Understanding storage I/O patterns through system call observability

Tânia Esteves, Ricardo Macedo, Rui Oliveira and João Paulo
INESC TEC & University of Minho

INTRODUCTION

Users and developers often encounter performance, correctness, and dependability issues in their applications that are hard to diagnose and take considerable time to debug.

Existing tools that can assist users in this process have several limitations. First, they often require instrumenting the applications' source code [1]. Secondly, many of these tools only cover the data collection step, delegating the analysis and visualization of traced data to users[2]. Moreover, solutions that provide a complete analysis pipeline are designed for rigid analysis scenarios (*e.g.*, security, distributed systems' causality), lacking the flexibility to perform different and customized analyses [3].

In this work, we introduce a flexible and practical tool that allows users to transparently and non-intrusively trace, analyze and visualize applications' I/O system calls in near real-time while imposing reduced performance overhead for targeted applications.

DESIGN OVERVIEW

We propose a generic tool for diagnosing the I/O interactions between applications and in-kernel POSIX storage systems.

Its design consists of three main components: the *tracer*, the *backend*, and the *visualizer*. The *tracer* uses eBPF technology to intercept syscalls from applications non-intrusively (*i.e.*, without source code instrumentation), collecting their information (*i.e.*, type, arguments, return value) and enriching it with extra context from the kernel (*e.g.*, file type, offset). As soon as data is captured, it parses and forwards it to the *backend* without requiring manual user intervention. Users can then access the traced syscalls through the *backend* and *visualization* components in near real-time, apply filters to get specific data, and build correlation algorithms and customized visualizations (*e.g.*, histograms, time-series plots).

By offering an integrated solution that intercepts syscalls, enriches collected data with relevant context, and provides timely analysis and visualization, our tool facilitates the analysis and enables near real-time visualization of complex I/O patterns for data-intensive applications.

RESULTS

We validated the usability of our framework with four production-level applications. The results show that our framework enables the diagnosis of: *i)* inefficient use of syscalls that lead to poor storage performance in Redis; *ii)* unexpected file access patterns caused by the usage of high-level libraries that lead to redundant I/O calls in Elasticsearch; *iii)* resource contention in multi-threaded I/O that leads to high tail latency for user workloads in RocksDB; *iv)* erroneous file accesses that cause data loss in Fluent Bit.

Importantly, these I/O patterns are observable without resorting to code instrumentation or needing to manually combine multiple tools.

Experimental results show that the proposed tool can collect, parse, and forward to the analysis pipeline all the required information while imposing reduced performance overhead. Specifically, compared to the *vanilla* version where the targeted application runs without any tracer, our tool increases execution time to no longer than 1.38x for the most I/O intensive application (RocksDB).

FUTURE DIRECTIONS

In future work, we aim at further simplifying the diagnosis of applications for users by providing a richer collection of predefined correlation algorithms at the *backend*, which can be used to automate the detection of key I/O patterns. As examples, we could devise algorithms to quickly identify the inefficient behaviors observed in the aforementioned applications, namely for: *i)* finding sequences of syscalls repeated multiple times for a given file; *ii)* finding redundant operations, such as opening and closing a file for every write.

Further, we intend to use our tool to assist research in other scopes, such as security. For instance, the proposed tool could be used for analyzing the storage I/O patterns performed by malware. Such analysis would reveal how different malware families interact with the storage system, allowing users to compare them and find distinctive I/O behavior that could be used to build or improve malware detection tools.

REFERENCES

[1] 2022. Jaeger: open source, end-to-end distributed tracing. Retrieved November, 2022 from https://www.jaegertracing.io

[2] Ibrahim Umit Akgun, Geoff Kuenning, and Erez Zadok. 2020. Reanimator: Versatile high-fidelity storage-system tracing and replaying. In *13th ACM International Systems and Storage Conference*. ACM, 61–74.

[3] Iman Kohyarnejadfard, Daniel Aloise, Michel R Dagenais, and Mahsa Shakeri. 2021. A framework for detecting system performance anomalies using tracing data analysis. *Entropy* 23, 8 (2021), 1011.