Co-Designing Cloud-Native Database Systems and Unikernels

A Story of Radicalization

Viktor Leis

Technische Universität München

Operating Systems and Database Systems: An Uneasy Relationship

- OS:
  - Manages, multiplexes, abstracts hardware
  - Isolates processes/containers/users from each other
- DBMS:
  - Wants full control over hardware
  - Assumes it owns the machine
- Both implement similar things:
  - Compute scheduling
  - Memory management
  - I/O scheduling, file system, RAID
  - Caching

Case Study 1: Virtual Memory for Caching

Memory-mapping the database file using `mmap` seems perfect:

- OS caches pages in RAM
- Access to all data is transparent
- Uses hardware support (MMU, TLB) to make cached accesses fast

Memory-mapping the database file using `mmap` seems perfect:

- OS caches pages in RAM
- Access to all data is transparent
- Uses hardware support (MMU, TLB) to make cached accesses fast

However, this is almost always a bad idea for DBMSs:

- Problem #1: Transactional Safety
- Problem #2: I/O Stalls
- Problem #3: Error Handling
- Problem #4: Performance Issues

Memory-mapping the database file using `mmap` seems perfect:

- OS caches pages in RAM
- Access to all data is transparent
- Uses hardware support (MMU, TLB) to make cached accesses fast

However, this is almost always a bad idea for DBMSs:

- Problem #1: Transactional Safety
- Problem #2: I/O Stalls
- Problem #3: Error Handling
- Problem #4: Performance Issues

$\Rightarrow$ Database systems virtually always (re-)implement caching

Changing Linux [Leis et al., SIGMOD'23]

Virtual memory-assisted buffer management (vmcache):

- DBMS is in control over faulting and eviction
- Exploit MMU and TLB
- Can be implemented with vanilla Linux, but:
  - Linux virtual memory primitives do not scale
- exmap: Linux kernel module that provides high-performance

Changing Linux [Leis et al., SIGMOD'23]

Virtual memory-assisted buffer management (vmcache):

- DBMS is in control over faulting and eviction
- Exploit MMU and TLB
- Can be implemented with vanilla Linux, but:
  - Linux virtual memory primitives do not scale
- exmap: Linux kernel module that provides high-performance

Downsides:

- Changing/extending Linux is very hard
- Maintainability

Case Study 2: I/O

OS Overhead for I/O: Random 4KB Read Microbenchmark

AMD 96-core EPYC 9654P CPU with 8 Kioxia CM7-R SSDs:

	M IOPS	k cycles per IO	CPU Util. (Cores)
pread	0.8	87.2	19.4
− page cache (O_DIRECT)	6.1	48.1	81.5
io_uring	6.7	28.4	52.9
− file system (XFS)	21.6	22.4	134.4
− RAID 0 (md)	23.3	19.0	123.0
+ uring opt. (polling, passthru)	23.3	9.7	62.8

OS Overhead for I/O: Random 4KB Read Microbenchmark

AMD 96-core EPYC 9654P CPU with 8 Kioxia CM7-R SSDs:

	M IOPS	k cycles per IO	CPU Util. (Cores)
pread	0.8	87.2	19.4
− page cache (O_DIRECT)	6.1	48.1	81.5
io_uring	6.7	28.4	52.9
− file system (XFS)	21.6	22.4	134.4
− RAID 0 (md)	23.3	19.0	123.0
+ uring opt. (polling, passthru)	23.3	9.7	62.8

⇒ To exploit fast SSDs, one has to disable all OS features + large overhead

Kernel Bypassing

- Directly access PCIe device
- Intel DPDK: networking
- Intel SPDK: NVMe storage

	M IOPS	k cycles per IO	CPU Util. (Cores)
+ uring opt. (polling, passthru)	23.3	9.7	62.8
SPDK (user-space I/O)	23.3	1.6	10.4

Kernel Bypassing

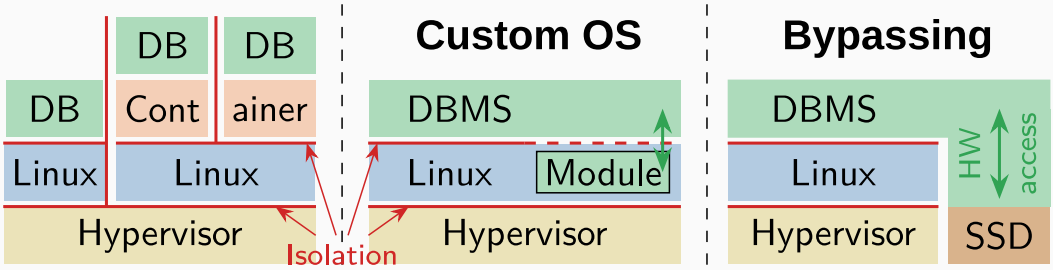
- Directly access PCIe device
- Intel DPDK: networking
- Intel SPDK: NVMe storage

	M IOPS	k cycles per IO	CPU Util. (Cores)
+ uring opt. (polling, passthru)	23.3	9.7	62.8
SPDK (user-space I/O)	23.3	1.6	10.4

Downsides:

- Very low level, we're doing the job of the OS
- Memory allocation issues
- No OS features (e.g., TCP, file system)

Models of DBMS/OS Interaction



- overhead

- difficult
- maintainability

- OS features
- priviliged instr.

Linux

- Experience in exploiting modern hardware with all three approaches
- Lots of engineering effort (customized Linux, bypassing)
- Or bad performance (vanilla Linux)
- This is despite Linux being a marvel of technology
- But Linux is impossible to fix given its constraints:
  - POSIX, obscure features, backward compatibility
  - Support for obsolete hardware platforms
  - Process isolation

Taking A Step Back

- I argue that the root of the problem is legacy OS interfaces (POSIX)
- Have been designed decades ago for a very different hardware landscape
- Consequences:
  - complexity
  - inefficiency
  - bugs
- fsyncgate Linux/PostgreSQL anecdote: `https://danluu.com/fsyncgate/`

Can we do better?

Unikernel + Cloud DBMS = A Perfect Match?

Unikernel:

- Single-address operating systems: threads, but no processes
- *Everything* runs in kernel space: privileged instructions possible, no syscall overhead, direct access to hardware

Unikernel + Cloud DBMS = A Perfect Match?

Unikernel:

- Single-address operating systems: threads, but no processes
- *Everything* runs in kernel space: privileged instructions possible, no syscall overhead, direct access to hardware

Cloud DBMS:

- DBaaS: users don't care about OS kernel
- Virtual machine runs only the database process

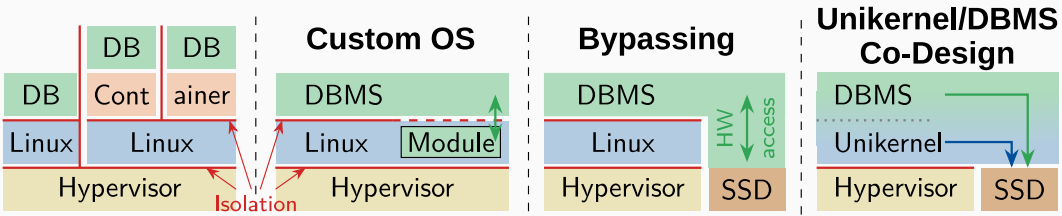
Unikernel + Cloud DBMS = A Perfect Match?

Unikernel:

- Single-address operating systems: threads, but no processes
- *Everything* runs in kernel space: privileged instructions possible, no syscall overhead, direct access to hardware

Cloud DBMS:

- DBaaS: users don't care about OS kernel
- Virtual machine runs only the database process



What About Security?

- Hypervisor provides security isolation across virtual machines
- Every virtual machine runs one (database) service
- In this world, Linux process isolation has little additional benefit
- Bug in DBMS will expose user data anyway
- Unikernels have a smaller attack surface
  - Anecdote: Google disabled uring in production:
    `https://security.googleblog.com/2023/06/learnings-from-kctf-vrps-42-linux.html`

- Implements much (but not all of) POSIX
- Most code runs out-of-the-box or is easy to port
- Supports x86 and ARM multi-core CPUs
- Boot through kvm/QEMU, debug using gdb
- Simple, clean C++ code
  (virtual memory implementation in OSv: 2k lines; Linux: 110k lines)
- No process isolation and no kernel/user space split make code much easier

OSv Deployment

- Can directly boot in AWS EC2 virtual machine or metal instance (in <100ms)
- Storage: NVMe driver
- Networking: ENA driver
- And that's it
- Can also run on top of Firecracker

Unikernel/DBMS Co-Design [Leis and Dietrich, VLDB'24]

- Migrating to OSv will not magically make your DBMS faster
- Will often be slower out-of-the-box because Linux is well engineered

- Migrating to OSv will not magically make your DBMS faster
- Will often be slower out-of-the-box because Linux is well engineered

New Powers

- CPU:
  - Preemption: timer, IPI
  - Sleep/power states

New Powers

- CPU:
  - Preemption: timer, IPI
  - Sleep/power states
- Virtual memory:
  - Page table: 4-level radix tree
  - TLB flush instructions

New Powers

- CPU:
  - Preemption: timer, IPI
  - Sleep/power states
- Virtual memory:
  - Page table: 4-level radix tree
  - TLB flush instructions
- NVMe:
  - Completion queue: 64 byte entry
  - Submission queue: 16 byte entry
  - Enable/disable interrupts

New Powers

- CPU:
  - Preemption: timer, IPI
  - Sleep/power states
- Virtual memory:
  - Page table: 4-level radix tree
  - TLB flush instructions
- NVMe:
  - Completion queue: 64 byte entry
  - Submission queue: 16 byte entry
  - Enable/disable interrupts
- Hypervisor:
  - Memory ballooning
  - CPU hot-plugging

Opportunities

Only makes sense for data plane:

- Virtual memory-supported caching
- Virtual memory-supported memory allocation
- Virtual memory-supported snapshotting
- Unified thread/task scheduling with preemption
- High-performance storage and network I/O: <100 cycles per I/O?
- Easier and faster resource overprovisioning in multi-tenant settings

Conclusions

- This a personal story of radicalization after fighting against the OS for years
- Hardware is usually much simpler than legacy OS stacks
- Existing operating systems makes exploiting modern hardware
  - complex
  - inefficiency
  - bug-prone
- Unikernels and DBaaS in the cloud may make custom OS kernels realistic
  - Enables *faster and simpler* data plane systems
- Dropping Linux sounds radical, but it's the best path forward
  - eBPF and uring are only band-aids

`https://www.cs.cit.tum.de/dis/research/cumulus/`