Asynchronous I/O for `cp -r`

Vincent Lee
Gualberto A. Guzman

University of Texas at Austin

vincent_lee@utexas.edu, gualbertoguzman@utexas.edu

December 8, 2017

Overview

Goals

- Use Linux's asynchronous I/O (aio) interfaces to build an optimized version of `cp -r`.
- Learn the effects of caches, readahead, etc. on aio performance.
- Explore the differences in aio on disk drives vs. SSDs.

Major Decisions

1. POSIX AIO vs Linux AIO
   - POSIX AIO is standardized, but `glibc` simulates it completely in userspace [1]
   - Kernel is not able to schedule or reoder aio tasks
2. C++ vs C
   - C++ is just as fast as C and comes with Boost library
   - Easy use of `recursive_directory_iterator` for breadth-first traversal in copied directory
   - For small enough files, we copy directly since overhead of creating aio task dominates performance

Challenges

1. Poor documentation
   - Little to no documentation for Linux AIO
   - Userspace wrapper library not updated since 2015
   - Had to find 2012 Ubuntu docs to find out how to use Linux AIO [2]
2. Backporting Boost method
   - `boost:filesystem::relativize` is crucial to path traversal but not included in version 1.58
   - Decided to backport small (160 lines) implementation from Boost source control
3. Poor `io_queue_run`
   - Userspace wrapper library polls kernel event queues and processes one event per system call
   - Opted to invoke system calls manually
   - Decreased number of system calls to handle same number of events

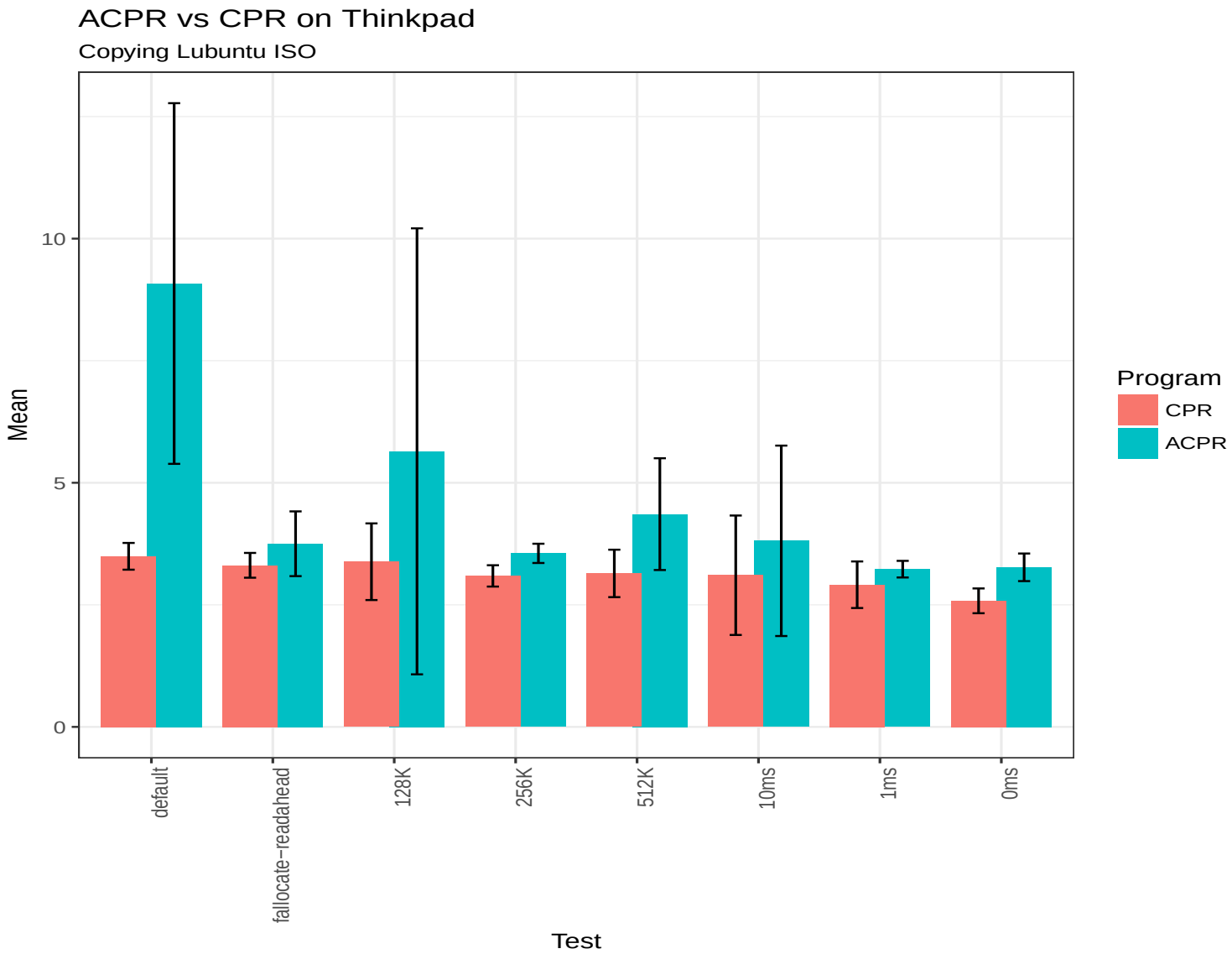
Challenges

4. Cheating `fsync`
   - Trying to copy Linux 4.4 source (711 MB).
     `cp -r`: 1 second, `acpr`: 7 minutes+
   - Turns out `cp -r` does not `fsync` anything after finishing (verified using `strace`)
   - So we turned it off as well, adding a flag to reenable it
   - `acpr` now copies Linux 4.4 tree in a respectable 3 seconds with default flags
5. Attributes and Links
   - Currently do not copy all attributes
   - Due to complications with relative links and lack of time, skip all symbolic links

Evaluation: SSD

- Environment
  - 2013 Thinkpad X1 Carbon
  - Quad-core Intel Core i7-4600U, 8 GB RAM, Samsung SSD
- Copying Large Files
  - Copy single folder with 880 MB Lubuntu 16.04.2 ISO
  - `acpr` took 9s by default, speedups to 3.75s by preallocating the file with `fallocate` and initiating `readahead`.
  - Increasing block size to 256K yields speedups, going to 512K produces a slowdown from 256K
  - Adjusting timeout for gathering events from kernel has negligible effect

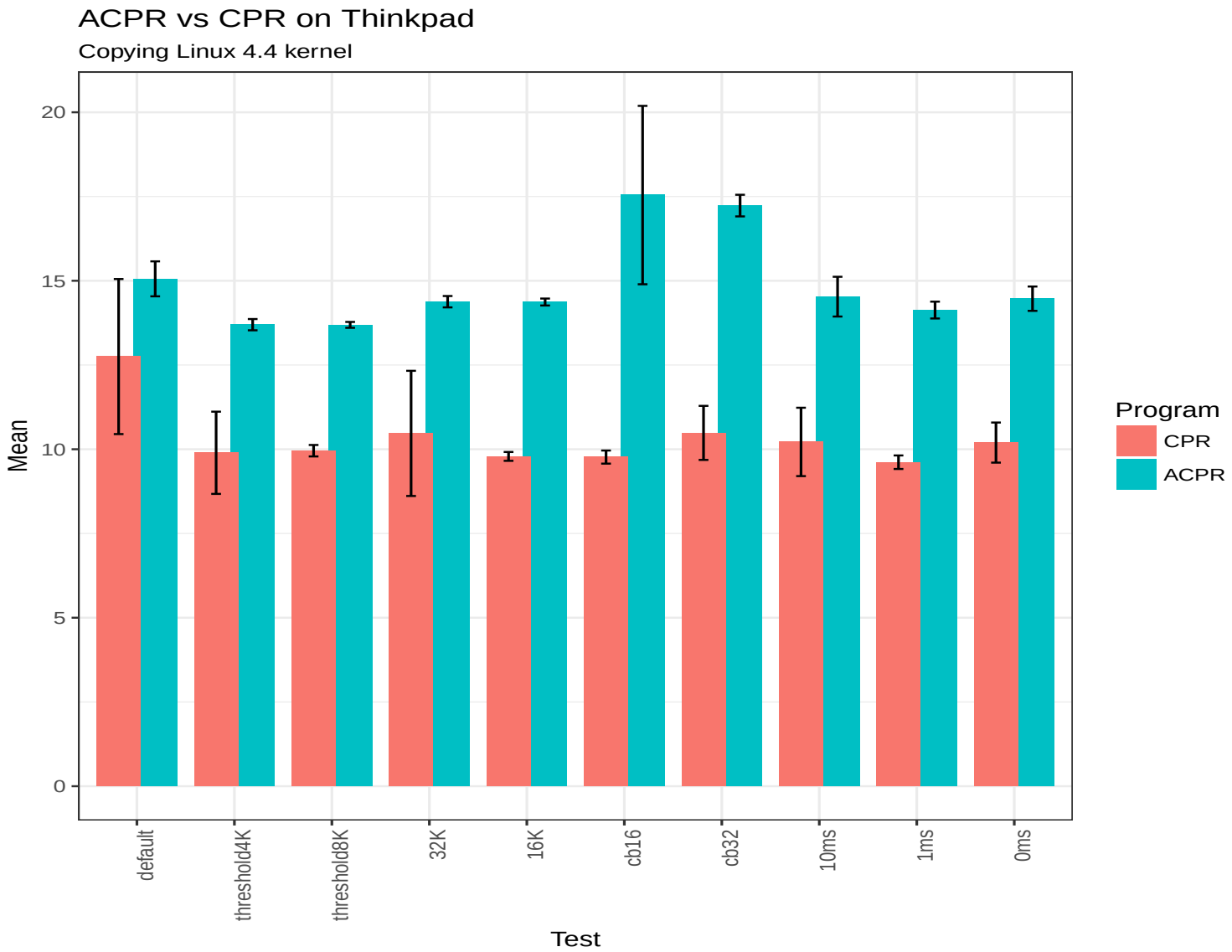
Evaluation: SSD



Evaluation: SSD

- Copying Linux Source
  - Both `cp` and `acpr` quite slow: 12.75s and 15.05s by default
  - Tried raising threshold for using AIO (since most files small), slight performance boost to 13.69s
  - Tried smaller block size (since most files small), slight performance boost to 14.37s
  - Tried increasing number of `iocb` used per file copy, performance loss
  - Again, adjusting timeout had negligible effect

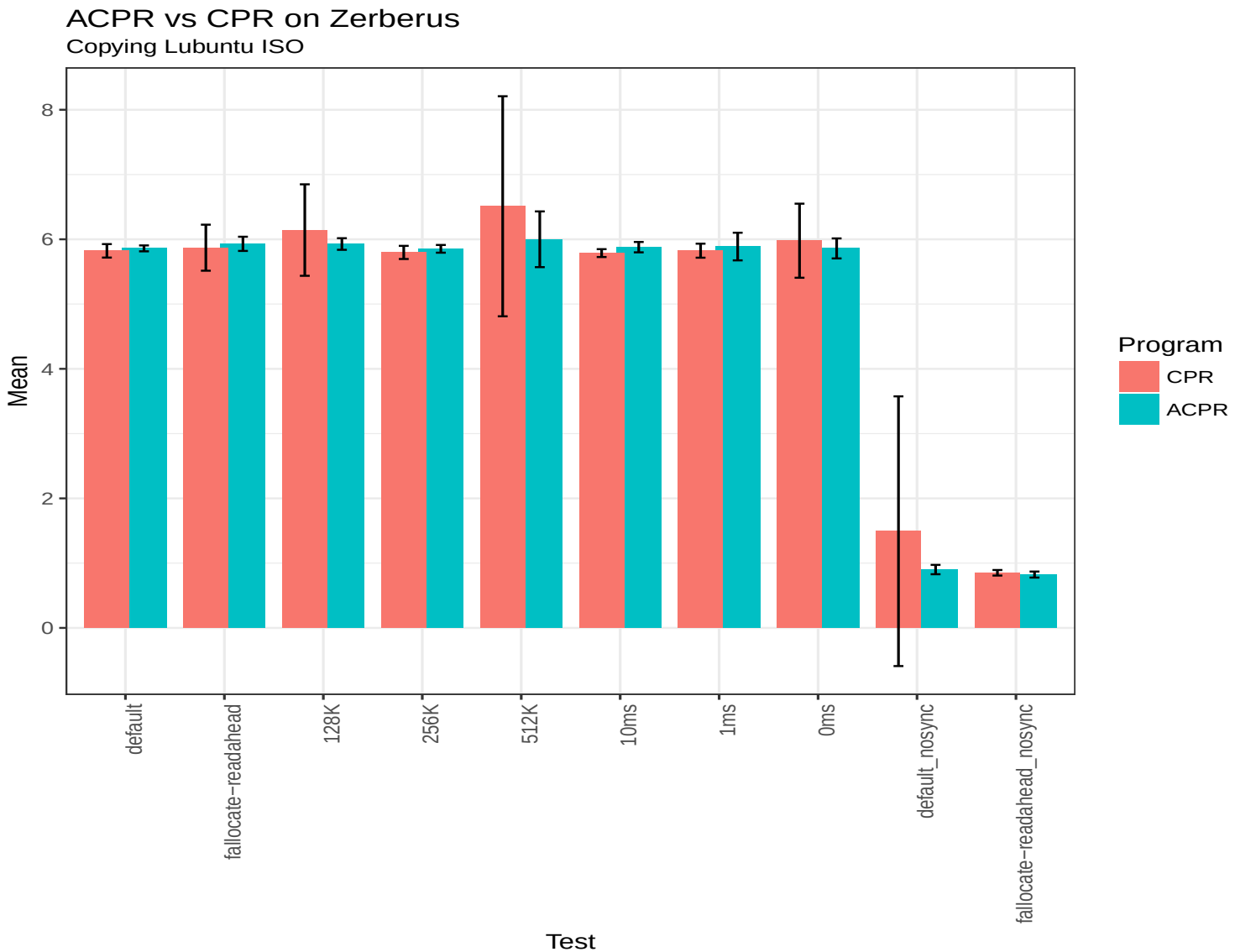
Evaluation: SSD



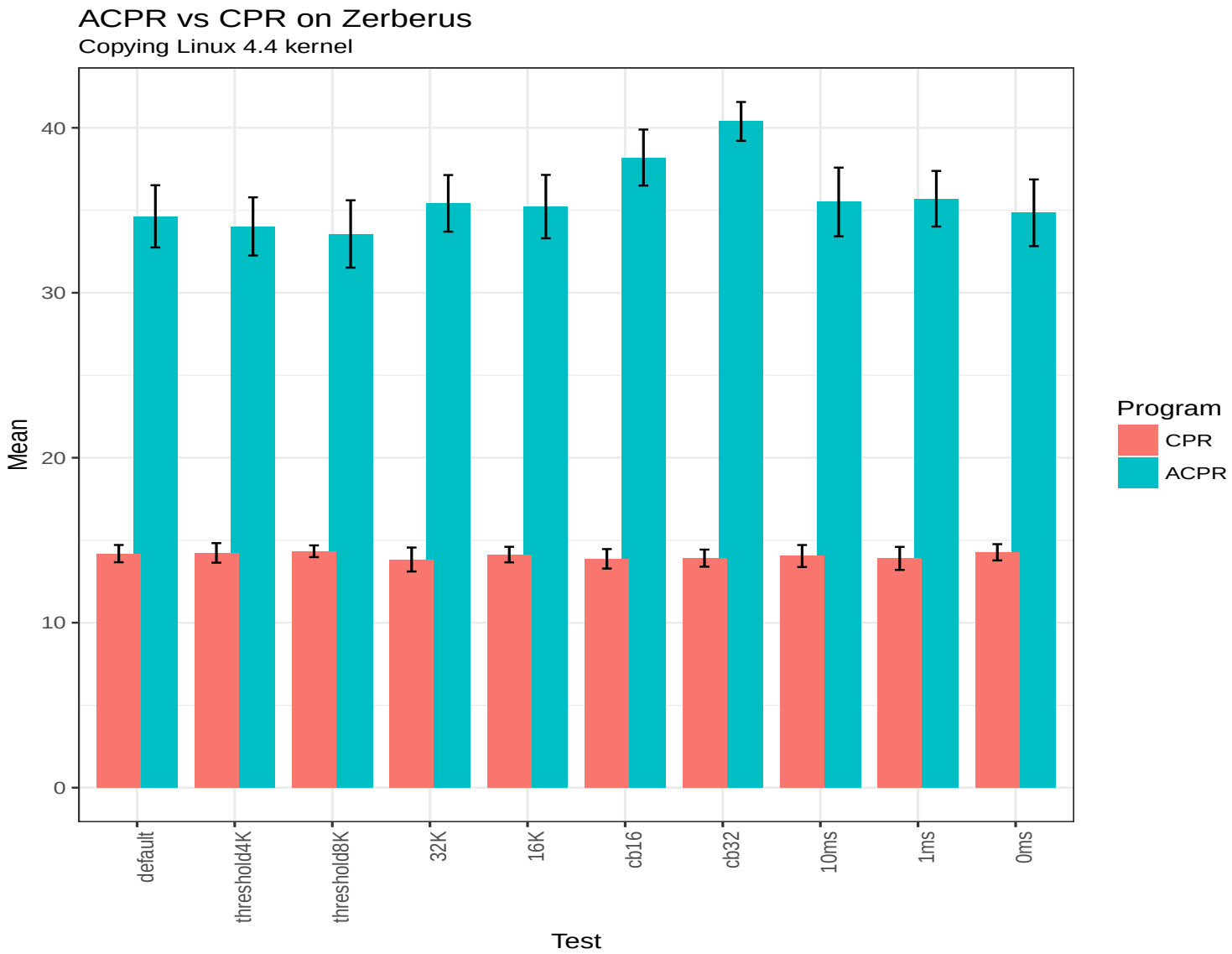
Evaluation: HDD

- Environment
  - zerberus.csres.utexas.edu
  - 32-core Xeon system, 128 GB RAM, 7200 RPM Hard Disk
- Copying Large Files
  - `cp` took 5.82s, `acpr` 5.86s
  - Increasing block size slowed it down
  - For all other experiments, negligible changes
  - We attribute this to disk bottleneck as we are flushing before each iteration
- Copying Linux Source
  - Slow for both: `cp` took 14.9s, `acpr` 34.6s
  - For all other experiments, negligible changes as well

Evaluation: HDD



Evaluation: HDD



Evaluation: HDD

- Copying Large Files, Without Caching
  - If disk is the bottleneck, what if we disable flushing before each test?
  - As expected, giant performance jump: `cp` 1.49s, `acpr` 0.9s
  - Using fallocate and readahead and re-running yields more improvmeent: `cp` 0.85s, `acpr` 0.82s
  - We beat cp!
  - Possibly due to the kernel using different memory management strategies? Not fully certain what the cause is since both should be served from buffer cache.

Conclusion

- Got close (or surpassed in two cases) `cp` performance using AIO
- Effects were more pronounced on SSD than HDD, since HDD was heavy bottleneck
- Improvements: multi-threaded event processing, support symlinks, batching write task submissions, custom `copy_file`

References


aio(7), http://man7.org/linux/man-pages/man7/aio.7.html


Asynchronous IO,
http://manpages.ubuntu.com/manpages/precise/en/man3/io.3.html