Towards Robust Distributed Systems

Sebastião Amaro
sebastiao.amaro@tecnico.ulisboa.pt
U. Lisboa & INESC-ID

***Abstract*—Distributed Systems are an essential part of today's modern infrastructure and underlie many critical systems and services. Given their criticality, we need tools to assess and improve their reliability which is commonly achieved by subjecting the systems to faults and observing whether their behavior still matches the specification. Unfortunately, the complexity of distributed systems, composed of different components that can fail in unpredictable ways, either due to internal faults, unexpected interactions among the components, or between them and the environment, makes assessing reliability extremely challenging. State-of-the-art tools are either system-specific (e.g.: PACE), and thus not broadly applicable, or rely on randomized fault injection which hampers reproducibility (e.g.: Chaos Engineering). In this Ph.D., we plan to research and develop novel tools and techniques for assessing the robustness of distributed systems that are black-box, efficient, and reproducible by leveraging recent advances in Linux kernel observability and process control.**

1. Introduction

Distributed systems are at the core of our modern digital society and power many of the services and infrastructures we rely on a daily basis. Unsurprisingly, when those systems fail, the result is failures and losses that can lead to large losses in revenue to organizations [1]. Outages occur more often than we think, and for different reasons. To mention just a few, these can be due to natural phenomena such as heat waves [2] and lightning strikes [3], human error [4], or even unpredictable bugs in the software [5]. It is therefore of the utmost importance to have tools and techniques to assess the reliability of distributed systems. Previous work [6] has demonstrated how software-based fault injection tools have evolved to achieve fault representativeness, efficiency, and usability, proving that they are a powerful tool for assessing distributed systems' reliability. While conceptually simple, in practice fault injection is extremely complex in part due to the inherent complexity and heterogeneity of distributed systems. Distributed systems' complexity stems from different reasons, for example, the complex nature of the algorithms they employ on uncontrollable environments such as the network and the physical hardware, which leads to non-deterministic behavior. Their heterogeneous nature stems from the fact they employ multiple different frameworks and libraries, which have distinct dependencies and rely on different software. These two characteristics make assessing the reliability of these systems extremely hard since there are vast amounts of possibilities as to where faults can appear, not only in the components themselves but in the interactions between them and with the environment.

Therefore, a lot of tools focus on specific components [7], [8], [9], [10], [11], [12], [13], [14], [15]. However, faults may depend on multiple components which make these tools not the best to access highly heterogeneous systems. One approach that has become popular among leading IT companies is Chaos Engineering [16] which creates experiments that randomly inject faults into the production system and evaluate several key metrics. However, its randomized nature which does not depend on the system state, makes it hard to reproduce and often understand the underlying cause. Given the current limitations of the state-of-the-art, our main goal is to develop tools and techniques that meet three main criteria. First, it must be reproducible, allowing developers to inject faults based on the system state rather than at random or based for instance on time, and hence allow to precisely understand and reproduce the conditions that led to the failure. Second, it must treat the system as a black-box, so that it can be applied to any system without requiring system-specific implementations. Third, it must be efficient to not substantially hinder system performance, and hence potentially hiding faults that depend for instance on specific interleavings. To achieve this goal, we will leverage recent advances in Linux kernel observability and process control, namely eBPF [17]. eBPF provides a safe way to monitor the system state with minimal overhead. It is safe because all the programs must pass through the eBPF verifier, and at the same time application agnostic since it does not depend on what application is running. Based on these primitives, we plan to research and develop techniques for efficiently building and maintaining the application state and, based on it, allow the developer to inject faults at arbitrary system states.

2. Overview

We now discuss the key ideas to achieve our goal of building a reproducible, black-box, and efficient fault-injection tool. We envision two main components: System Tracing and Fault Injection.

System Tracing. One of the main challenges of our approach is to efficiently collect information related to the

different aspects of the system state, such as network, disk I/O, memory, threading, etc. We intend to leverage eBPF to achieve this. eBPF provides multiple ways to observe the kernel with mechanisms such as `kprobes`, `kretprobes` and `tracepoints` which enables the user to run user-written code when a certain kernel function, or a certain event happens in the system. It also provides `uprobes` and `uretprobes`, which allow to hook into userspace programs and run user code at certain function invocations.

At the network level, we can precisely track things such as active connections and network usage at specific times, among others. At the disk level, we can trace the latency of I/O operations, how much I/O processes are doing, as well as the cache miss/hit ratios. System calls are another major point of observation, which offers us the ability to track process and thread lifecycles, among others. This plethora of information allows us to precisely track process state and construct a deterministic state at any point in time. Because this state can quickly balloon to a very large size, we will also research techniques to efficiently represent the process state.

Fault Injection. For fault injection, we aim to leverage the primitives discussed above but rather than tracing we plan to skip the behavior of certain system calls which allow us to model and emulate a large class of faults and failures. We also aim to model resource constraint faults and gray faults (faults that may lead to gray failures) which are essential to assess the reliability of performance sensitive systems, among others. To this end, we will leverage `cgroups` (control groups) functionalities which allow us to group processes and limit their resource usage in different aspects such as file system cache, I/O bandwidth limit, CPU quota limit, or CPU set limit. These faults can be triggered based on the state of the system, as dictated by the System Tracing component. This enables a vast and diverse set of faults and scenarios, such as: limiting the CPU quota of a machine after a process reaches a certain state to emulate a noisy neighbor, limit the I/O of a process after reading a file from disk, to emulate a disk problem, dropping all packets outward, when the system creates a certain amount of threads to emulate a network partition when a protocol reaches a certain stage, among many others.

While Tracing and Fault Injection can be done in kernel space, the control logic is likely to reside in userspace. Given that transitioning from one to the other is expensive, we will research techniques to minimize the transitions needed.

3. Preliminary results

We have preliminary work and results on systems observability and control leveraging kernel functionalities. More specifically, we have extended and redesigned Kollaps [18], a scalable network emulator, as follows. We have replaced the existing monitoring system with a novel eBPF-based system, specifically one based on socket filters [19], and perf [20] resulting in a reduction in CPU usage up to 90%. Using kernel functionality, we can also inject faults in the target system, such as packet loss or connection losses. The resulting work is currently under submission. While restricted to the network subsystem, this work allowed us to gain intimate knowledge and experience with key technical and technological subjects and hence laid the basic foundations for the work to be conducted during the rest of the Ph.D.

4. Work to be done

Status: *I am at the beginning (first year) of my Ph.D. program, which is expected to last for four years.*

During the first year and a half, we will research and develop the System Tracing component, namely how to efficiently represent information and bring it to userspace. For the next eighteen months we will focus on the Fault Injection component and for the final year we will develop the interface and a domain-specific language to express the experiments. Given that the work on the main components is of independent interest, we plan to publish a paper on each one of them, and a final paper describing the system as a whole, evaluated using real industry-grade open-source systems as target applications.

5. Related Work

We now present the current approaches to Fault Injection in distributed systems.

Specific-system tools. PACE [11] is a crash exploration framework. PACE explores correlated crash vulnerabilities in distributed filesystems by systematically generating persistent states that exist in the execution in the presence of correlated crashes. Sieve [7], is an automatic reliability-testing tool for cluster-management controllers. MEPFL [8] is an approach for latent error prediction and fault localization of microservice applications. FCatch [9] aims at automatically detecting bugs, specifically, Time-of-Fault (TOF) bugs, in cloud systems. CrashMonkey [10], automatically simulates faults at different points of a given workload and tests the correctness of the file system after each fault injection. Peter Alvaro et al. [15], developed an approach to fault injection called Lineage-driven Fault Injection. It is based on the concept of data lineage, used in database literature. This lineage can be seen as the model of a particular execution of a process **Generic tools**. Faultsee [21], is a toolkit composed of (1) Faultsee Domain System Language (FDSL), a configuration language based on YAML to concisely describe the experiment, and (2) a Faultsee platform to deploy and automatically execute the experiments specified in FDSL. Jepsen [22] is research that makes analyzes of real world and modern distributed systems and checks for bugs with the usage of fault injection. **Chaos Engineering**. Chaos Monkey [23], is a tool that randomly terminates virtual machine instances and containers in a production environment.

Acknowledgments This work was supported by Fundação para a Ciência e a Tecnologia (FCT) under grants UIDB/50021/2020 and PTDC/CCI-COM/4485/2021 (Ainur).

References

[1] "Google lost $1.7m in ad revenue during youtube outage," accessed: 2023-1-27. [Online]. Available: https://www.foxbusiness.com/technology/google-lost-ad-revenue-during-youtube-outage-expert

[2] "Google's london data center outage during heatwave," accessed: 2023-1-27. [Online]. Available: https://www.datacenterdynamics.com/en/news/googles-london-data-center-outage-during-heatwave-caused-by-simultaneous-failure-of-multiple-redundant-cooling-systems/

[3] "Lightning in belgium disrupts google cloud services," accessed: 2023-1-27. [Online]. Available: https://www.datacenterknowledge.com/archives/2015/08/19/lightning-strikes-google-data-center-disrupts-cloud-services

[4] "Data center backup failure blamed on 1 person," accessed: 2023-1-27. [Online]. Available: https://www.datacenterknowledge.com/manage/data-center-backup-failure-blamed-1-person-not-nyse-leadership

[5] "Google dashoard bug incident," accessed: 2023-1-27. [Online]. Available: https://status.cloud.google.com/incident/zall/20013

[6] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing dependability with software fault injection: A survey," *ACM Comput. Surv.*, vol. 48, no. 3, feb 2016. [Online]. Available: https://doi.org/10.1145/2841425

[7] X. Sun, W. Luo, J. T. Gu, A. Ganesan, R. Alagappan, M. Gasch, L. Suresh, and T. Xu, "Automatic reliability testing for cluster management controllers," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 143–159. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/sun

[8] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 683–694. [Online]. Available: https://doi.org/10.1145/3338906.3338961

[9] H. Liu, X. Wang, G. Li, S. Lu, F. Ye, and C. Tian, "Fcatch: Automatically detecting time-of-fault bugs in cloud systems," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 419–431. [Online]. Available: https://doi.org/10.1145/3173162.3177161

[10] J. Mohan, A. Martinez, S. Ponnapalli, P. Raju, and V. Chidambaram, "Finding Crash-Consistency Bugs with Bounded Black-Box Crash Testing," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2018)*. Carlsbad, CA: USENIX Association, 2018.

[11] R. Alagappan, A. Ganesan, Y. Patel, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Correlated crash vulnerabilities," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. GA: USENIX Association, 2016, pp. 151–167. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/alagappan

[12] J. Mohan, A. Martinez, S. Ponnapalli, P. Raju, and V. Chidambaram, "Finding crash-consistency bugs with bounded black-box crash testing," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 33–50.

[13] H. Liu, G. Li, J. F. Lukman, J. Li, S. Lu, H. S. Gunawi, and C. Tian, "Dcatch: Automatically detecting distributed concurrency bugs in cloud systems," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 677–691. [Online]. Available: https://doi.org/10.1145/3037697.3037735

[14] K. Kingsbury and P. Alvaro, "Elle: Inferring isolation anomalies from experimental observations," *CoRR*, vol. abs/2003.10554, 2020. [Online]. Available: https://arxiv.org/abs/2003.10554

[15] P. Alvaro, J. Rosen, and J. M. Hellerstein, "Lineage-driven fault injection," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 331–346. [Online]. Available: https://doi.org/10.1145/2723372.2723711

[16] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *CoRR*, vol. abs/1702.05843, 2017. [Online]. Available: http://arxiv.org/abs/1702.05843

[17] "extended berkeley packet filter." accessed: 2023-1-27. [Online]. Available: https://ebpf.io/what-is-ebpf

[18] P. Gouveia, J. a. Neves, C. Segarra, L. Liechti, S. Issa, V. Schiavoni, and M. Matos, "Kollaps: Decentralized and dynamic topology emulation," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387540

[19] "Linux socket filter," accessed: 2023-2-1. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/filter.txt

[20] "Perf wiki." accessed: 2023-02-06. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[21] M. Amaral, M. L. Pardal, H. Mercier, and M. Matos, "Faultsee: Reproducible fault injection in distributed systems," in *2020 16th European Dependable Computing Conference (EDCC)*, 2020, pp. 25–32.

[22] "Jepsen," accessed: 2023-2-1. [Online]. Available: https://github.com/jepsen-io/jepsen

[23] "Chaos monkey github," accessed: 2023-1-27. [Online]. Available: https://github.com/netflix/chaosmonkey