:

- An *Online* service portal.
  Load balancing is achieved via client cooperation. The front-end tier is responsible for online services like e-mail, newsgroups and, in this case, for a web proxy service. The service was deployed to Solaris SPARC and x86 machines [1, pp. 1 f.].
- A service were read actions were the majority of access, called *ReadMostly*.
  Load Balancing is achieved via DNS redirection. The front-end tier is responsible for locating the data on the back-end machine(s) and routing it to clients. The service was deployed to x86 machines running an open-source operating system. [1, pp. 1 f.].
- A *Content* hosting service.
  Load balancing is achieved via client cooperation. The front-end tier is responsible for locating the data on the back-end machine(s) and routing it to clients. The service was deployed to x86 machines running an open-source operating system. [1, pp. 1 f.].

The front-end software of all three sites is developed by the service provider. The back-end software of the *Online* service is not developed by the service provider of the site, but the back-end software of the *ReadMostly* and the *Content* services is (see Table I).

All in all, Oppenheimer et al. had

- approx. 207 million hits per day
- more than 3000 machines for approx 20 sites which are served to clients
- 501 component failures
- 107 services failures

recorded from these three services [1, p. 2].

The data was fetched from a 3 to 7 month study.

III. Oppenheimers' (et al.) approach

Each failure report was categorized according to the supposed root cause[1]: each failure report was assigned a "cause"

[1]The authors noted that the root cause itself may be caused by an underlying flaw of some sort.

TABLE I: Custom written Software in Services

	Custom Software		
	LB	Font-end	Back-end
Online		✓	
ReadMostly		✓	✓
Content		✓	✓

and a "location" from a predefined list. For cause, the list contained the following classes:

- node hardware
- network hardware
- node software
- network software
- environment
- operator error
- overload
- unknown

For location, the list contained:

- front-end node
- back-end node
- network
- unknown

After categorization, the number of component and service failures were counted for each combination of cause and location for each of the three services. Because of categorization difficulties and ambiguities, some combinations were merged together for some services. For example, Internet Service Provider (ISP) related outages were often assigned the cause "unknown" and the location "network".

Additionally, for each source-location category, an average Time To Repair (TTR) was calculated from the TTRs provided by the individual reports in a category. The authors note that in the context of their paper, the TTR refers to the time between the registration of the failure until normal operation is achieved again.

Based on the relative number of occurrences of different failures, a number of measurements for avoiding failures are evaluated. For example, the number of failures which could have been prevented with a measurement were counted and compared to the cost of the measurement.

IV. Oppenheimers' (et al.) findings

Overall, Oppenheimer et al. found out, that the majority of component failures were caused by Operator failure via maintaining failures rather than by faulty bug fixes. Also, only few of the failures were caused by the backend infrastructure of the respective services. For the *Content* and *Online* services, 21% and 13% of the analyzed component failures led to system failures. In case of the *Online* and *Content* services, the service failures were mainly caused by the frontend infrastructure (more than two thirds) and in case of the *ReadMostly* service, the network infrastructure was the main source of service failure. In all cases only few failures were not categorized in either Frontend-, Backend- or Network-failures.

For the service failure time to repair, Oppenheimer et al. found, that

> [...] operator errors often take significantly longer to repair than do other types of failures; [...]

[1, chapter 3.3]. These values can, as they state, be misleading. If an operator assigns a low priority to a failure, the TTR can be significantly longer than it would be with a high priority assigned to the failure. They also note that this does not depend on the failure alone, but also on the context the failure appears in.

Oppenheimers et al. findings show that the average TTR for the *Content* service is significantly higher for backend failures than for the other types of services. They also show that failures in the system of the *Online* service take approximately the same time to repair independent of they context they appear in. The *ReadMostly* service takes few hours to repair in relation to the others.

Using the mitigation techniques, Oppenheimer et al. list, 26 failures alone could have been mitigated or entirely avoided in the *Online* service context by applying *Online correctness testing*, a technique for actively testing components for correctness. Also, 12 failures could have been mitigated or avoided by exposing software and hardware failures to a monitoring system more intensively.

V. Review: Oppenheimers' et al. approach

The overall approach seems reasonable for identifying techniques for avoiding a large fraction of service failures with as little effort as possible. However, we found a few details of the reasoning rather unsound.

For example, reports from three services were analyzed. Each service was described through its characteristics and architecture. Seemingly to obfuscate the services from which the data was taken, each service was termed by a name corresponding to the "class" of the service (*Online*, *Content*, *ReadMostly*). This resulted in reasoning based on classes of services, with each class being populated by only one service. Thus, it appeared that the reasoning would apply to any service of that class, when the statistics only apply to one specific one.

This could have been prevented by using a larger data set. However, since every single report had to be reviewed manually, a larger set of reports would have also driven the effort for the analysis.

Oppenheimer at al. determined not only failure counts and percentages of component failures which turned into service failures but also average TTRs. However, without the indication of the standard error or derivations, the averages provided are of little significance for the discussion. Mere averages cannot be used for arguing in favor for counter measurements if they only prevent or avoid a fraction of the failures.

For example, consider a countermeasure preventing half of the failures of a certain category. If the measurement only prevents easy-to-fix failures with a low TTR, the overall down-time may only be reduced marginally. It is thus unsurprisingly that the average TTRs were only used to indicate that

> operator errors often take significantly longer to repair than do other types of failures[.]

While this statement is still based on unsound statistics, Oppenheimer at al. proceed discussing counter-measures in order to reduce the number of component failures rather than the overall service down-time. A number of selected techniques are discussed and evaluated based on an estimate of the number of failures which could have been prevented using a specific technique.

Apparently, only a subset of the failures were analyzed. It was not stated how those failures were selected or why the evaluation was only performed on some subset. However, since the evaluation was performed manually, one may guess that the reason for the selection was the reduction of effort.

Overall, while the approach does seen reasonable, the results would doubtlessly be of greater quality with a larger data set and more man-power. However, at the time, the amount of data accessible to the researchers may have been limited. Also, even if the researchers had access to a larger data set, the time available for performing the *manual* evaluation might have been limited.

VI. Review: Oppenheimers' et al. findings

Although the overall findings of Oppenheimer et al. are not surprising - surely Operator is a great source of service failure - some details are rather surprising.

For example, Oppenheimer et al. found that in the context of the *ReadMostly* service, the majority of failures were caused by the network infrastructure rather than by the backend infrastructure.

Another interesting point Oppenheimer et al. make, is that backend failures are significantly harder to repair than frontend failures. They also state that failures in different domains take more time to repair than in others. This statement might be misleading, though. As stated in V, normalizing services by their class is a suspect approach, as one might get the impression that findings apply to all instances of a certain class. Thus, we want to relativise Oppenheimers statements: They found that, for the services for which they examined failure databases, the service in the *Online* class had almost equal TTR for both backend, frontend and network infrastructure. This was not the case for the *ReadMostly* and *Content* class services.

This also becomes obvious from the evaluation of possible counter-measures. Oppenheimer et al. state that some of the counter-measures are already in place in some services. This statement alone shows that the results are very specific to the certain services analyzed. It also shows that the analysis focused on *additional* counter-measures which could be employed in order to improve the situation –from a 2003 perspective. Hence, we would take their findings with a grain of salt.

All in all we can state that their findings are, to some extend, still valuable. However, the value may have shifted somewhat from a recommendations for deployment of new techniques to motivating techniques which may now, in 2017, be common in large-scale services.

VII. Conclusion

Overall we can conclude that the structure of the paper Oppenheimer et al. presented is not optimal. The data, analysis and findings are mixed in the chapters rather than structured clearly. Also, they suggest world-readable failure databases and benchmarks, but fail to provide a solid explanation why these approaches seem valuable.

Oppenheimer et al. present their figures in a very detailed manner. They explicitly state how to interpret each found value and what to keep in mind during reviewing, which we liked.

On the other hand, some of their statistics are missing some critical values which would help the reader a lot for estimating the impact of the respective figure. For example, their mitigation statistics do not state how much failures could not be avoided, which yields the statistic rather useless. If three of 200 failures could've been avoided by the respective mitigation technique, the technique would be rather useless compared to if it would have avoided three of eight failures.

We also recognized some things missing in the paper Oppenheimer et al. published. We consider the tooling to deploy software to production a critical point in an evaluation of TTR. For example, the whole topic of software configuration management[4], which got more and more attention since Oppenheimer et al. published their paper, is never mentioned in the paper. Tools like ansible, puppet, chef or nix(ops)[5] (which was developed *after* Oppenheimer et al. published their paper) are designed to make maintainance of a system easy[6] and even reproducible[7]. With decent configuration management tools in place, an operator-failure could be rolled back without effort (and, with certain tools, even atomically[8]).

We appreciated that Oppenheimer et al. showed examples for operator failure, such as component failures that led to loss of a failure-database in case of the *Online* infrastructure as well as human error which also led to loss of the complete failure database of the *Content* infrastructure. These simple, but disastrous failures show how important the topic of failure management is. Current failures show that Oppenheimer et al. addressed a problem which is of high importance and which is still not solved yet. For example, the incident at gitlab [9] in early 2017 shows the criticality of this topic today.

References

[1] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251460.1251461

[2] The Sunday Indian. [Online]. Available: http://www.thesundayindian.com/en/photo-albums/241/

[3] J. Graham. (2005) Video websites pop up, invite postings. [Online]. Available: https://usatoday30.usatoday.com/tech/news/techinnovations/2005-11-21-video-websites_x.htm

[4] A. Dearle, "Software deployment, past, present and future," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 269–284. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.20

[5] E. Dolstra, M. Bravenboer, and E. Visser, "Service configuration management," in *Proceedings of the 12th international workshop on Software configuration management*. ACM, 2005, pp. 83–98.

[6] D. Hall, *Ansible Configuration Management*. Packt Publishing Ltd, 2013.

[7] E. Dolstra and A. Hemel, "Purely functional system configuration management." in *HotOS*, 2007.

[8] S. van der Burg, E. Dolstra, and M. de Jonge, "Atomic upgrading of distributed systems," in *Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades*. ACM, 2008, p. 8.

[9] gitlab.com. (2017) Gitlab.com database incident. [Online]. Available: https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/