Online OS reconfiguration for Cloud DBMS

Hristina Yordanova Grigorova
Advisor: Ilya Meignan--Masson
Systems Research Group
https://dse.in.tum.de/

28.04.2025 – 28.08.2025

Motivation: Research context

Cloud-native Database Management Systems:

- dynamic, multi-tenant environment
- resource utilization, fault tolerance
- flexibility

General-purpose OSes

- heavyweight and resource-hungry

Research gap

Unikernels - alternative to bulky OSes

- specialized and lightweight
- reduced attack surface
- single-address-space
- BUT: limited flexibility

Problem statement

How can we dynamically and securely reconfigure unikernel-based DBMS?

System: Runtime for dynamic reconfiguration

Provide reconfiguration through a mechanism that enables safe and dynamic execution of extensions within the unikernel

System design goals:

- Live adaptability
- Lightweight-ness
- Performance
- Safety
- Portability

Outline

- ~~Motivation~~
- Background
- Design
- Implementation
- Evaluation

Background

Extended Berkley Packet Filter (eBPF)

- OS level reconfiguration
- sandboxed programs in kernel space
- helper functions

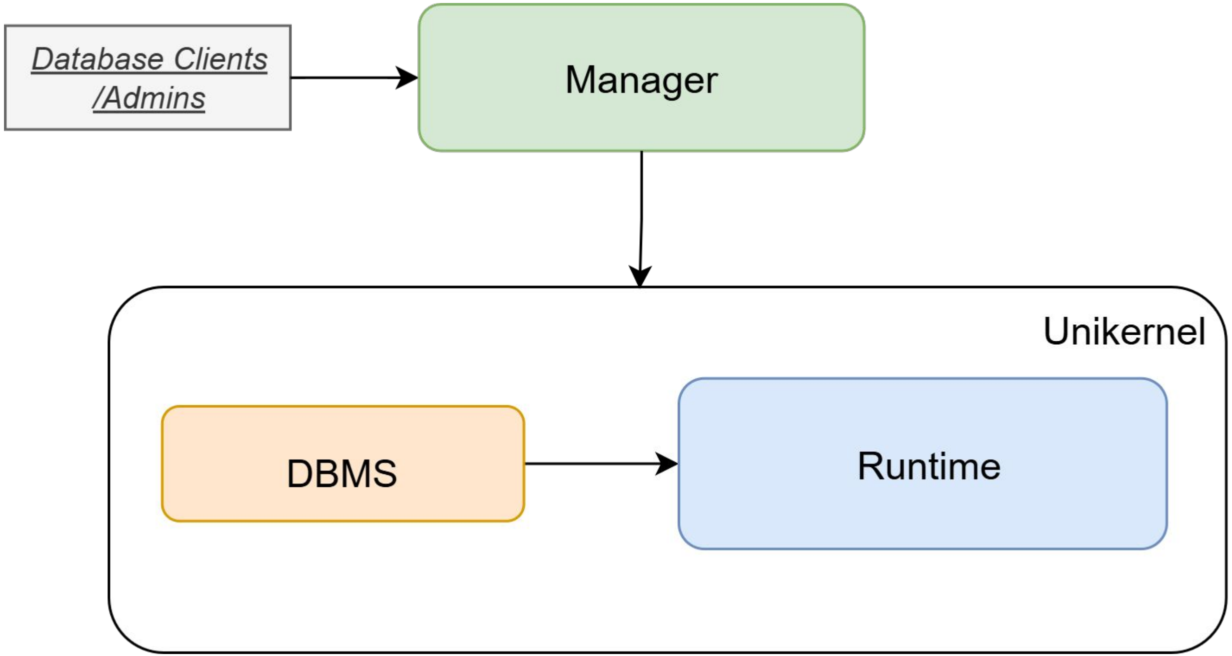
uBPF

- user-space implementation of eBPF

Outline

- ~~Motivation~~
- ~~Background~~
- Design
- Implementation
- Evaluation

System overview



Design overview

- DBMS
  - instrumented with hookpoints
- Runtime
  - execution engine for extensions
- Extensions
  - executed in an isolated environment
  - helper functions

The system is designed to evolve dynamically at runtime.

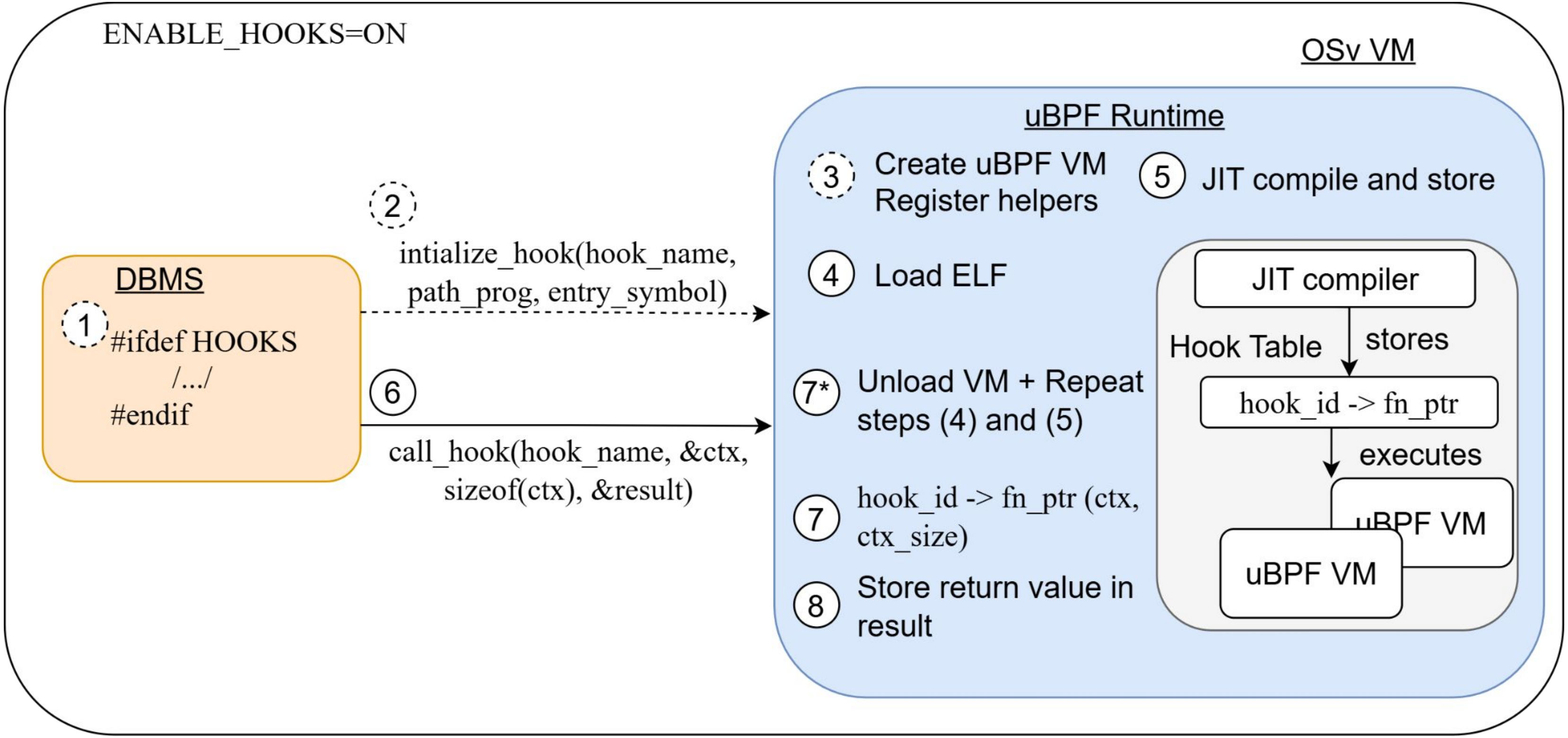
Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- Implementation
- Evaluation

Implementation

- OSv unikernel - lightweight
- eBPF based extension mechanism using uBPF- safety and extensibility
  - helper functions
- JIT Compilation - performance
- DuckDB
  - instrumented with hookpoints

Detailed Workflow



Extensions and Helper Functions

- Hookpoints
  - instrumented into DuckDB
  - replicate original behavior
- Helper Functions
  - internal functionality and state
  - e.g., scheduling queues, controlling buffer access

Hookpoint function	**# Helpers**
`Hash`	1
`ExecuteForever`	9
`ScheduleTask`	1
`Load`	2
`UpdateUsedMemory`	3
`AllocateNewBlock`	2
`Unpin`	5

Outline

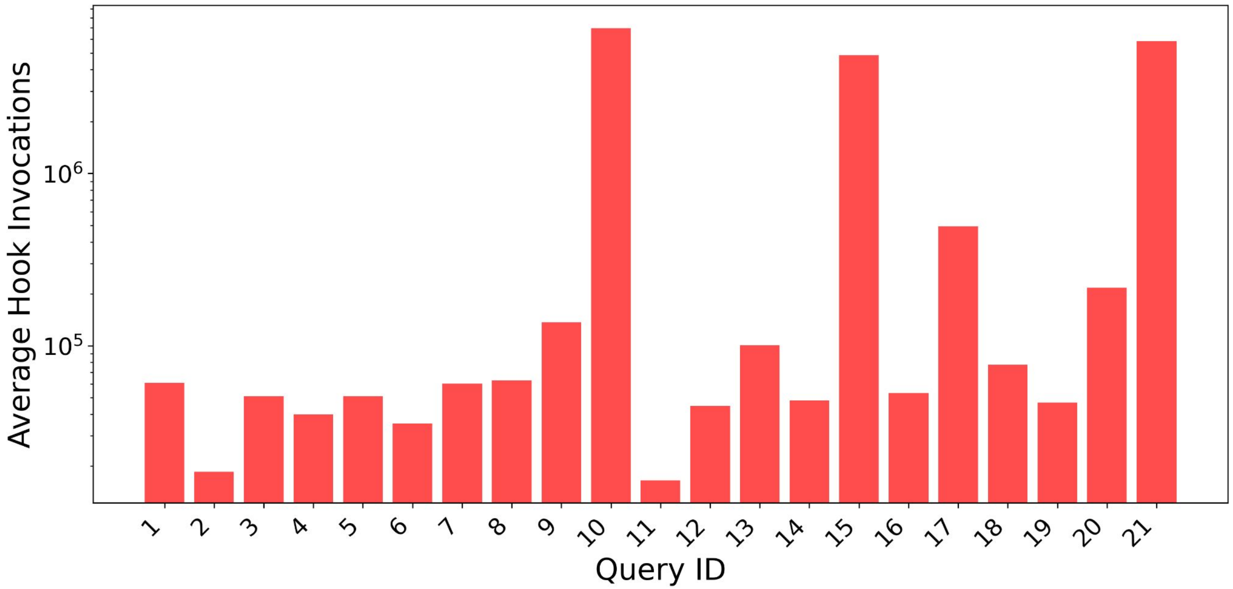
- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- ~~Implementation~~
- Evaluation

Evaluation

- System under evaluation:
  - DuckDB instrumented with 7 hookpoints + uBPF Runtime
- Baseline:
  - DuckDB without hook support
- Research Questions:
  - Can the system preserve the performance of the DBMS?
  - Is the design lightweight, minimal memory overhead?
  - Does the mechanism improve reconfiguration times?

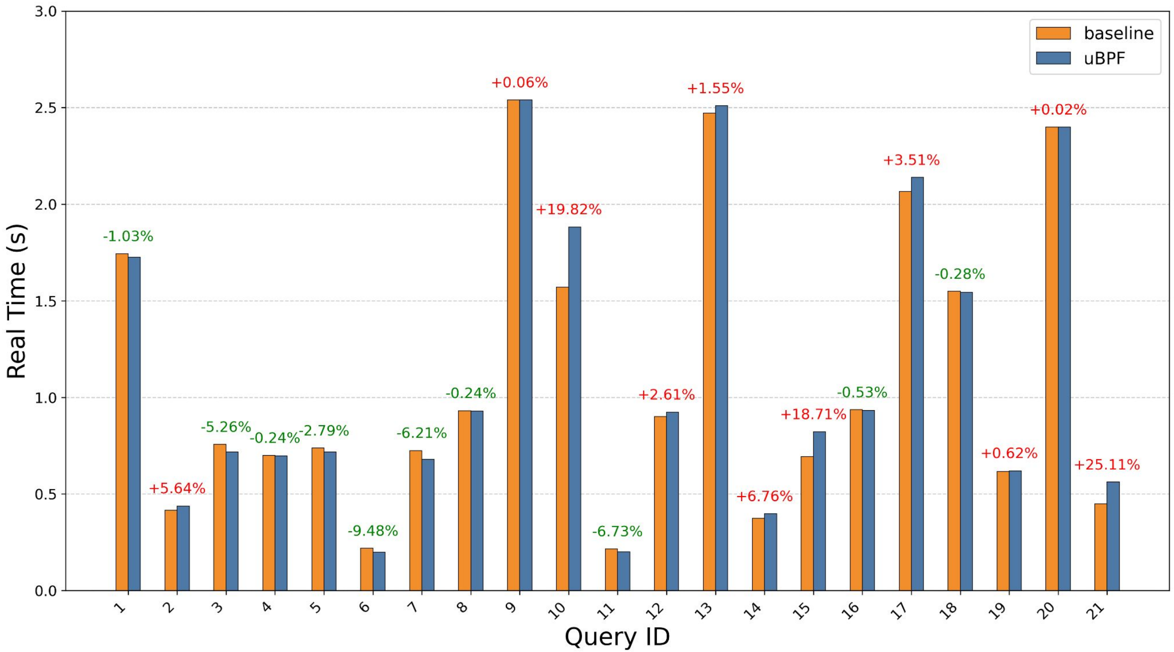
Performance (Number of Hook Invocations)

TPC-H benchmark, 21 queries, scale factor 20



Performance overhead

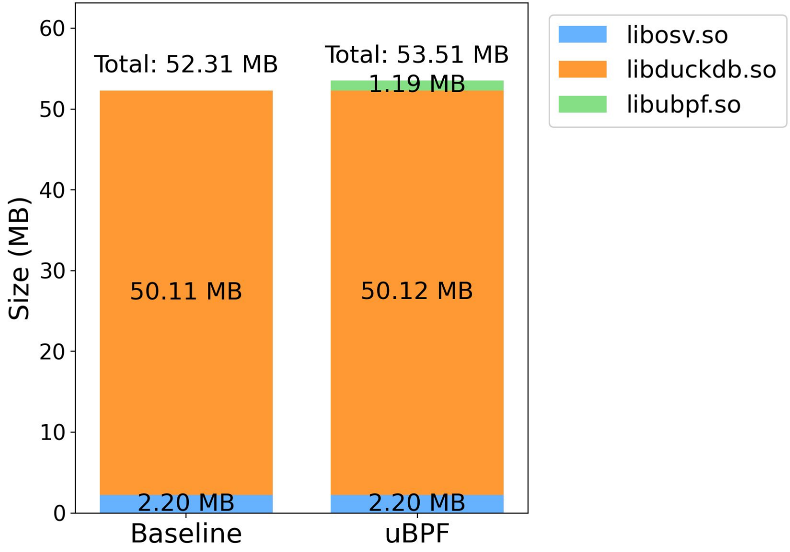
TPC-H benchmark, 21 queries, scale factor 20



The system introduces minimal performance impact

Memory footprint

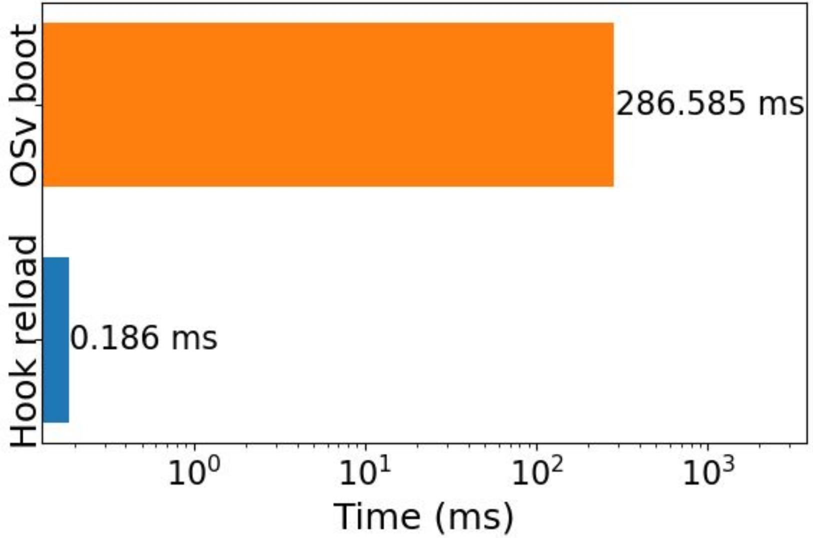
Compare OSv image library sizes



Insignificant memory overhead.

Reconfiguration time

Reboot time vs Reload path



Live reconfiguration substantially outperforms static updates.

Summary

Proposition: Enable safe and dynamic reconfiguration at the DBMS/OS boundary through a lightweight extension mechanism

Extensible, cloud-native DBMS deployments:

- lightweight unikernel
- instrumented DBMS engine
- safe eBPF-based Runtime component