Eggstrain

Vectorized Push-Based inspired Execution Engine

Asynchronous Buffer Pool Manager

Authors: Connor, Sarvesh, Kyle

Original Proposed Goals

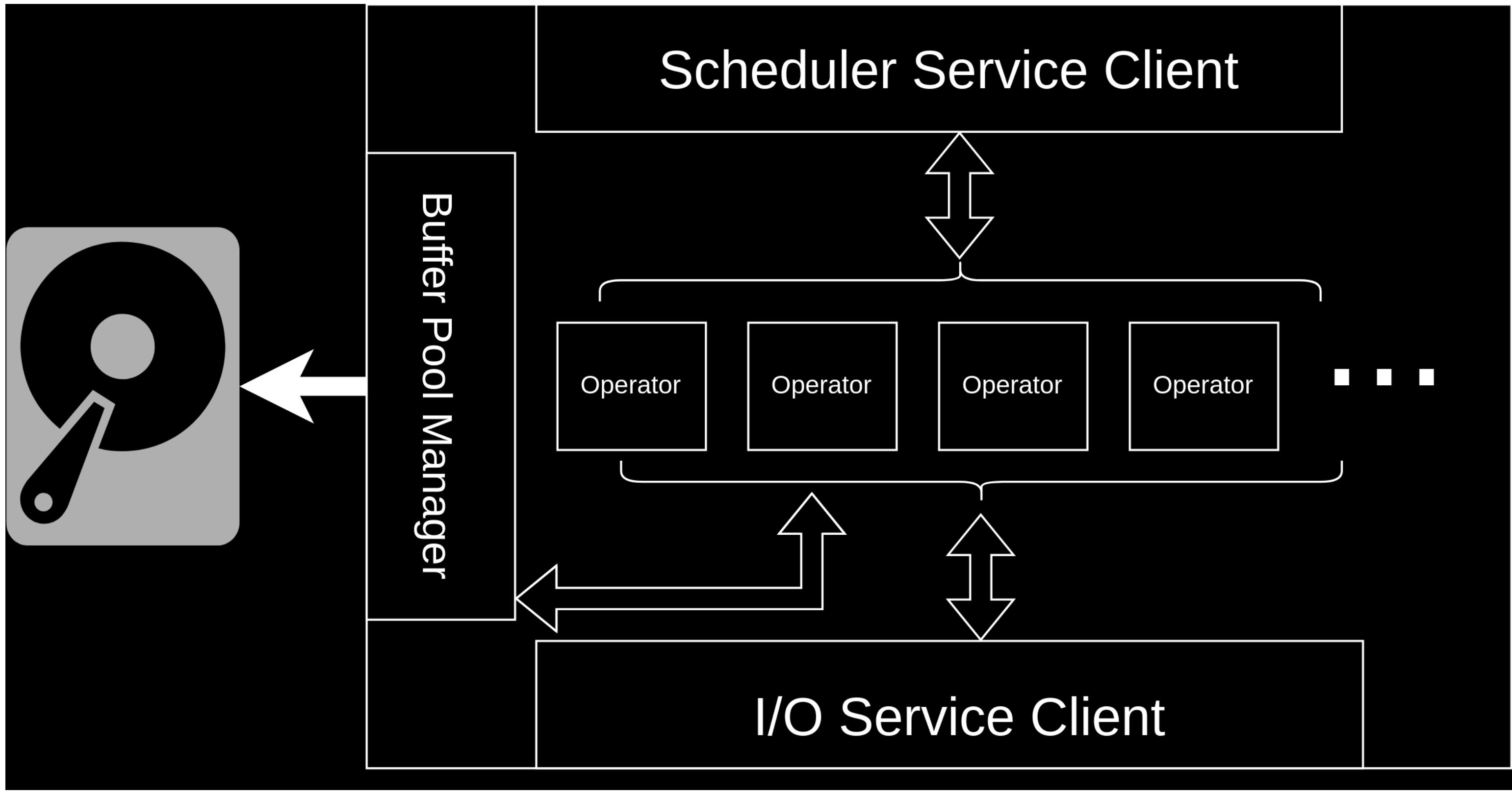
- 75%: First 7 operators working + integration with other components
- 100%: All operators listed above working
- 125%: TPC-H benchmark working

Design Goals

- Robustness
- Modularity
- Extensibility
- Forward Compatibility

We made heavy use of `tokio` and `rayon` in our implementation.

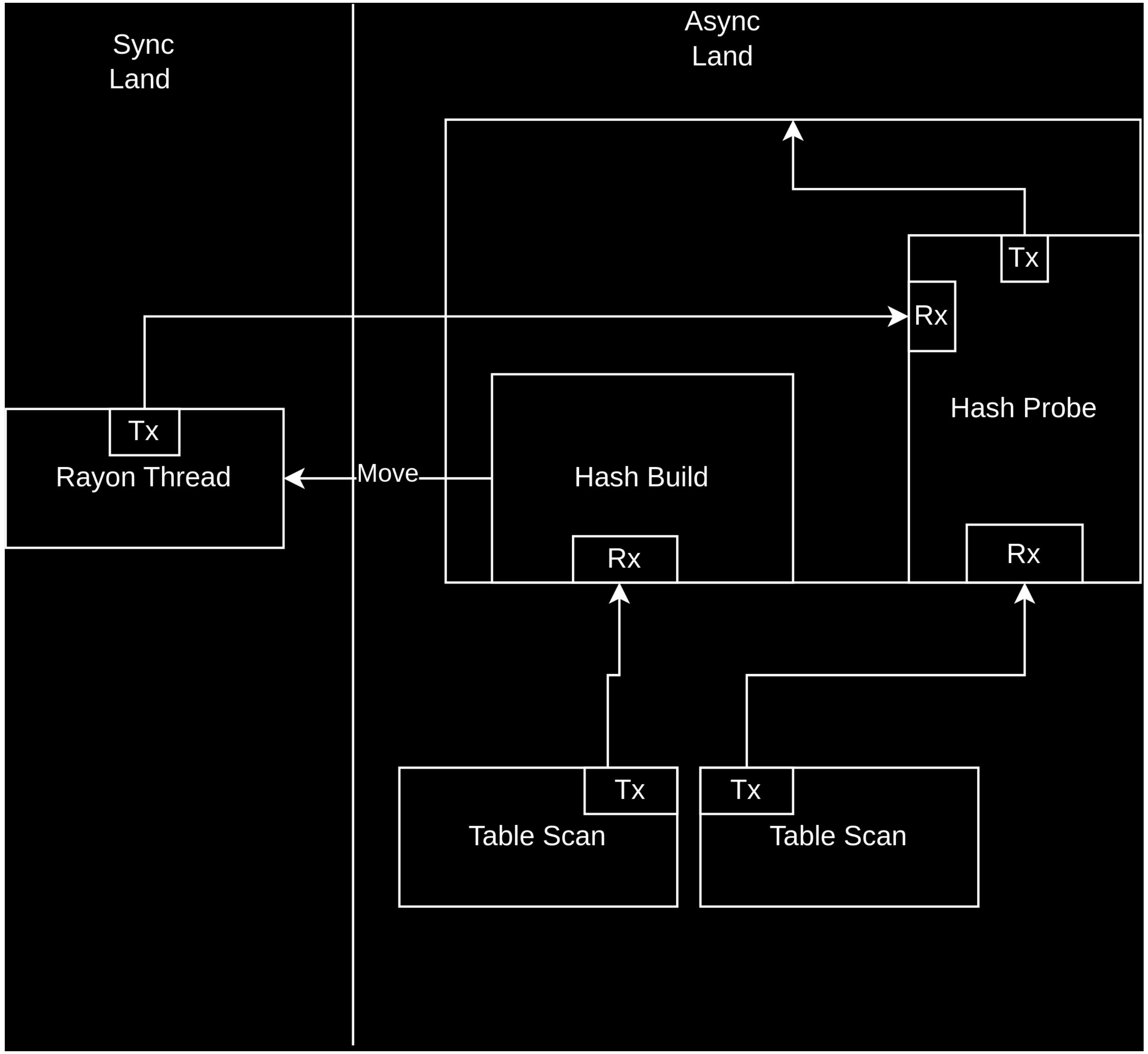
Refresher on Architecture



Refresher on operators

- `TableScan`
- `Filter`
- `Projection`
- `HashAggregation`
- `HashJoin` (`HashProbe` + `HashBuild`)
- `OrderBy`
- `TopN`

Example Operator Workflow



Progress Towards Goals

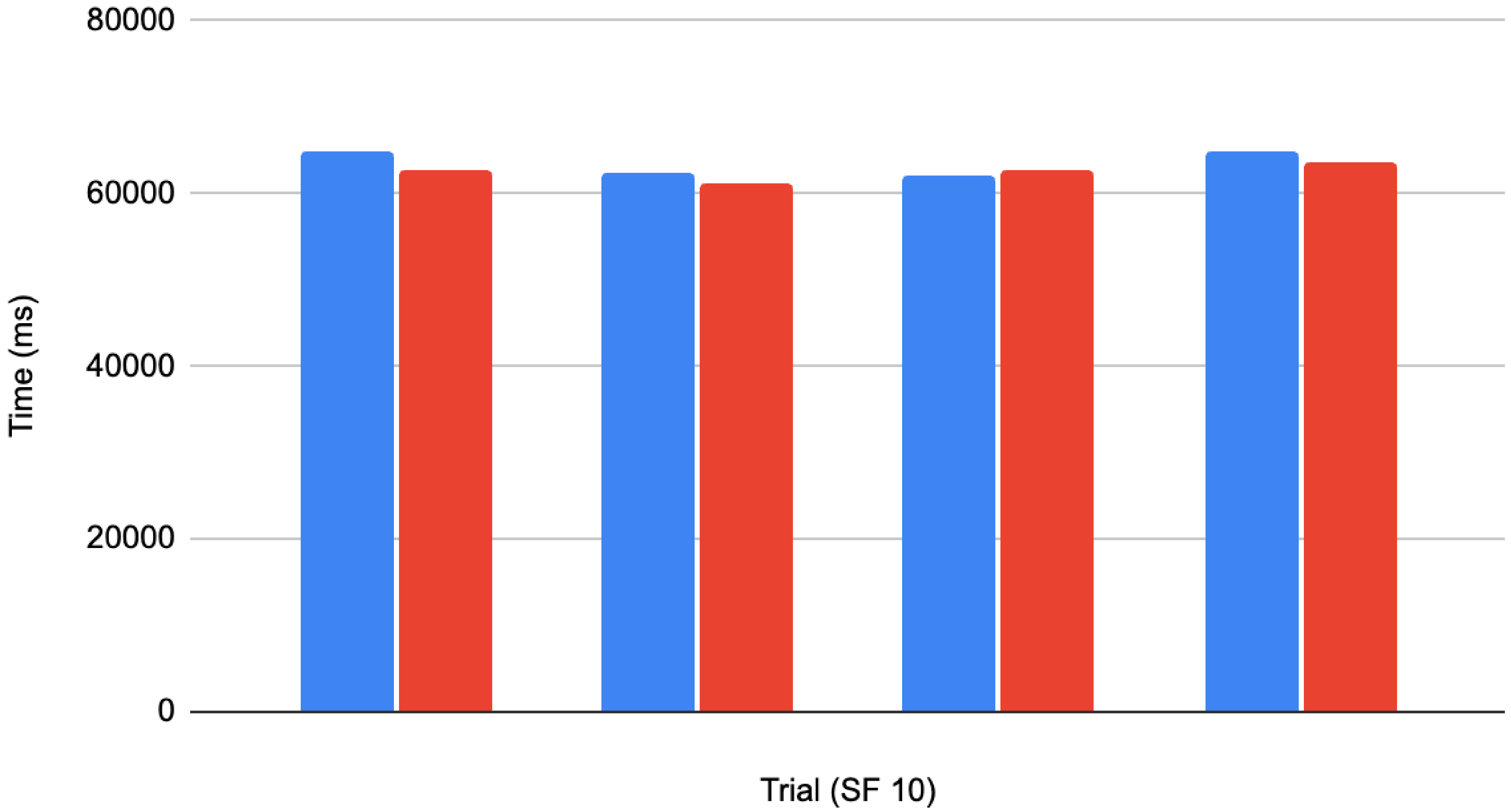
- 100%: All operators implemented, excluding `HashJoin`
- 125%: TPC-H benchmark working for Q1

Execution Engine Benchmarks

Hardware:

- M1 Pro, 8 cores, 16GB RAM

Datafusion vs Eggstrain TPC-H Q1



Correctness Testing and Code Quality Assessment

We tested correctness by comparing our results to the results of the same queries run in DataFusion.

Our code quality is high with respect to documentation, integration tests, and code review.

However, we lack unit tests for each operator. We instead tested operators integrated inside of queries.

Problem: In Memory?

We found that we needed to spill data to disk to handle large queries.

However, to take advantage of our asynchronous architecture, we needed to implement an **asynchronous buffer pool manager.**

Recap: Buffer Pool Manager

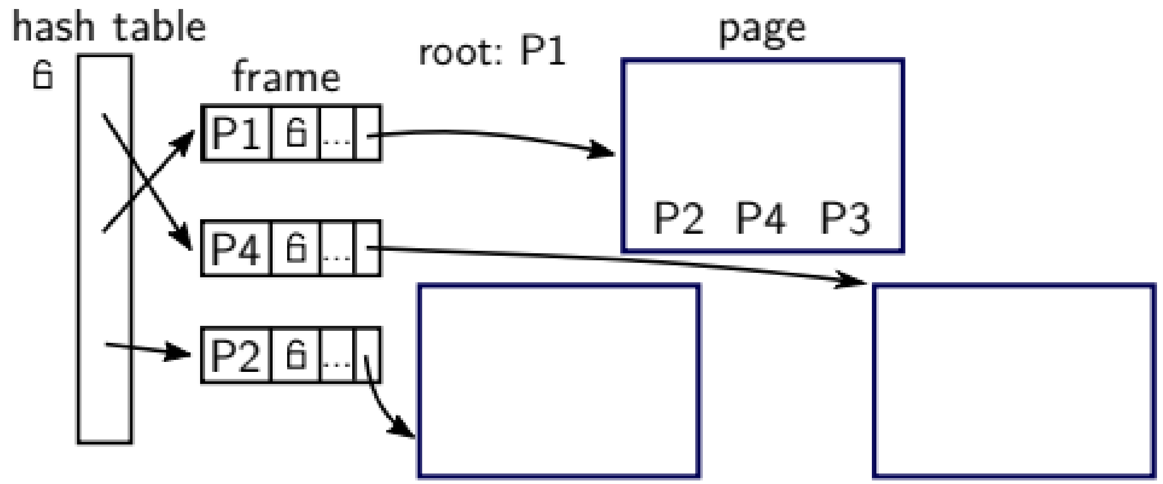
A buffer pool manager manages synchronizing data between volatile memory and persistent storage.

- In charge of bringing data from storage into memory in the form of pages
- In charge of synchronizing reads and writes to the memory-local page data
- In charge of writing data back out to disk so it is synchronized

Traditional Buffer Pool Manager

Traditional BPMs will use a global hash table that maps page IDs to memory frames.

- Source: *LeanStore: In-Memory Data Management Beyond Main Memory (2018)*



(a) traditional buffer manager

Recap: Blocking I/O

Additionally, traditional buffer pool managers will use blocking reads and writes to send data between memory and persistent storage.

Blocking I/O is heavily reliant on the Operating System.

> The DBMS can almost always manage memory better than the OS

- Source: 15-445 Lecture 6 on Buffer Pools

Recap: I/O System Calls

What happens when we issue a `pread()` or `pwrite()` call?

- We stop what we're doing
- We transfer control to the kernel
- *We are blocked waiting for the kernel to finish and transfer control back*
  - *A read from disk is probably scheduled somewhere*
  - *Something gets copied into the kernel*
  - *The kernel copies that something into userspace*
- We come back and resume execution

Blocking I/O for Buffer Pool Managers

Blocking I/O is fine for most situations, but might be a bottleneck for a DBMS's Buffer Pool Manager.

- Typically optimizations are implemented to offset the cost of blocking:
    - Pre-fetching
    - Scan-sharing
    - Background writing
    - `O_DIRECT`

Non-blocking I/O

What if we could do I/O *without* blocking? There exist a few ways to do this:

- `libaio`
- `io_uring`
- SPDK
- All of these allow for *asynchronous I/O*

`io_uring`

This Buffer Pool Manager is going to be built with asynchronous I/O using `io_uring`.

- Source: *What Modern NVMe Storage Can Do, And How To Exploit It... (2023)*

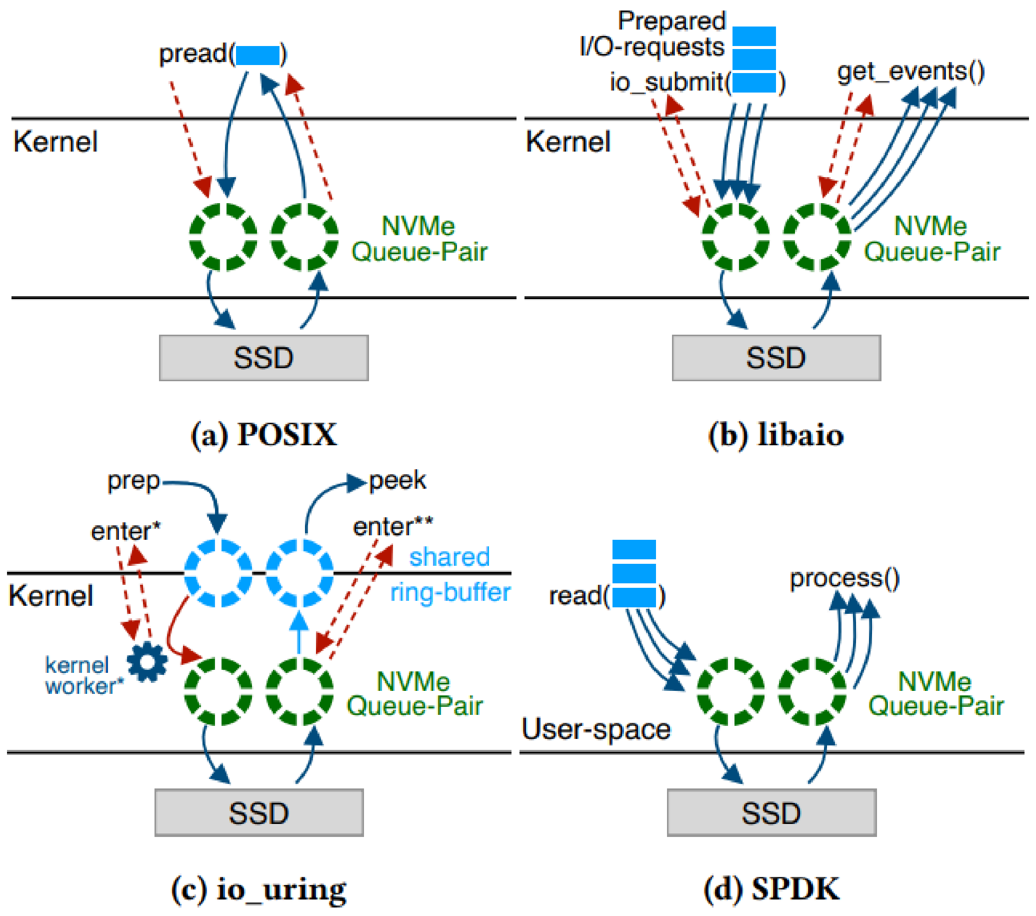


Figure 5: Comparison of Linux storage I/O interfaces.

Asynchronous I/O

Asynchronous I/O really only works when the programs running on top of it implement *cooperative multitasking*.

- Normally, the kernel gets to decide what thread gets to run
- Cooperative multitasking allows the program to decide who gets to run
- Context switching between tasks is a *much more* lightweight maneuver
- If one task is waiting for I/O, we can cheaply switch to a different task!

Eggstrain

The key thing here is that our Execution Engine `eggstrain` fully embraces asynchronous execution.

- Rust has first-class support for asynchronous programs
- Using `async` libraries is almost as simple as plug-and-play
- The `tokio` crate is an easy runtime to get set up
- We can easily create a buffer pool manager in the form of a Rust library crate

Goals

The goal of this system is to *fully exploit parallelism*.

- NVMe drives have gotten really, really fast
- Blocking I/O simply cannot match the full throughput of an NVMe drive
- They are *completely* bottle-necked by today's software
- If we can fully exploit parallelism in software *and* hardware...
    - **We can actually get close to matching the speed of in-memory systems, *while using persistent storage***

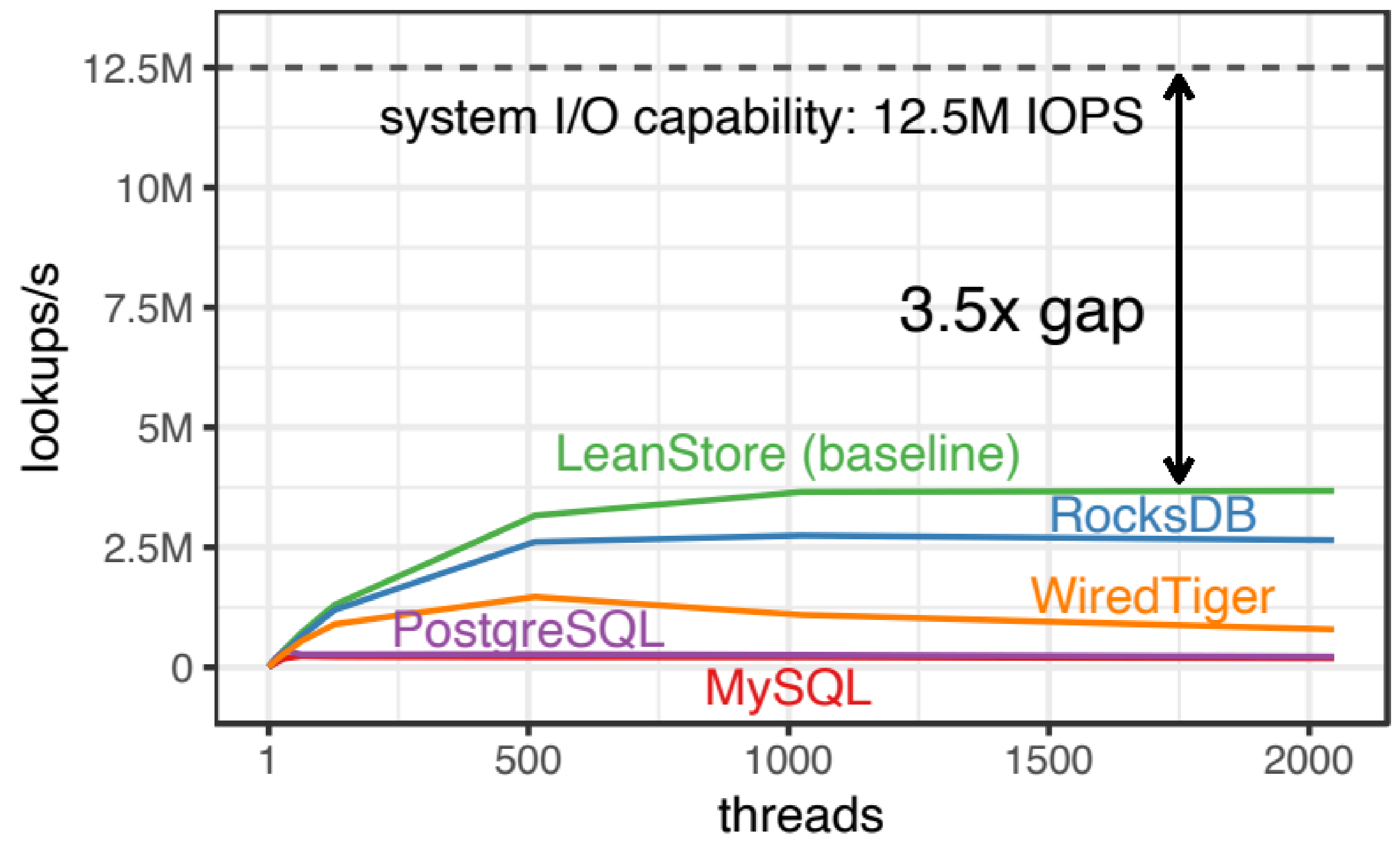
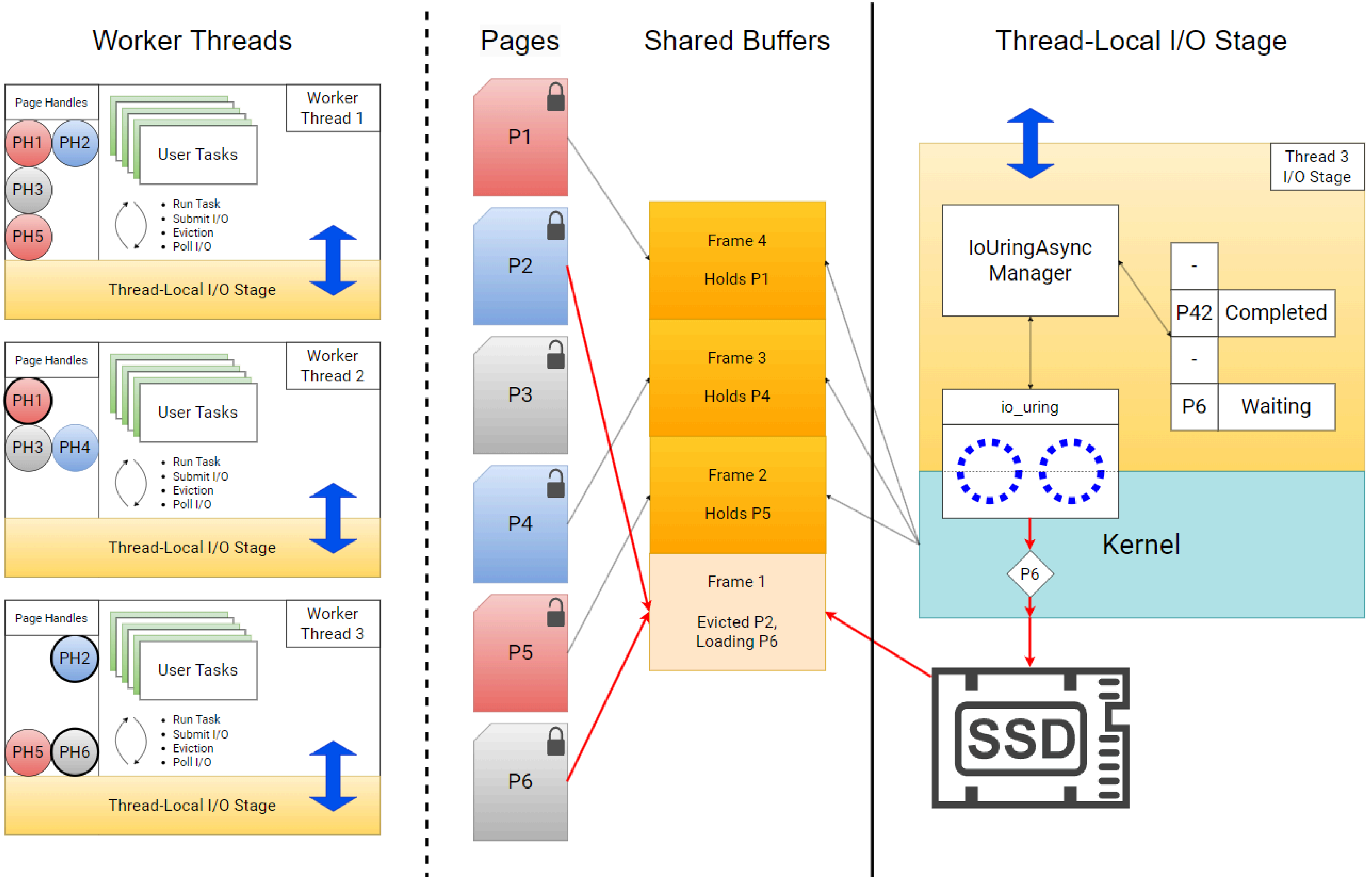


Figure 1: Out-of-memory performance for random lookups in 100 GB database with 10 GB buffer pool and 8 enterprise-grade SSDs.

Proposed Design

The next slide has a proposed design for a fully asynchronous buffer pool manager. The full (somewhat incomplete) writeup can be found here.

- Heavily inspired by LeanStore
  - Eliminates the global page table and uses tagged pointers to data
- Even more inspired by this paper:
  - *What Modern NVMe Storage Can Do, And How To Exploit It: High-Performance I/O for High-Performance Storage Engines (2023)*
    - Gabriel Haas and Viktor Leis
- The goal is to *eliminate as many sources of global contention as possible*



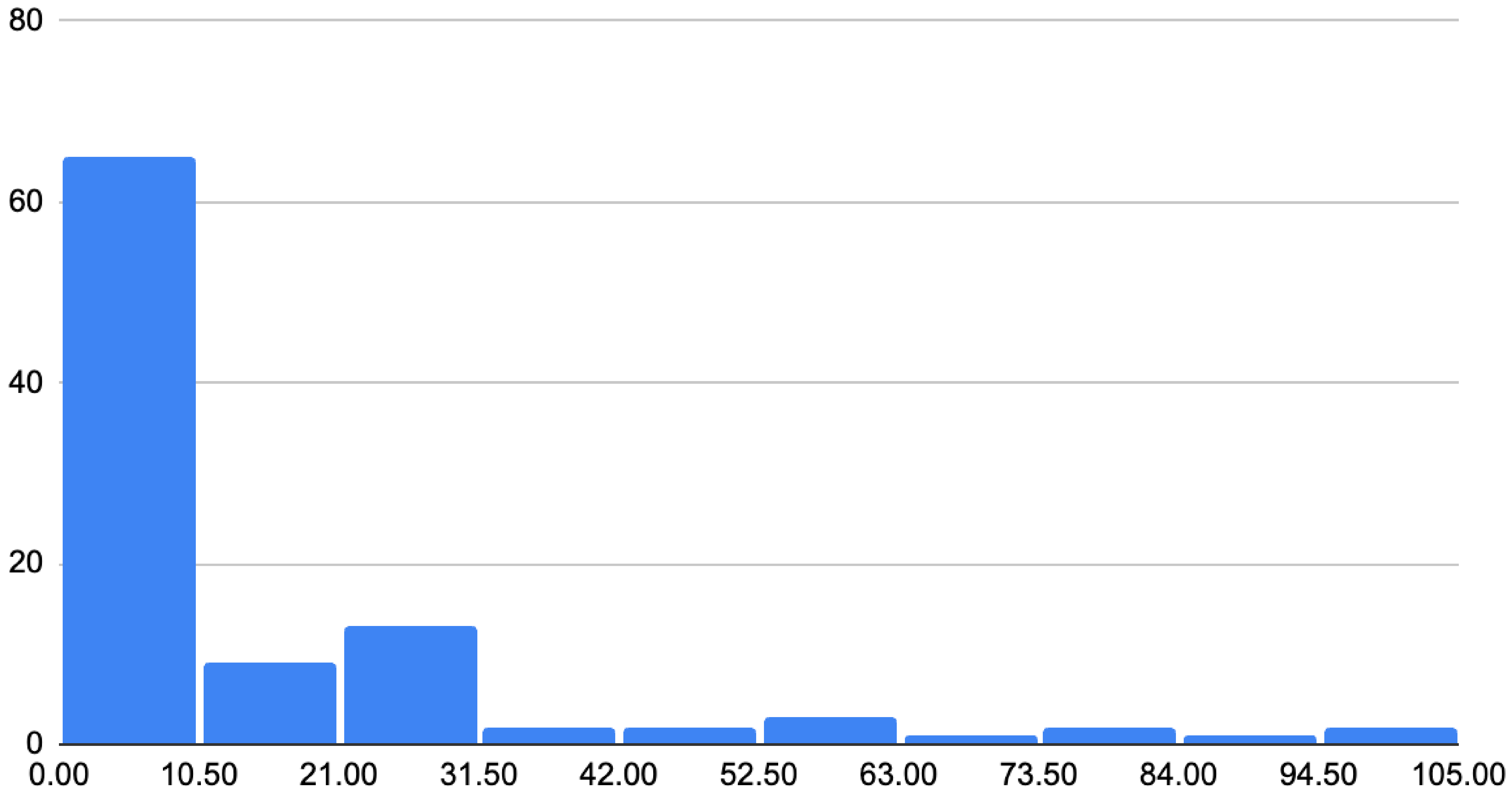
Worker Threads
Pages
Shared Buffers
Thread-Local I/O Stage
Page Handles
PH1
PH2
PH3
PH5
User Tasks
Worker Thread 1
Run Task
Submit I/O
Eviction
Poll I/O
Thread-Local I/O Stage
Page Handles
PH1
PH3
PH4
User Tasks
Worker Thread 2
Run Task
Submit I/O
Eviction
Poll I/O
Thread-Local I/O Stage
Page Handles
PH2
PH5
PH6
User Tasks
Worker Thread 3
Run Task
Submit I/O
Eviction
Poll I/O
Thread-Local I/O Stage
P1
P2
P3
P4
P5
P6
Frame 4
Holds P1
Frame 3
Holds P4
Frame 2
Holds P5
Frame 1
Evicted P2, Loading P6
Thread 3 I/O Stage
IoUringAsync Manager
-
P42 Completed
-
P6 Waiting
io_uring
Kernel
P6
SSD

BPM Benchmarks

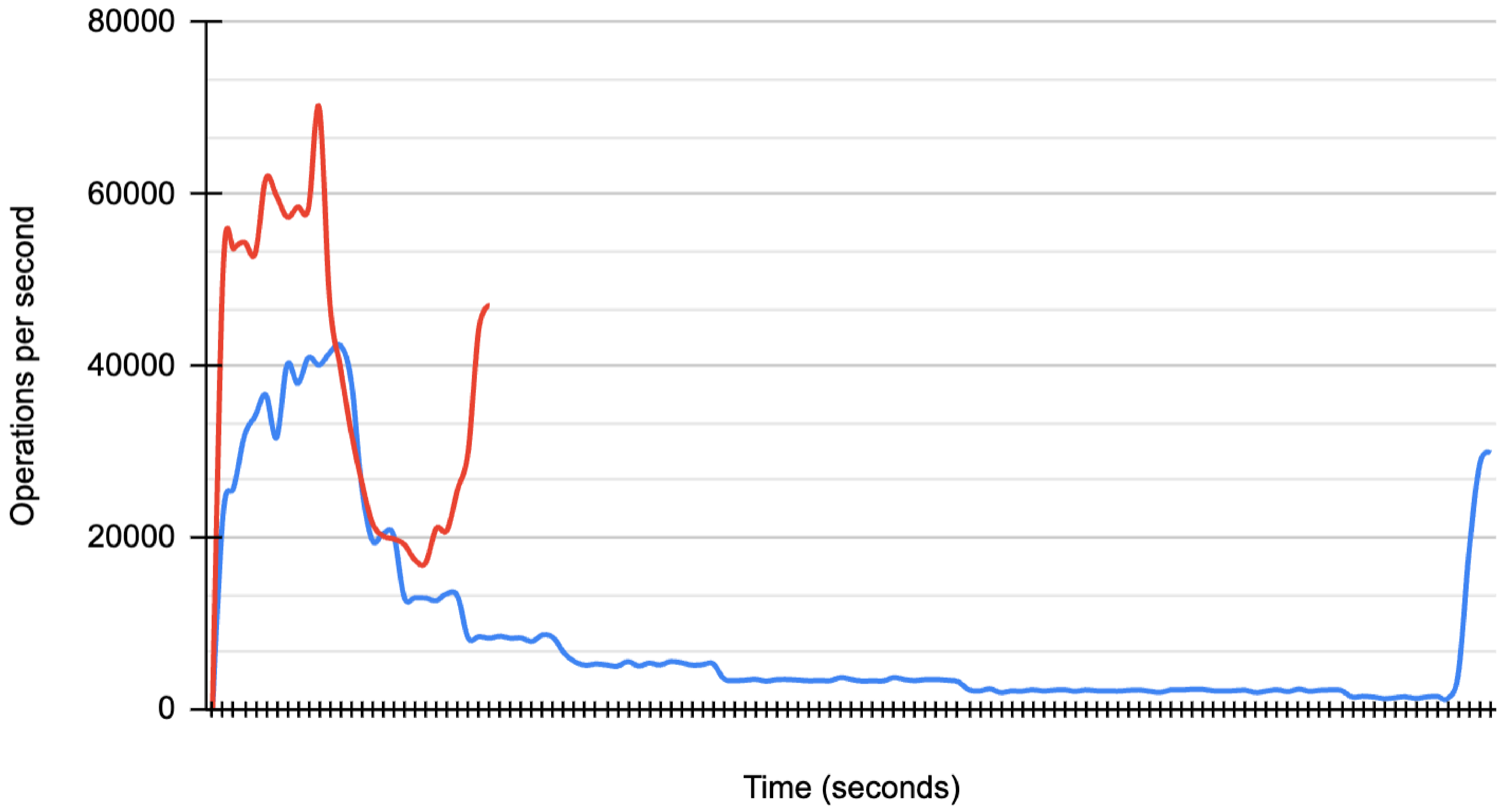
Hardware:

- Cray/Appro GB512X - 32 Threads Xeon E5-2670 @ 2.60GHz, 64 GiB DDR3 RAM, 1x 240GB SSD, Gigabit Ethernet, QLogic QDR Infiniband
- We will benchmark against RocksDB as a buffer pool manager

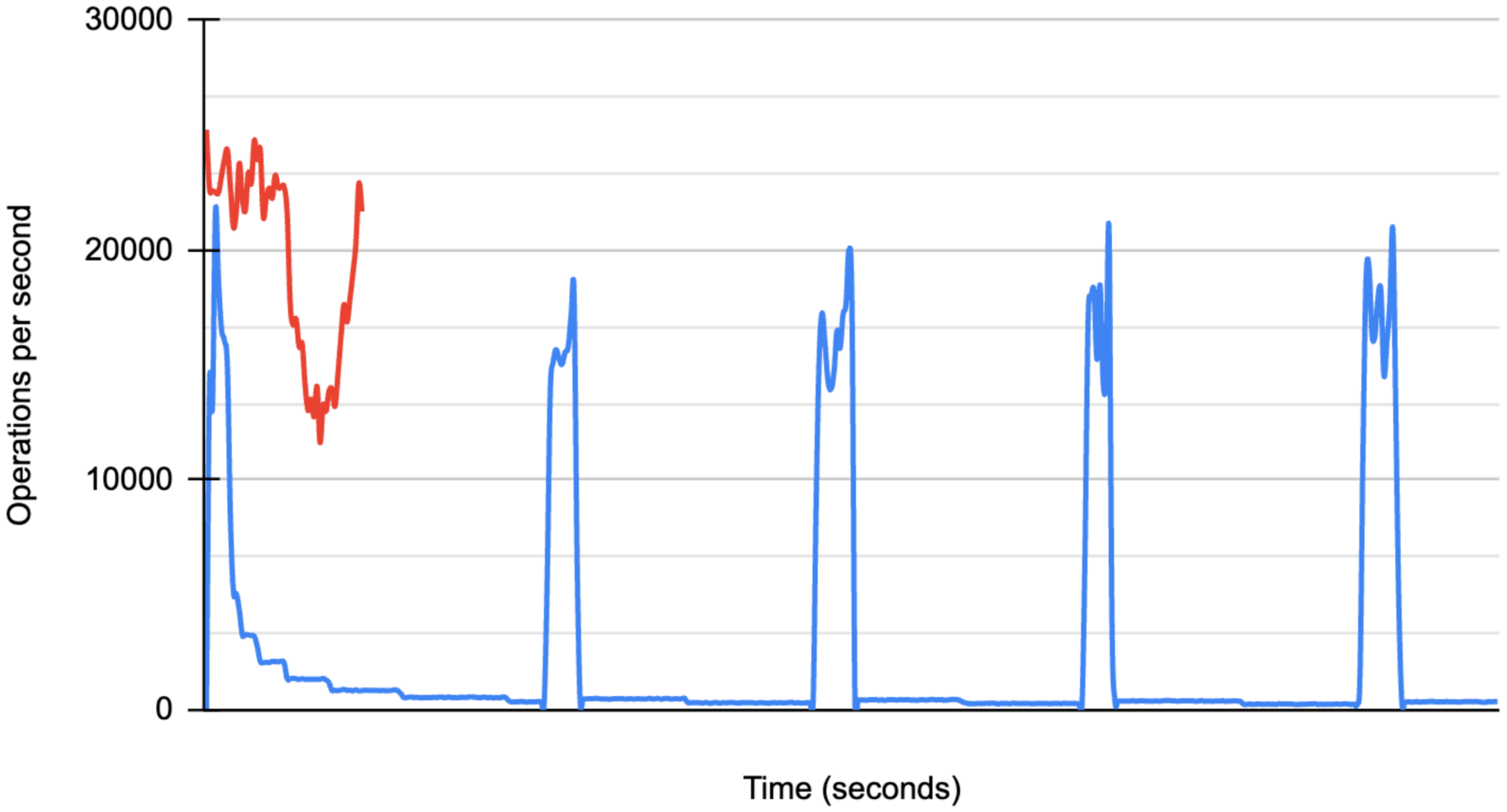
Zipfian 1.1


80
60
40
20
0
0.00
10.50
21.00
31.50
42.00
52.50
63.00
73.50
84.00
94.50
105.00

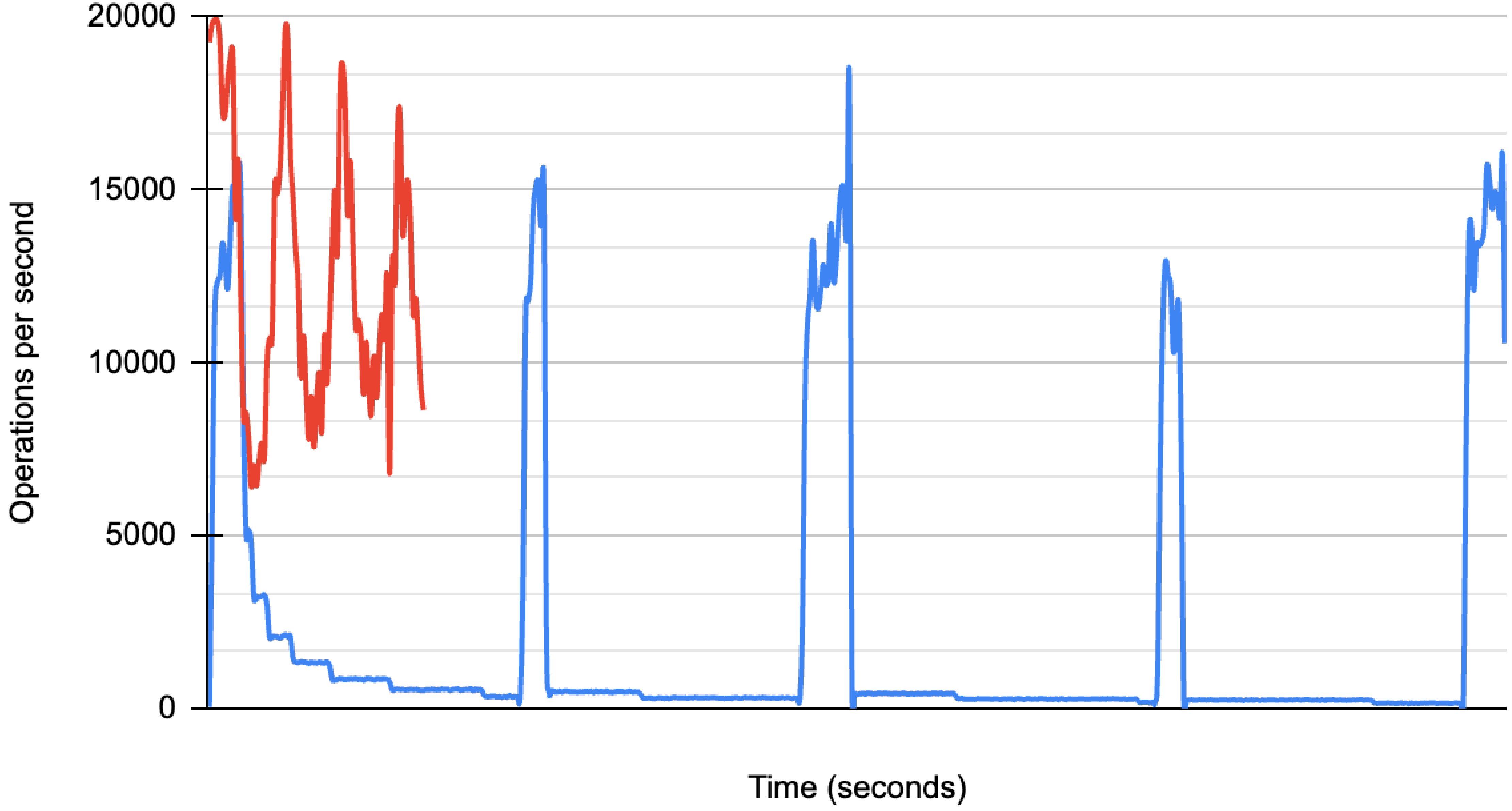
EggstrainBPM vs RocksDB on 20% write / 80% read



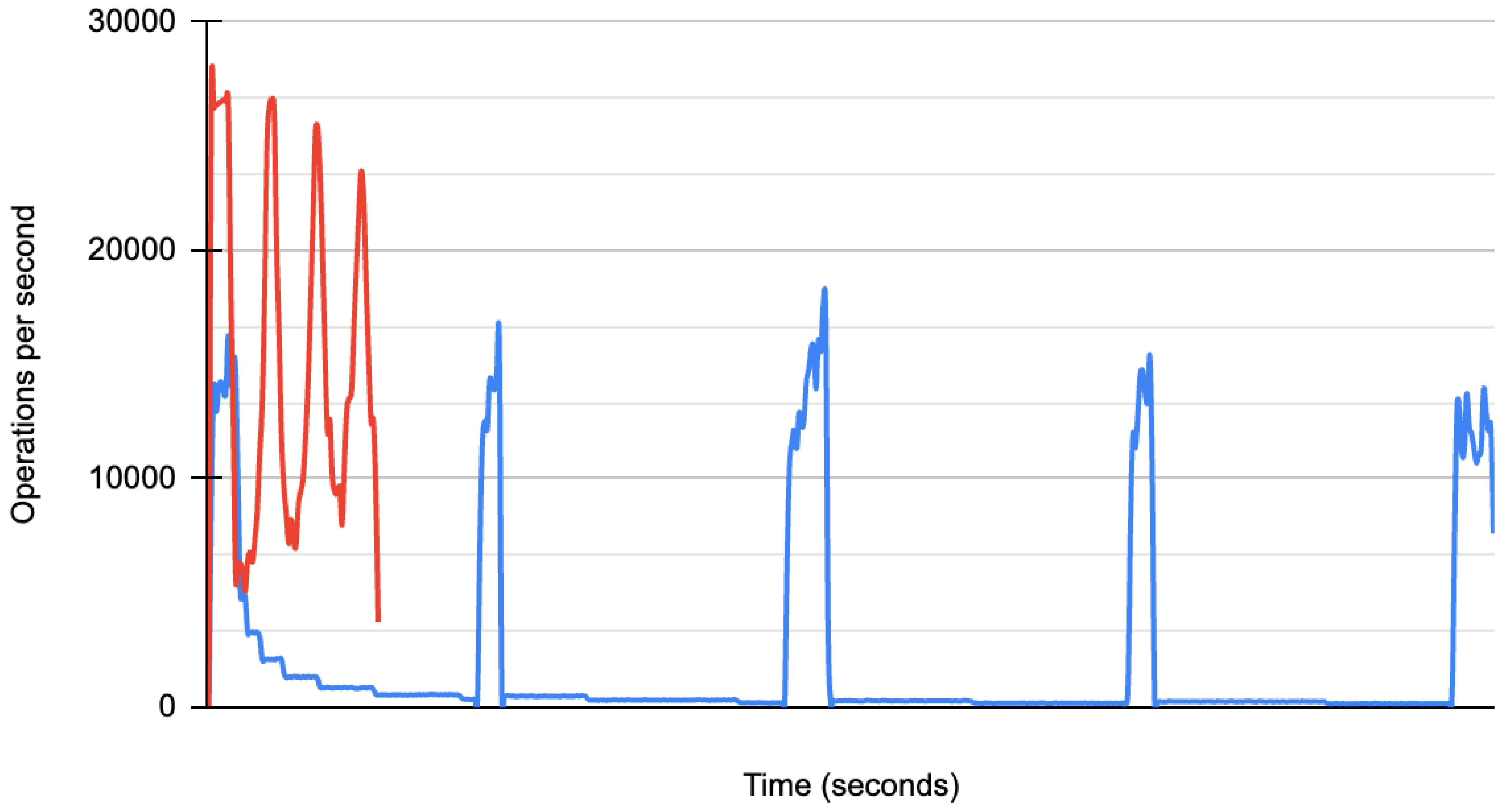
EggstrainBPM vs RocksDB on 80% write / 20% read



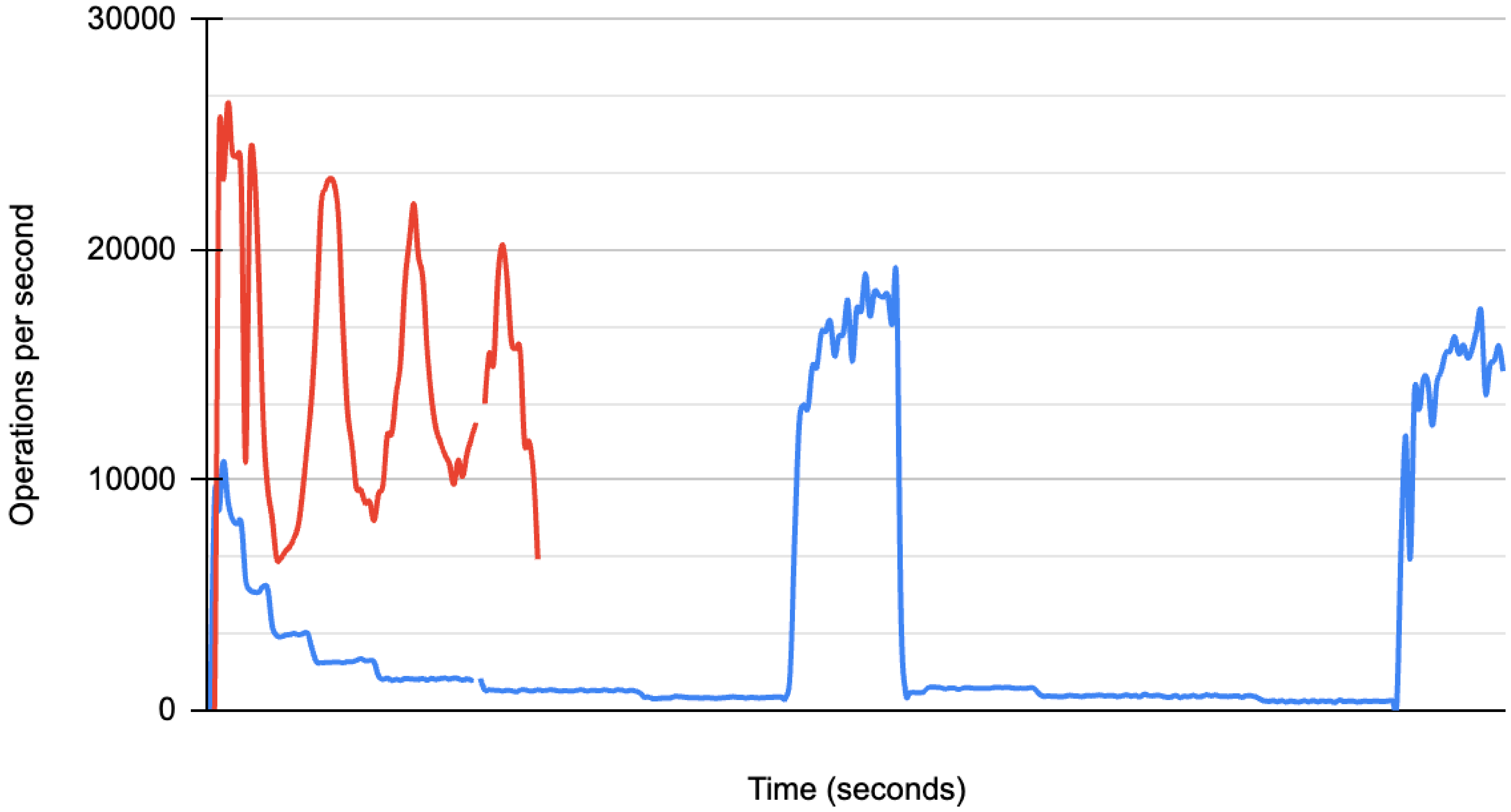
EggstrainBPM vs RocksDB on 80% write / 20% read (Uniform)



EggstrainBPM vs RocksDB on 20% write / 80% read (Uniform)



EggstrainBPM vs RocksDB on 50% write / 50% read (Uniform)



Future Work

- Asynchronous BPM ergonomics and API
- Proper `io_uring` polling and batch evictions
- Shared user/kernel buffers and file descriptors (avoiding `memcpy`)
- Multiple NVMe SSD support (Software-implemented RAID 0)
- Optimistic hybrid latches