Master's Thesis Project - I (CS57003)

XPLOG++: A Dynamic Observability Framework for Distributed Edge Applications for 6G Compute on Demand Architecture

Guide	: **Professor Sandip Chakraborty**
Presented by	: **Aman Sharma (20CS30063)**
Semester	: **Autumn 2024-25**
Date	: **11th November 2024**

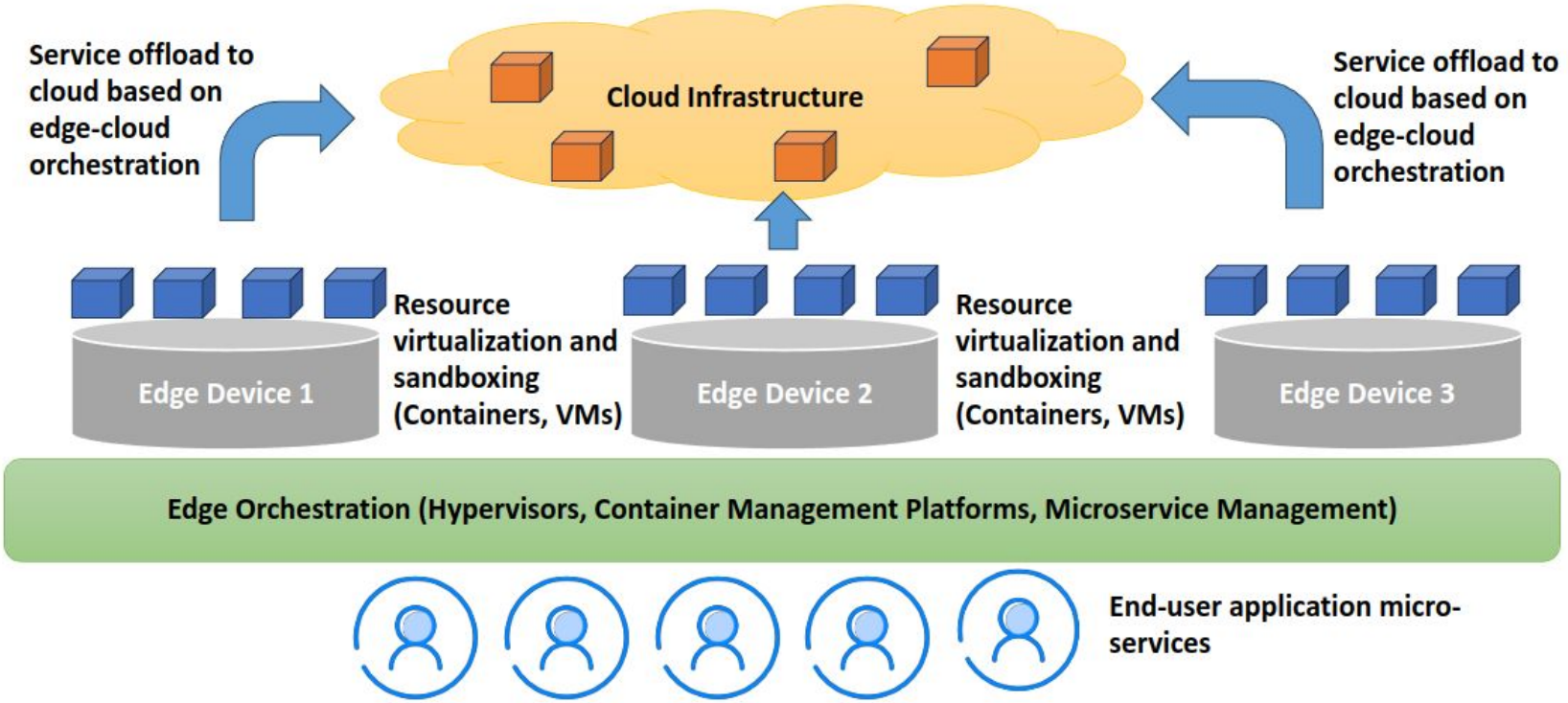
Motivation

6G Compute on Demand Requirements

- 6G networks enable compute on demand (CoD) by leveraging communication towers as hyperlocal servers.

- These towers, equipped with computing capabilities, can dynamically host applications close to users, minimizing latency and maximizing resource utilization.

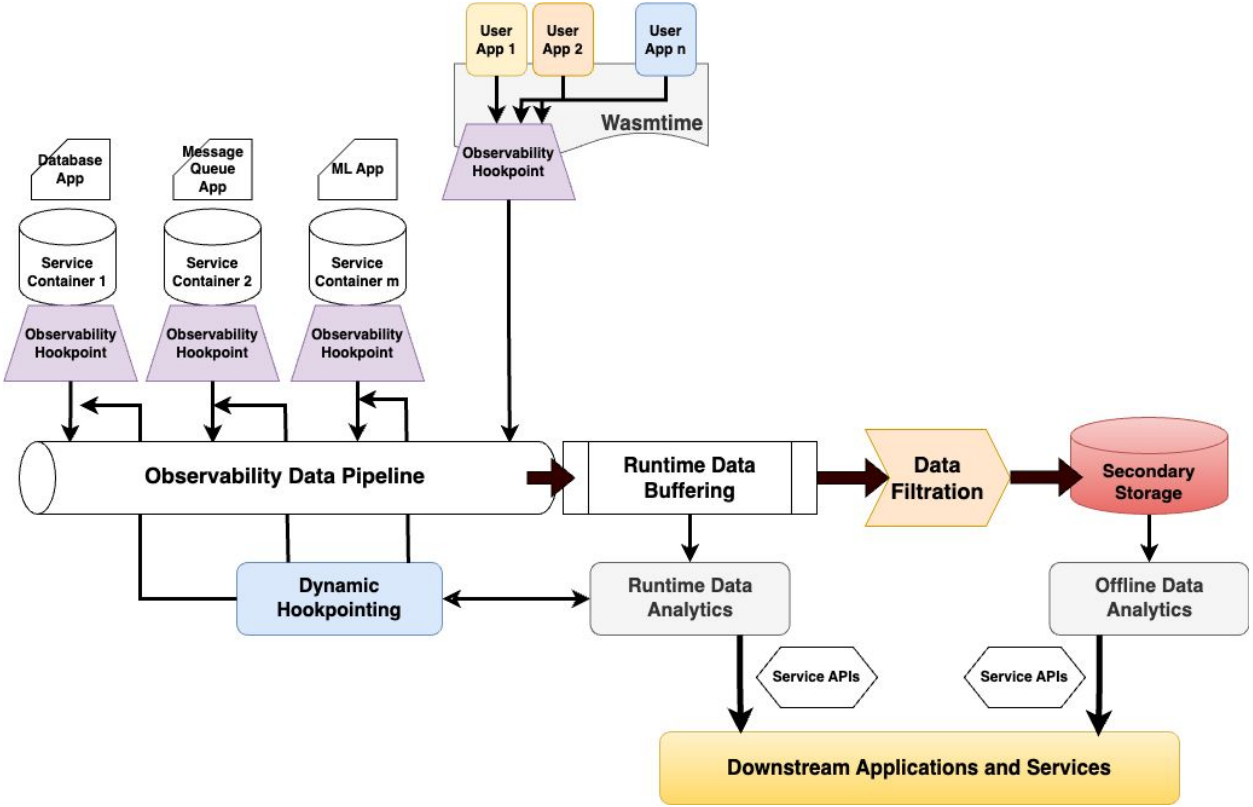
6G Edge Computing Architecture



Need for a Logging Framework

- A robust logging and observability framework is essential to manage and monitor applications running on these hyperlocal servers.
- The framework must efficiently schedule and monitor application performance without introducing excessive overhead, especially for latency-sensitive applications.
- The applications need to be moved when the tower is required for high communication loads.

6G CoD Architecture with System Observability



Existing Work

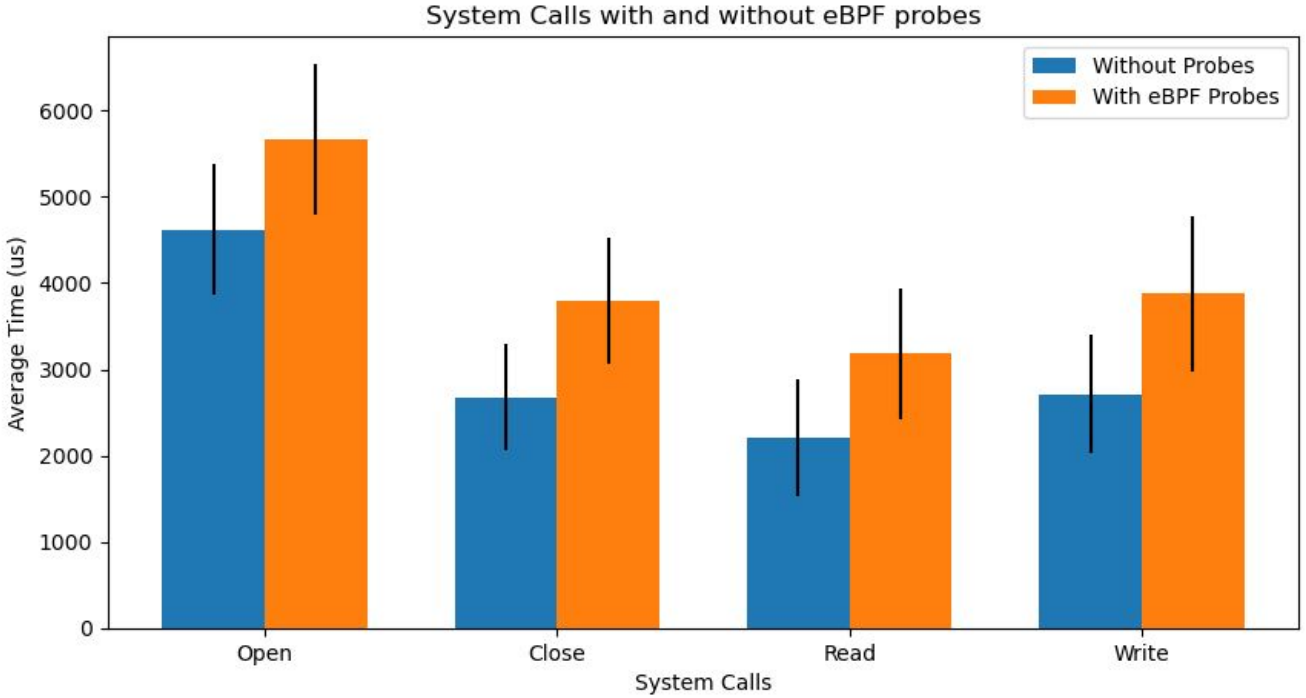
XPLOG: Framework for causally consistent logging

- XPLOG was designed to collect logs from distributed microservices, maintaining a causal order to accurately represent event dependencies.
- XPLOG utilized eBPF (Extended Berkeley Packet Filter) probes to monitor syscalls at the kernel level. This allowed for application-agnostic log collection without modifying the kernel or requiring kernel recompilation.

XPLOG: Limitations

- While XPLOG provided effective multi-layer logging, it lacked dynamic control over which syscalls were logged, resulting in high resource usage. This full logging approach led to increased CPU usage, memory consumption, and network overhead, especially in high-frequency syscalls (e.g., read and write in a database application).

XPLOG: Limitations

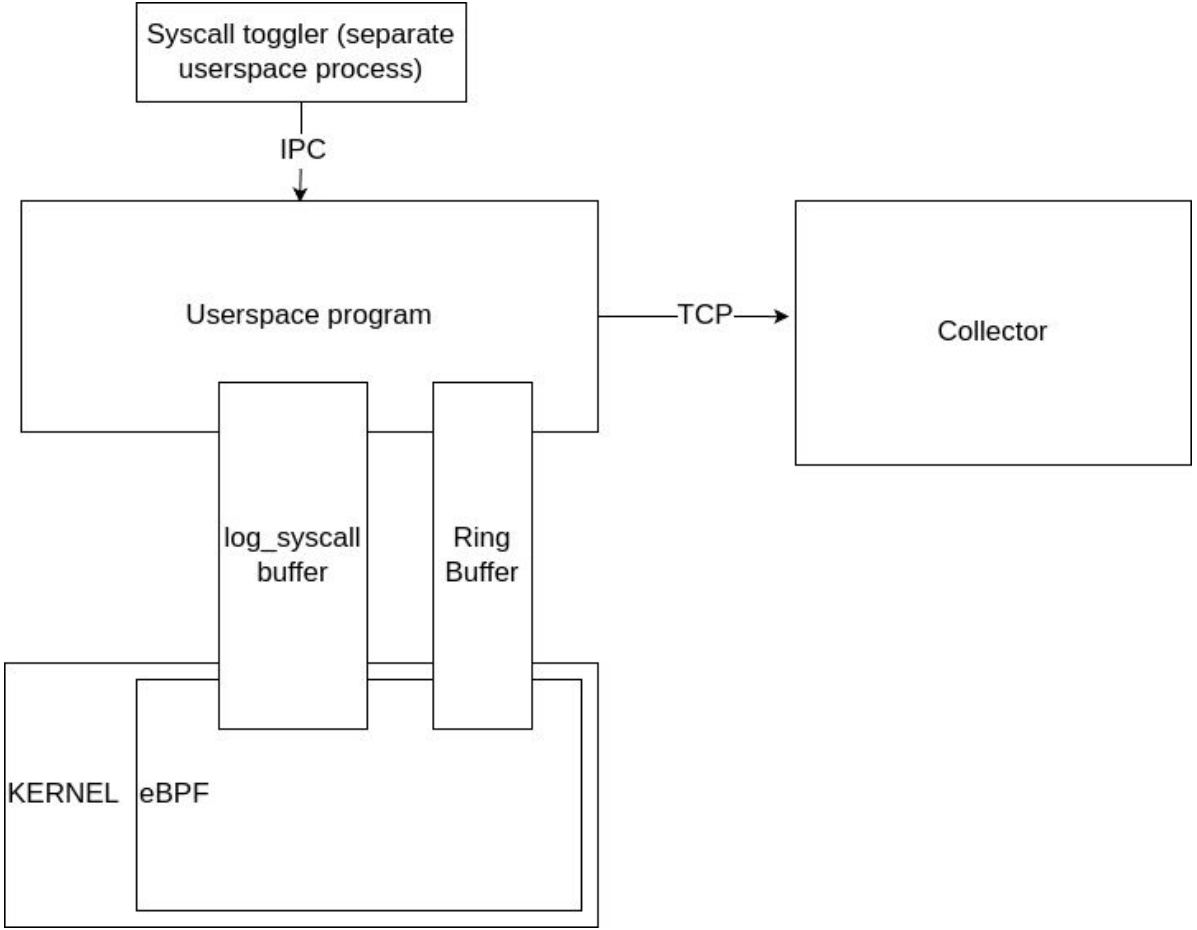


Problem Statement

Objective

- Develop a framework that dynamically logs critical system calls, supporting efficient resource management while meeting 6G requirements for low latency, scalability and easy deployment.

Methodology


Syscall toggler (separate userspace process)
IPC
Userspace program
TCP
Collector
log_syscall buffer
Ring Buffer
KERNEL
eBPF

Results

Experimental Setup

- We measure key performance metrics like CPU usage, memory consumption, log file size and log latency.
- Latency was measured between triggering the syscall and actually writing the log to the output file.
- To remove network latency, we run both the collector and the agent on the same machine for latency measurement.
- For all other metrics, the collector is run on a remote server for the experiment.

Performance Metrics

Metric	Without Dynamic Logging	With Dynamic Logging
CPU Usage	2.95%	0.25%
Memory Usage	792KiB	464KiB
Log File Size	5,645,319 Bytes	156,356 Bytes
Log Latency	38 ms	10 ms

Future Work

Planned Future Works

- Get runtime system metrics like CPU, memory usage etc.
- Dynamically schedule services based on system load

Thank you