Cloud Computing Applications and Services

(Aplicações e Serviços de Computação em Nuvem)

Diagnosing applications I/O behavior through system call observability

Tânia Esteves
tania.c.araujo@inesctec.pt
d12729@di.uminho.pt

2022/2023

Why the need for diagnosing applications I/O requests?

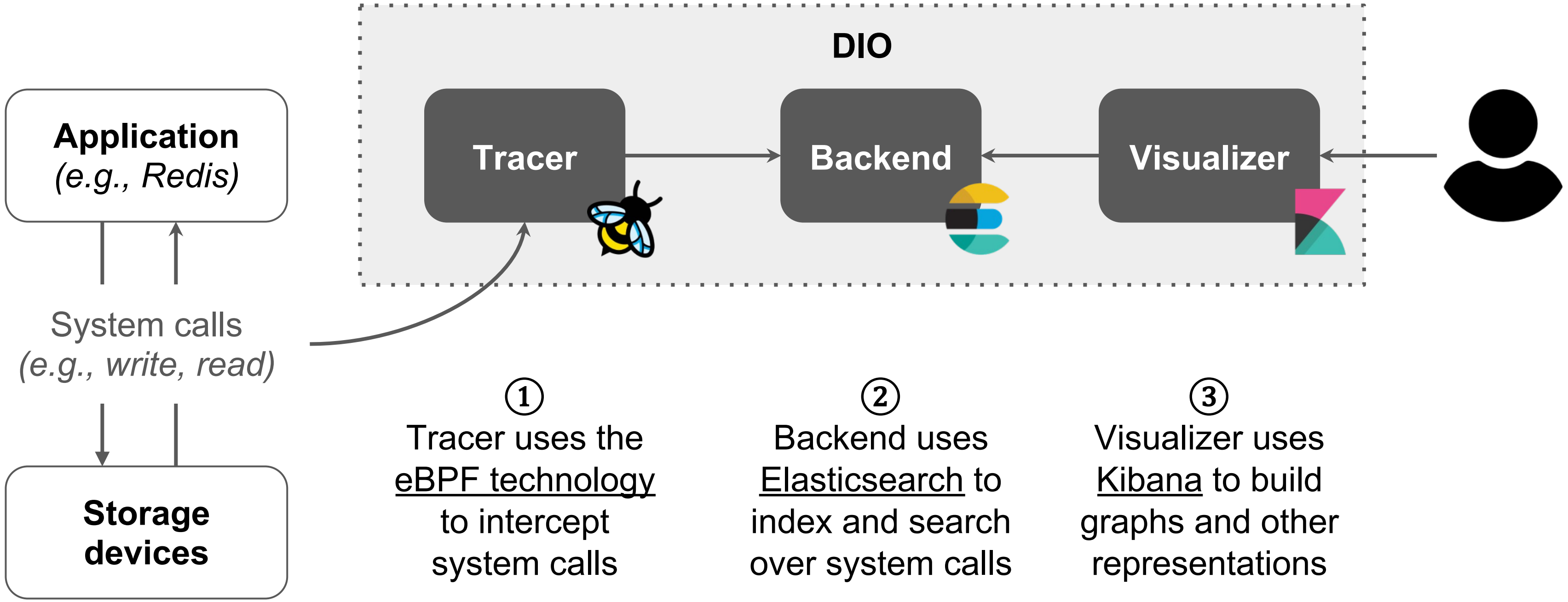
- Applications often exhibit inefficient or erroneous I/O behaviors that can compromise their performance, correctness and dependability:
  - costly access patterns (*e.g.,* small-sized I/O requests or random accesses)
  - redundant operations (*e.g.,* unnecessarily re-opening and closing a given file)
  - I/O contention caused by having concurrent requests accessing shared storage resources
  - erroneous usage of I/O calls (*e.g.,* accessing wrong file offsets)

- Analyze large codebases manually (*e.g.,* Redis has more than 100K LoC) to diagnose these inefficient patterns is a complex and time-consuming task.

DIO: A tool for diagnosing applications I/O behavior through system call observability

- DIO provides a full pipeline for capturing, analyzing, and visualizing I/O system calls made by applications.
- With DIO, users can observe:
  - inefficient use of system calls that lead to poor storage performance
  - unexpected file access patterns caused by the usage of high-level libraries, leading to redundant I/O calls
  - resource contention in multi threaded I/O that leads to high tail latency for user workloads
  - erroneous file accesses that cause data loss

How DIO works

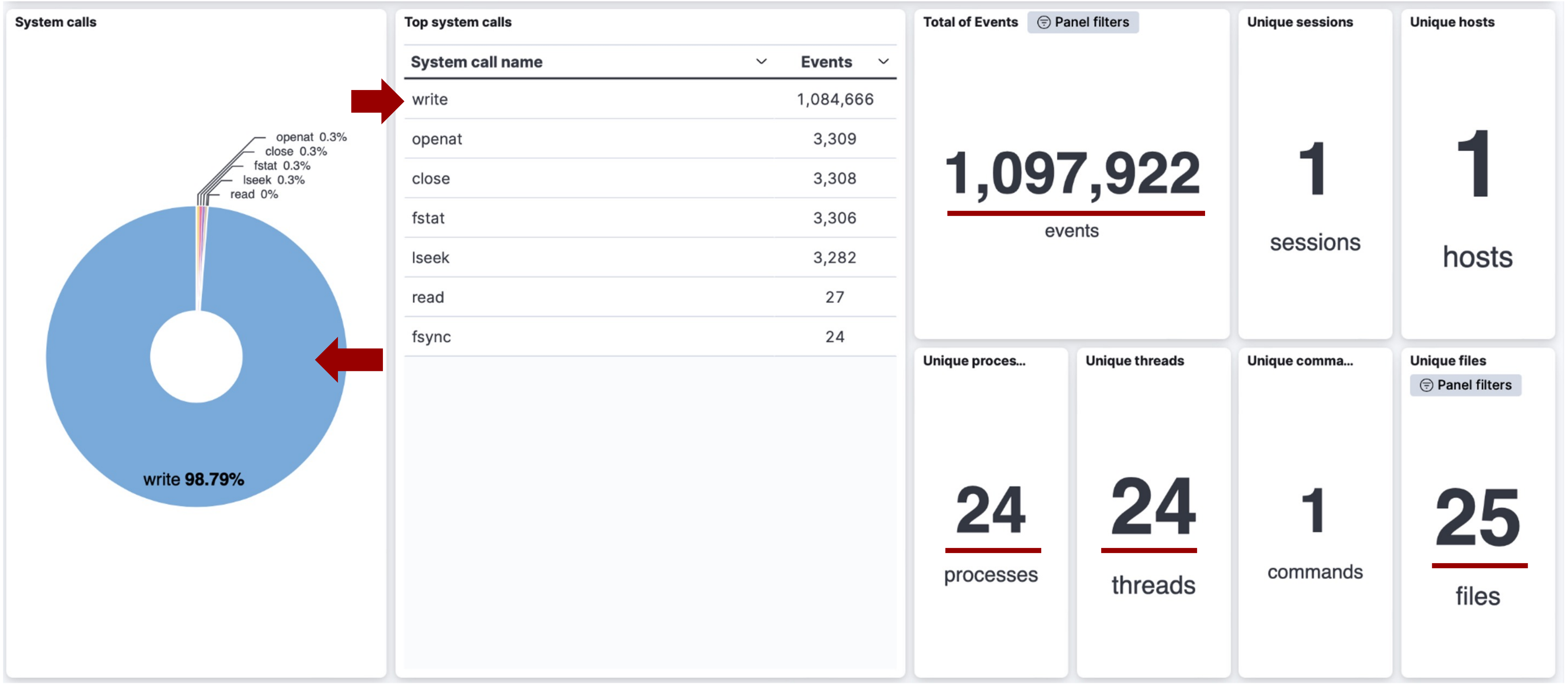


① Tracer uses the eBPF technology to intercept system calls

② Backend uses Elasticsearch to index and search over system calls

③ Visualizer uses Kibana to build graphs and other representations

DIO in practice - Redis use case



System calls

Top system calls

System call name	Events
write	1,084,666
openat	3,309
close	3,308
fstat	3,306
lseek	3,282
read	27
fsync	24

Total of Events — Panel filters: 1,097,922 events

Unique sessions: 1 sessions

Unique hosts: 1 hosts

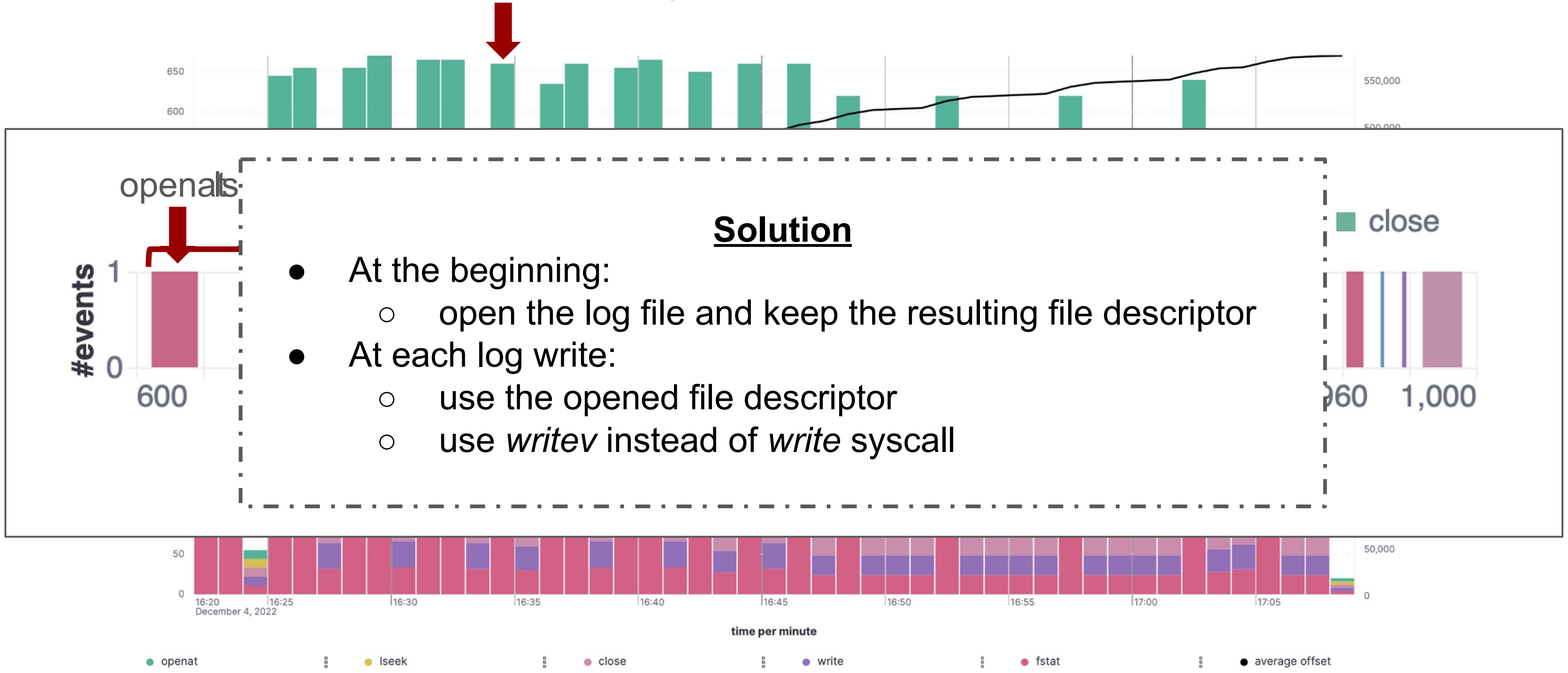
Unique proces...: 24 processes

Unique threads: 24 threads

Unique comma...: 1 commands

Unique files — Panel filters: 25 files

DIO in practice - Redis log file access pattern

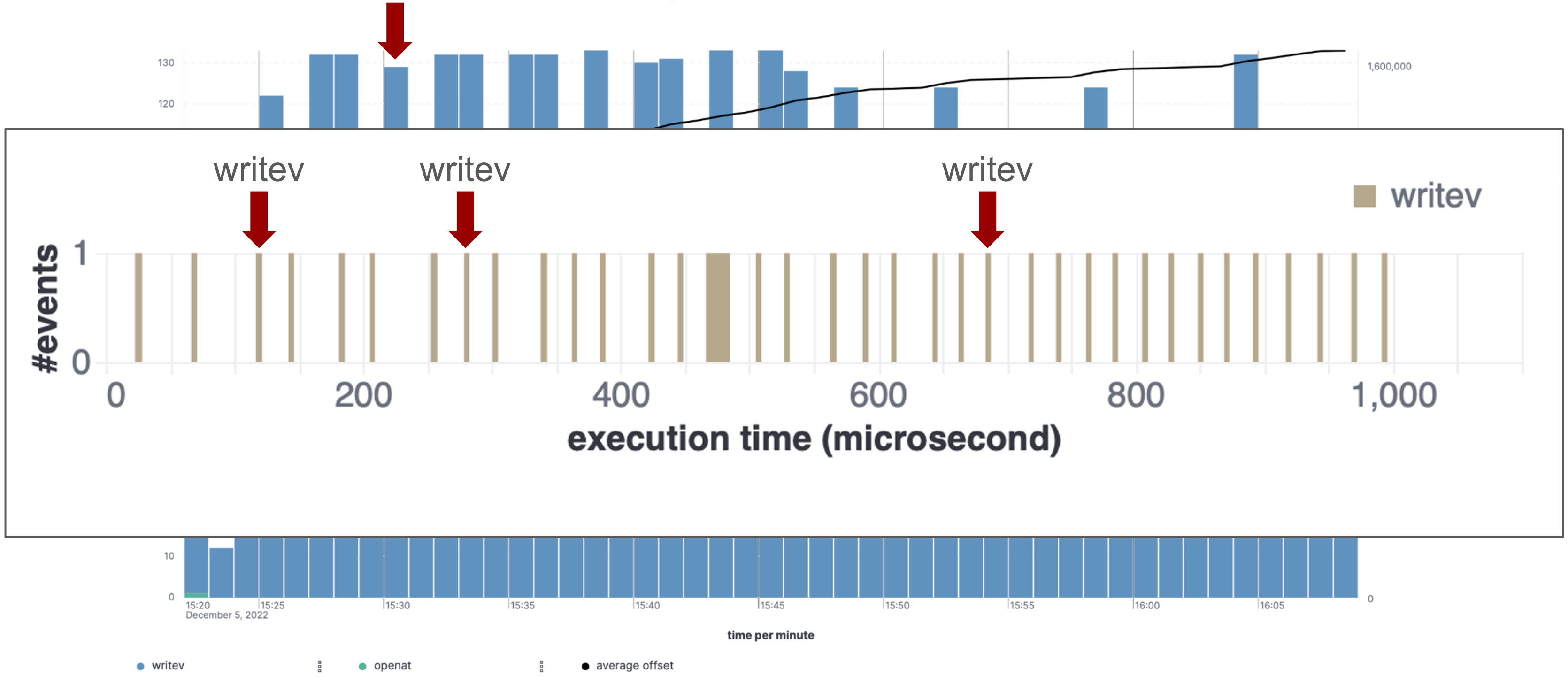


Solution

- At the beginning:
  - open the log file and keep the resulting file descriptor
- At each log write:
  - use the opened file descriptor
  - use *writev* instead of *write* syscall

- Same sequence of system calls repeated over time: **openat** → **lseek** → **fstat** → **write** → **close**
- Many system calls per minute (up to 650)

DIO in practice - Redis log file access pattern



- Only one type of system call is repeated over time: **writev**
- Less system calls per minute (up to 130)

Future directions

- Analyze Ransomware attacks *(ongoing research)*
  - Observe system calls patterns of Ransomware attacks
  - Detect (and prevent) Ransomware attacks based on their system calls patterns
- Analyze I/O events at other OS levels
  - Explore eBPF to trace events at the *Virtual File system* or *Cache* layer
- Improve scalability and performance of DIO
  - Minimize the imposed overhead and capture more events
- Improve data analysis and correlation algorithms
  - Explore machine learning algorithms to automate the analysis process

Questions?