Performance Study of Buffer Pool Manager on Emulated SSDs

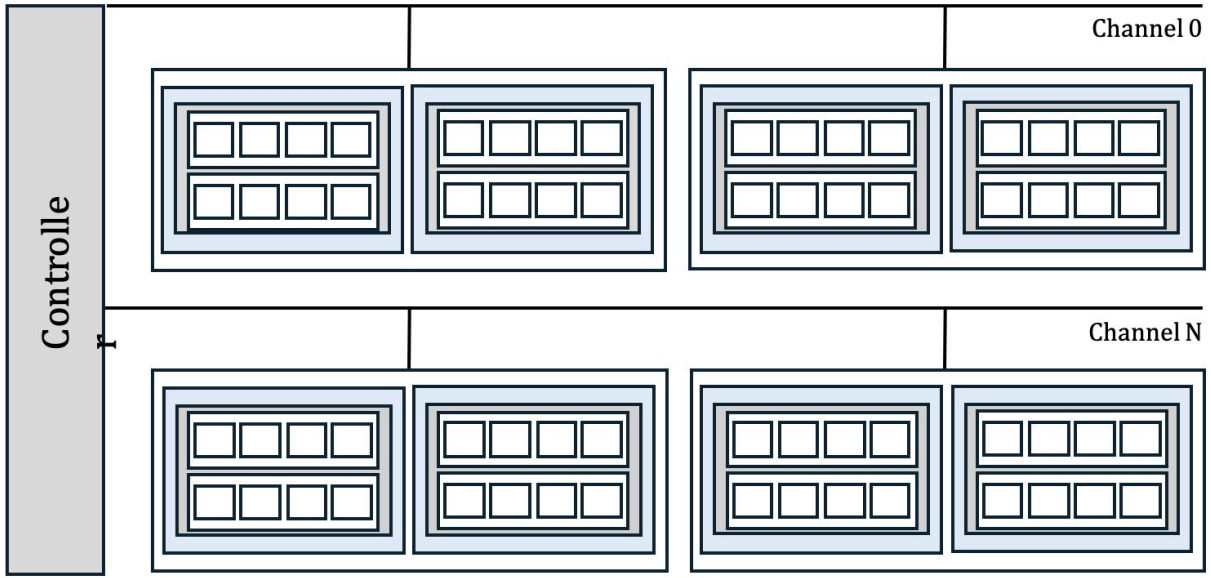
Can Gokmen, Toby Ueno, Tommy Ho

Background: SSDs

Faster read/write speeds and lower latency compared to HDDs.

Makes use of in internal parallelism architecture to concurrently read/write data to pages.

Stripes data across multiple planes allowing many parts of the SSD to function simultaneously.



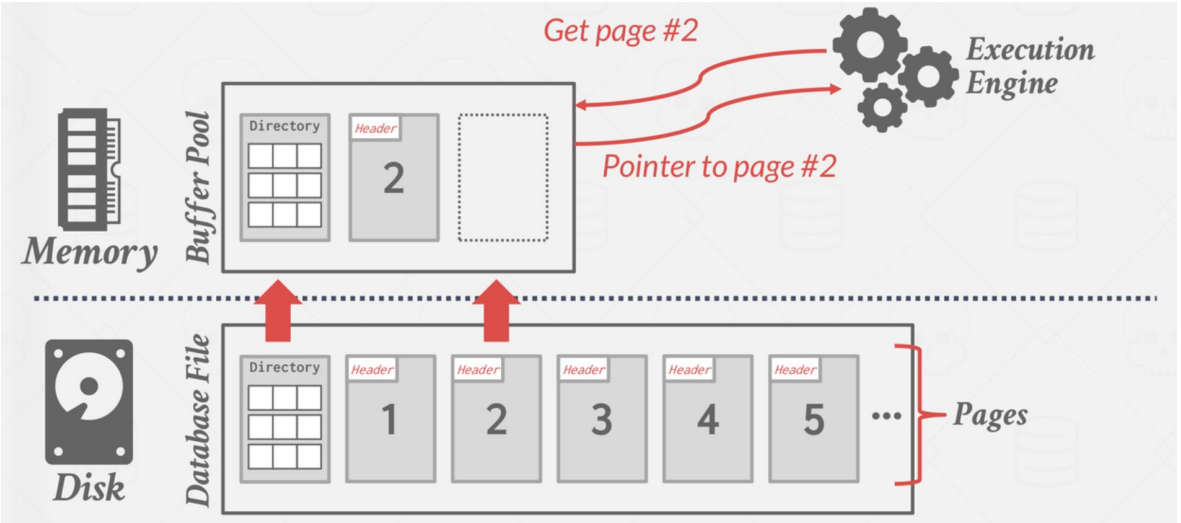
Flash Package -> Chip -> Die -> Plane -> Erase Block -> Page -> Nand Cells

Background: Bufferpools

Caches disk pages in memory to reduce disk access

Serves repeated requests from memory instead of re-reading from disk

Buffers dirty pages in memory, allowing delayed and more efficient write-backs to disk.



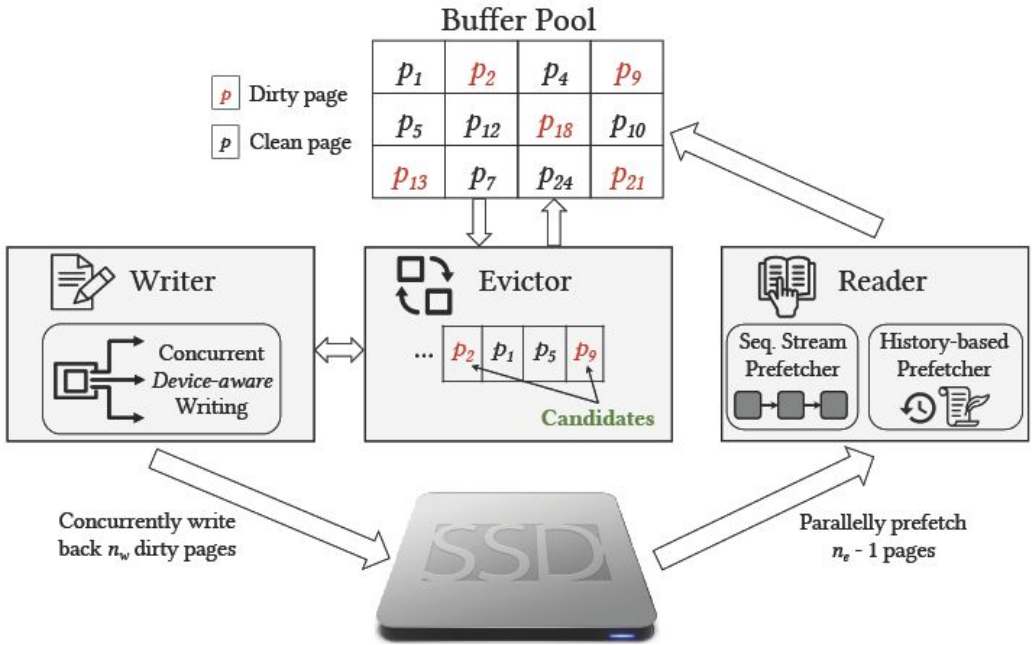
Background: ACE

Eviction Policies: **LRU, CFLRU, LRU-WSR**

Existing bufferpools assume no concurrency, meaning they retrieve pages and write back one at a time.

Takes into account read/write asymmetry allowing ACE to buffer dirty pages longer.

Sequentially evicts writes, taking advantage of concurrency making the cost of n writes equal to 1 write.



Problem: Testing

How to compare across SSD parameters? **(concurrency, asymmetry)**

Would require many physical SSDs!

Monetary limit

Can't set **arbitrary** parameters- have to rely on hardware

Project Goal: Examining ACE Performance on an SSD Emulator

FEMU: open-source NVMe SSD emulator (developed by Li et. al., FAST 2018)

Objective: setup, run, and observe ACE on FEMU

Motive 1: Confirm performance benefits of ACE on SSDs

Motive 2: Confirm viability of FEMU as a testing platform

Methodology

1. Install FEMU on SCC
2. Verify that FEMU parameters affect IOs
3. Install ACE on FEMU VM
4. Verify that ACE writes to the emulated SSD
5. Experiments
    a. Workload R/W ratio (ACE)
    b. Bufferpool size (ACE)
    c. Concurrency (ACE)
    d. Number of channels (FEMU)
    e. Write asymmetry (FEMU)

Challenges

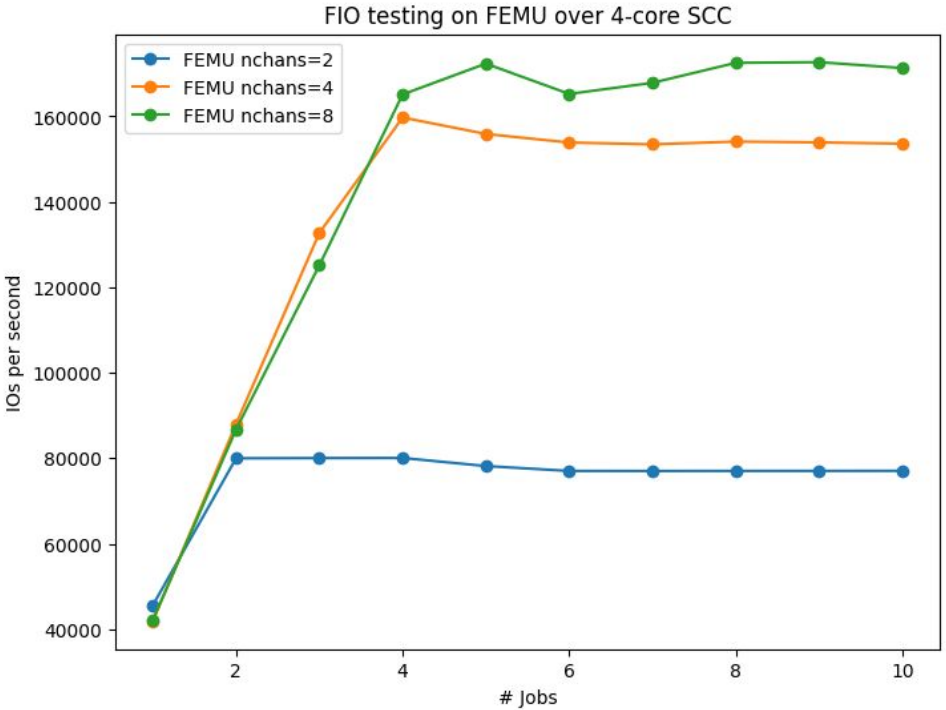
Installing FEMU

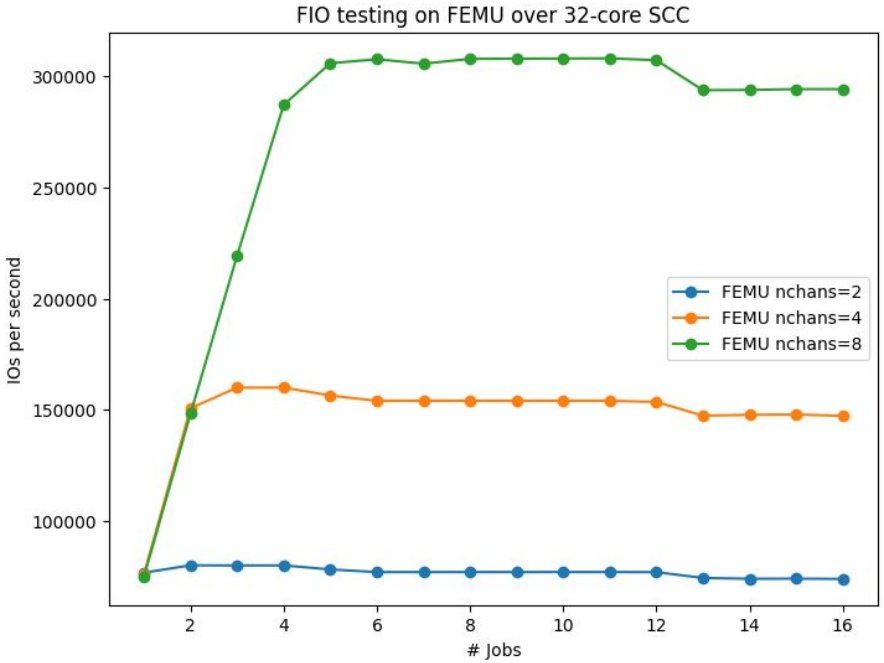
Understanding ACE parameters

Ensuring ACE wrote to the SSD

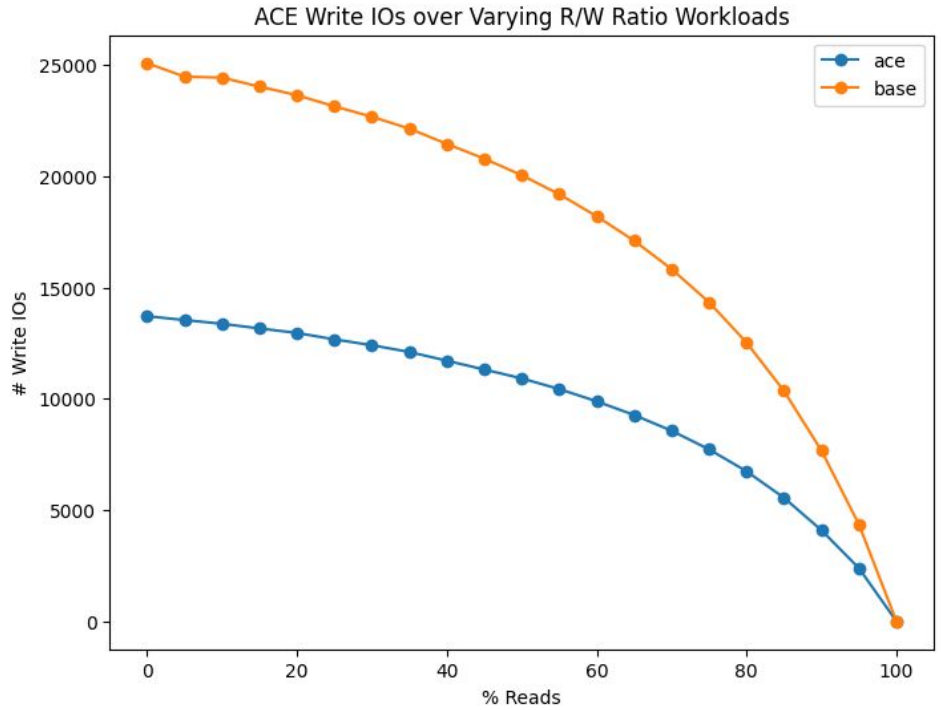
(a lot of rerunning experiments!)

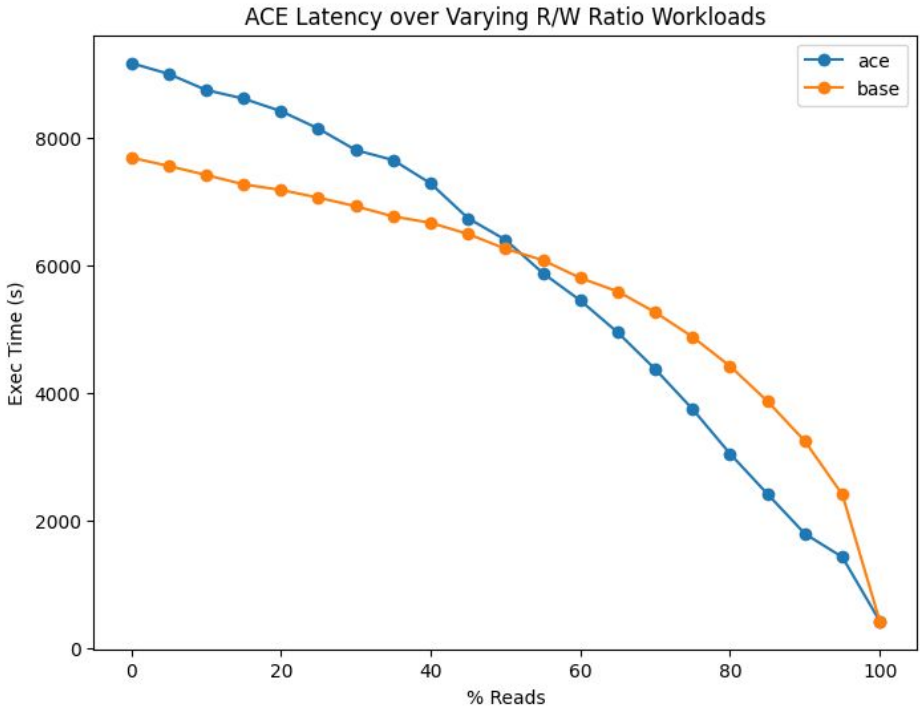
Results: FEMU Baseline





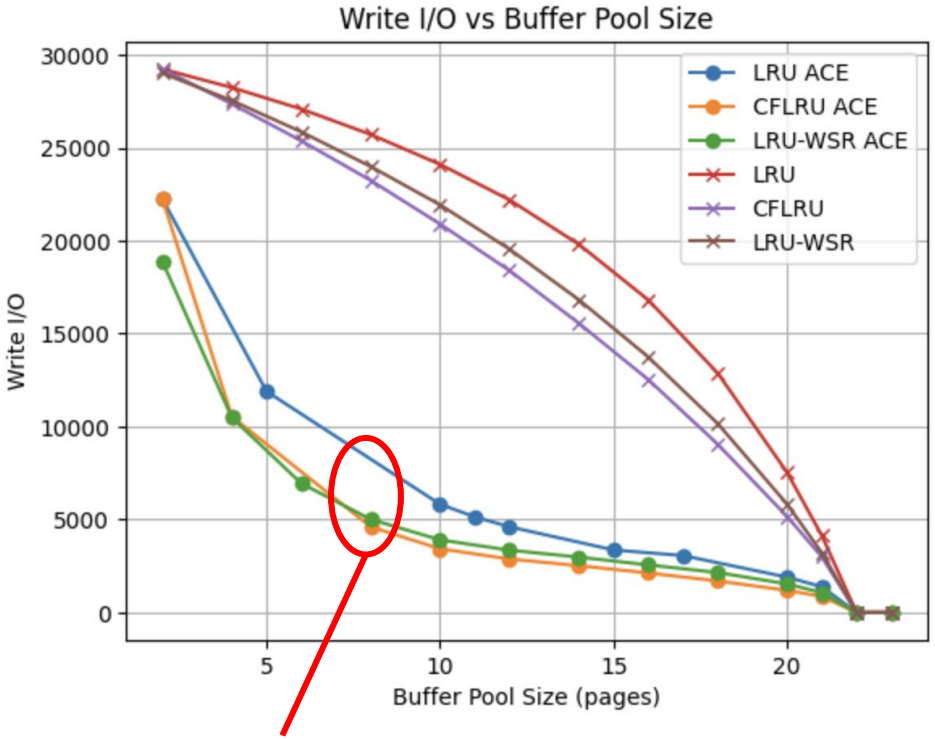
Results: R/W Ratio



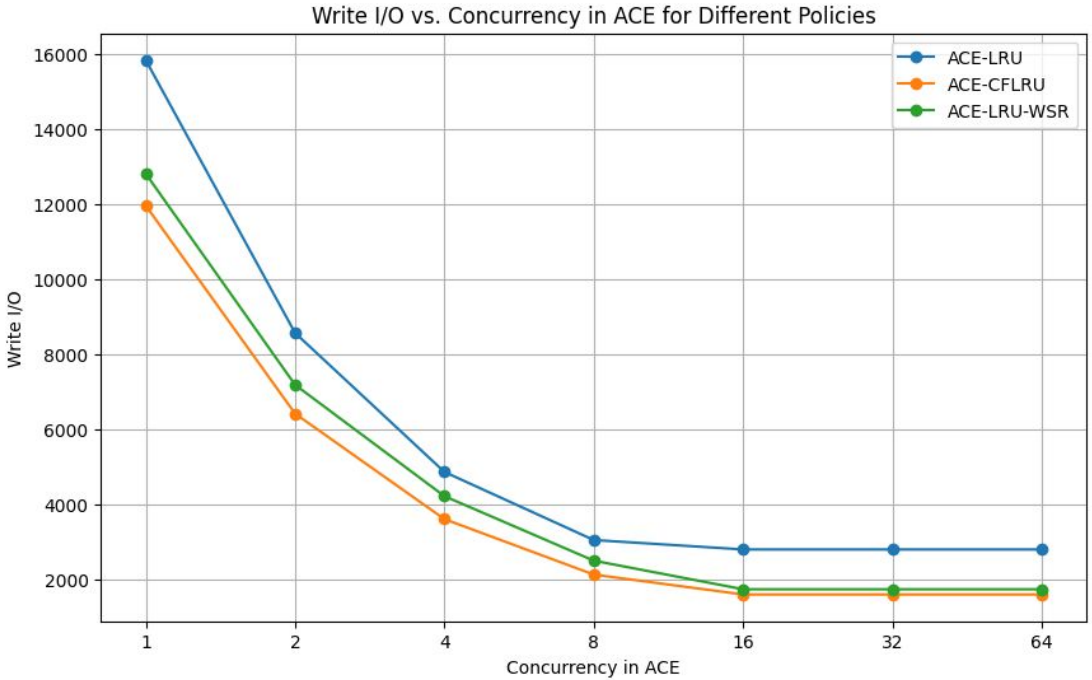


Results: Bufferpool Size

- For ACE as the bufferpool page size increases **Write IOs exponentially decay** until the bufferpool size = disk size.
- Once the bufferpool size exceeds the total disk size, all data can be served directly from the bufferpool, reducing Write IOs to 0.
- As the bufferpool size approaches the number of SSD channels, the decrease in Write IOs slows dramatically, leveling off.



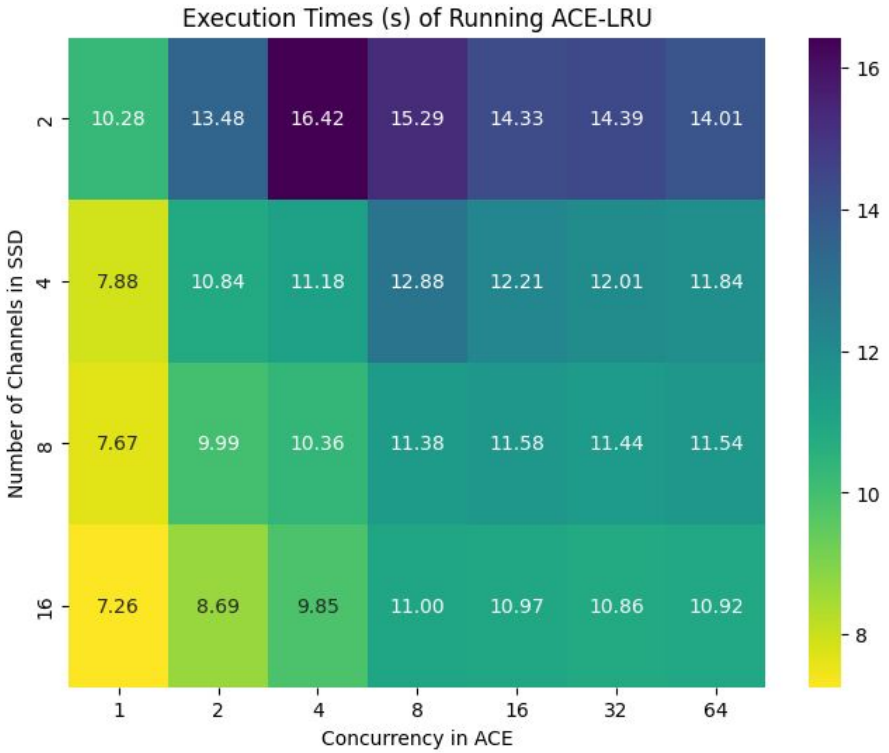
Results: ACE Concurrency



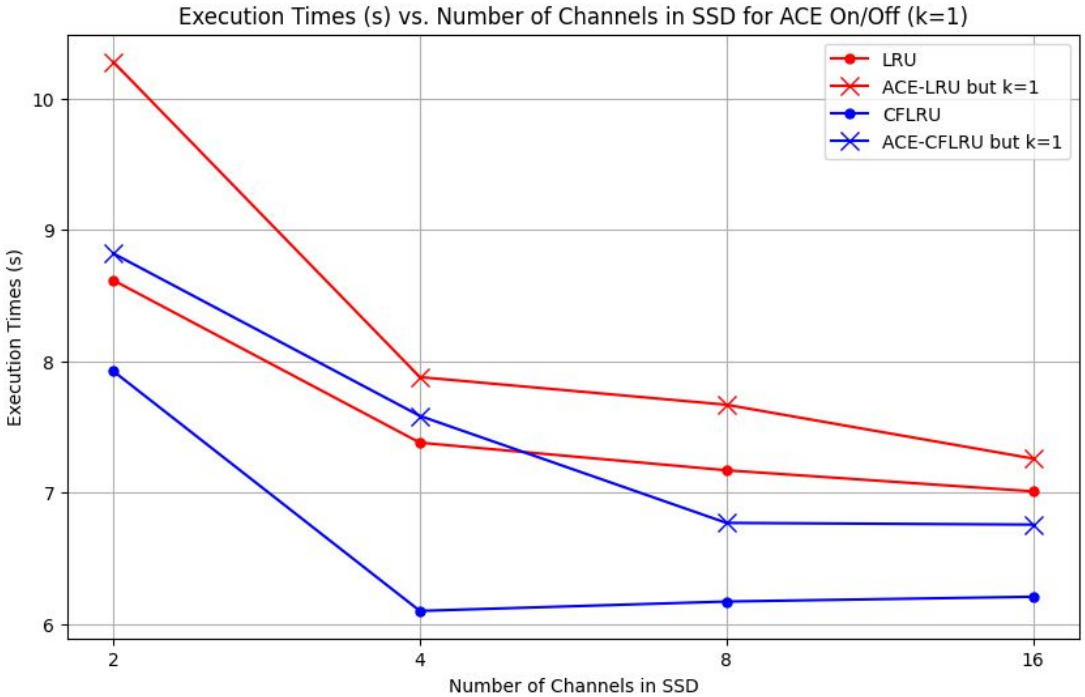
- Concurrency in ACE: Number of pages being written concurrently.
- Number of Write I/Os decrease as concurrency increases.

Results: Heatmap of Execution Times

- Increasing number of channels in SSD for set concurrency in ACE increases performance.

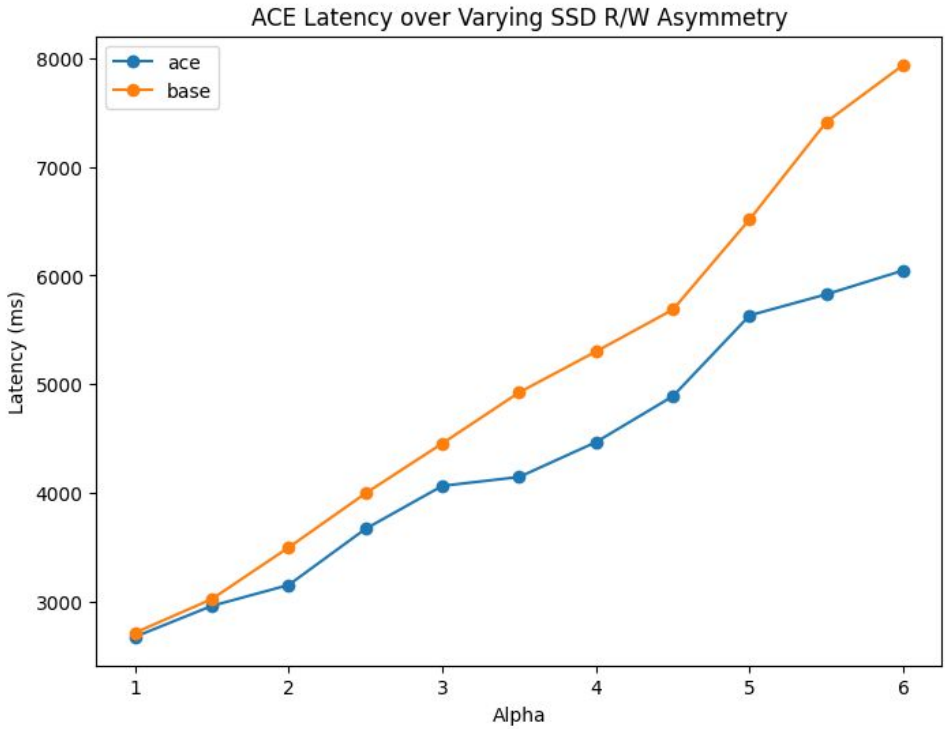


Results: ACE on vs. ACE off



- Using ACE creates overhead, and causes the same operation to be slower.
- k greater -> magnified

Results: SSD Latency



Conclusion

- ACE helps reduce write I/Os for the same workload, thus decreasing the execution time.
- Write IOs exponentially **decrease** as bufferpool size **increases**, reaching **zero** once the bufferpool fully fits the disk data.
- After a point, especially as the bufferpool size nears the number of SSD channels, the rate of improvement slows and **levels off**.
- As concurrency in ACE **increases**, write I/Os **decrease.**
- As number of channels in SSD increases, performance increases.
- This is due to additional overhead of running ACE.

Thank you for listening!