http://clyde.itu.dk
A project funded by the Danish Council for Independent Research

CLyDE Mid-Flight:
What we have learned about the
SSD-Based IO Stack

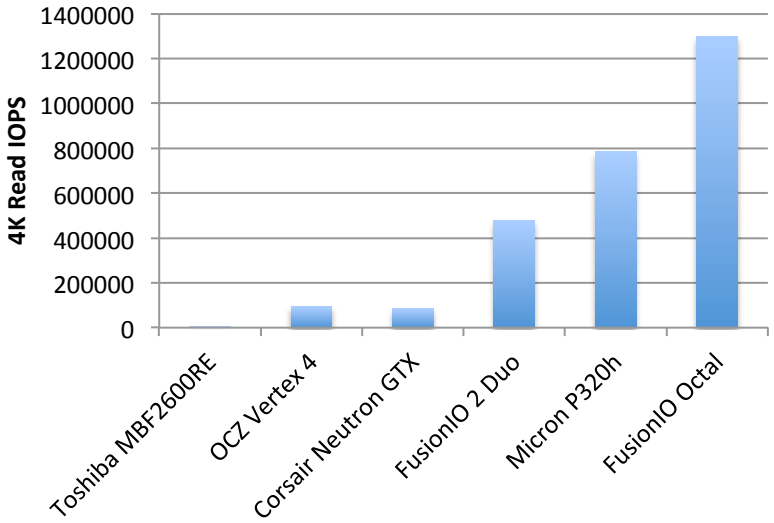
Philippe Bonnet – phbo@itu.dk

Joint work with Luc Bouganim (INRIA), Niv Dayan (ITU), Matias Bjørling (ITU), Jesper Madsen (ITU)

In Lyllaboration with Jens Axboe (Facebook), David Nellans (Nvidia), Zvonimir Bandic (HGST), Qingbo Wang (HGST), Aviad Zuck (Tel Aviv Univ.)

The Advent of SSDs

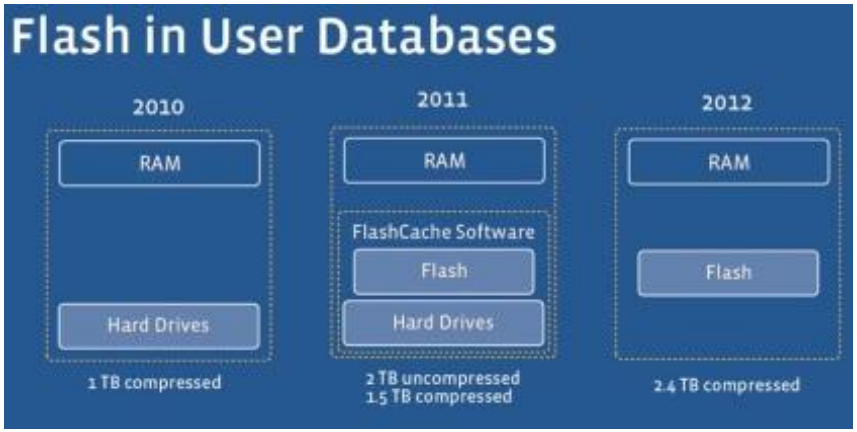
Balanced High-End Systems



High throughput/Low latency reduces the gap between RAM and IO performance

Energy Efficient Storage

Fewer Watts/TB drives higher data density on the cloud (e.g., OCZ vertex 4: 5W/TB; Toshiba MBF: 22W/TB)



Jason Taylor, Flash Memory Summit, 08/13

The Challenges

AWS I2.8xlarge instance: 32 vCPU, 244GiB RAM, 8x800 GB SSD, $6.82/h

SSD I/O Performance

To ensure the best IOPS performance from your I2 instance, we recommend that you use the most recent version of the Amazon Linux AMI, or another Linux AMI with kernel version 3.8 or later. If you use a Linux AMI with kernel version 3.8 or later and utilize all the SSD-based instance store volumes available to the instance, you can get at least the minimum random IOPS (4,096 byte block size) listed in the following table. Otherwise, you'll get lower IOPS performance than what is shown in the table.

Instance Size	Read IOPS	First Write IOPS
i2.xlarge	35,000	35,000
i2.2xlarge	75,000	75,000
i2.4xlarge	175,000	155,000
i2.8xlarge	365,000	315,000

As you fill the SSD-based instance storage for your instance, the number of write IOPS that you can achieve decreases. This is due to the extra work the SSD controller must do to find available space, rewrite existing data, and erase unused space so that it can be rewritten. This process of garbage collection results in internal write amplification to the SSD, expressed as the ratio of SSD write operations to user write operations. This decrease in performance is even larger if the write operations are not in multiples of 4,096 bytes or not aligned to a 4,096-byte boundary. If you write a smaller amount of bytes or bytes that are not aligned, the SSD controller must read the surrounding data and store the result in a new location. This pattern results in significantly increased write amplification, increased latency, and dramatically reduced I/O performance.

SSD controllers can use several strategies to reduce the impact of write amplification. One such strategy is to reserve space in the SSD instance storage so that the controller can more efficiently manage the space available for write operations. This is called *over-provisioning*. The SSD-based instance store volumes provided to an I2 instance don't have any space reserved for over-provisioning. To reduce write amplification, you should leave 10% of the volume unpartitioned so that the SSD controller can use it for over-provisioning. This decreases the storage that you can use, but increases performance.

You can also use the TRIM command to notify the SSD controller whenever you no longer need data that you've written. This provides the controller with more free space, which can reduce write amplification and increase performance. For more information about using TRIM commands, see the documentation for the operating system for your instance.

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/i2-instances.html

Cloud provider
How to design and characterize the IO service?

DBMS designer
What is the impact of this new IO stack on DBMS design?

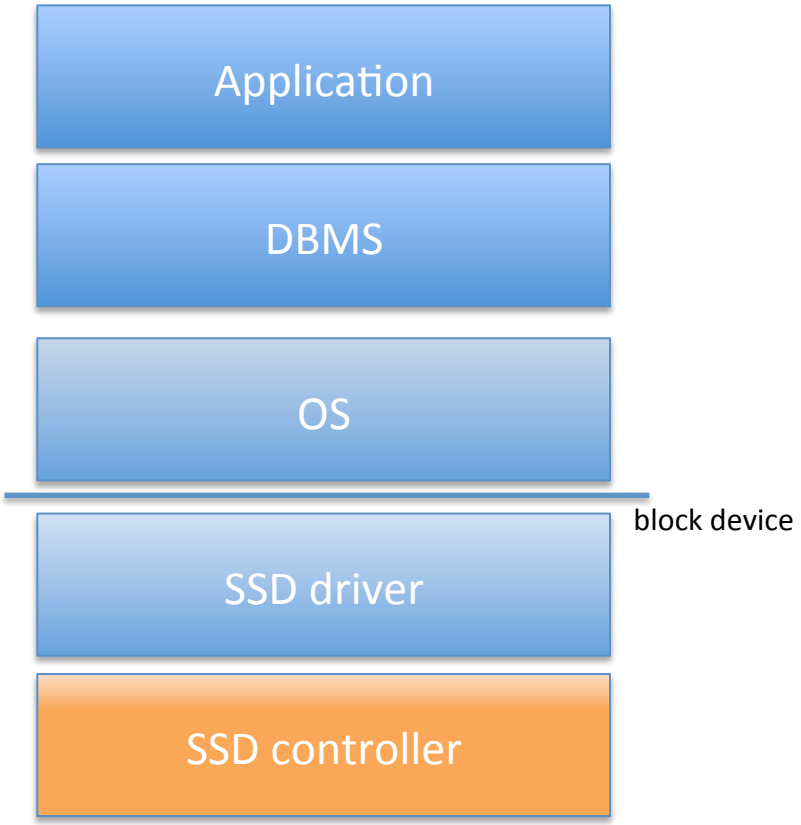
Focus of CLyDE

Database Administrator
How to tune for performance?

How to design for SSDs?

Business as usual

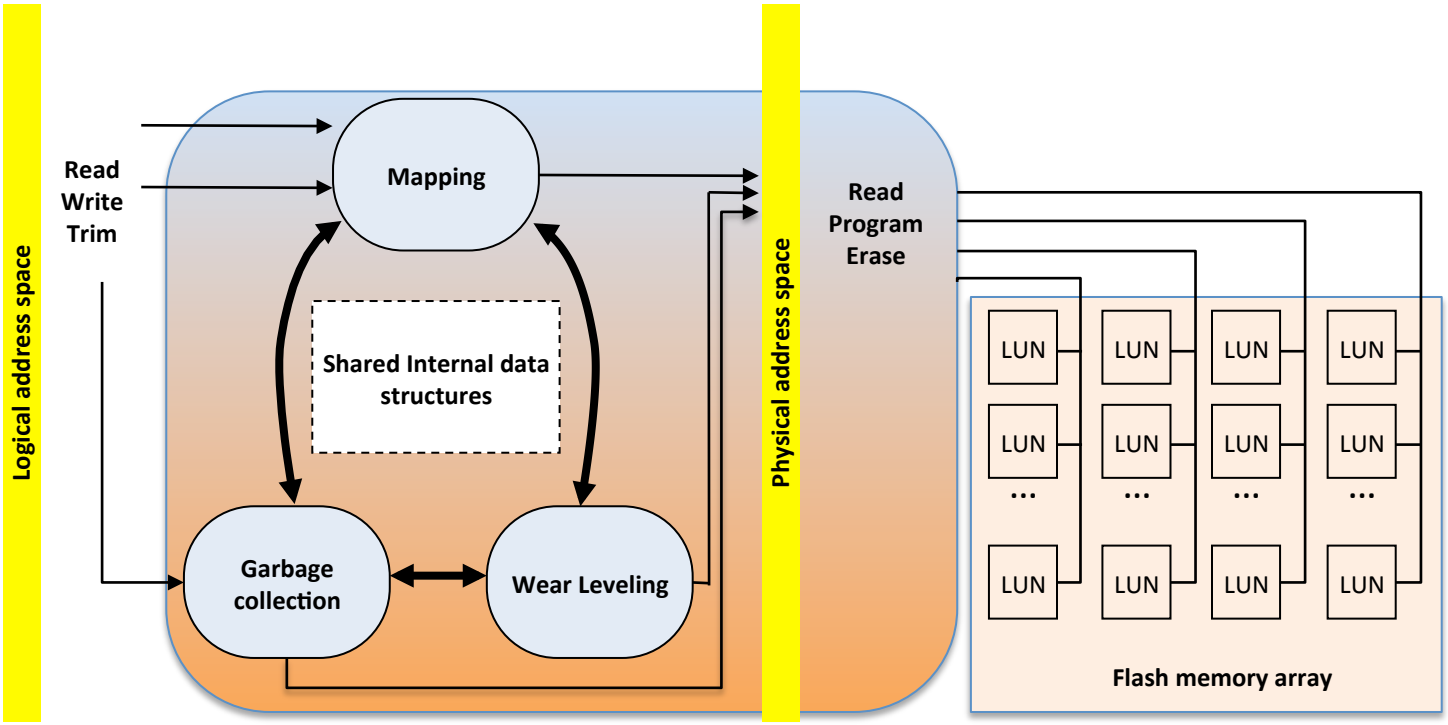
- Layered design
- SSD as block device
- SSD as a black box
- Performance model for SSD to drive design decisions



Lessons learnt [CIDR'09][Sigmod'10][DEB'10]:

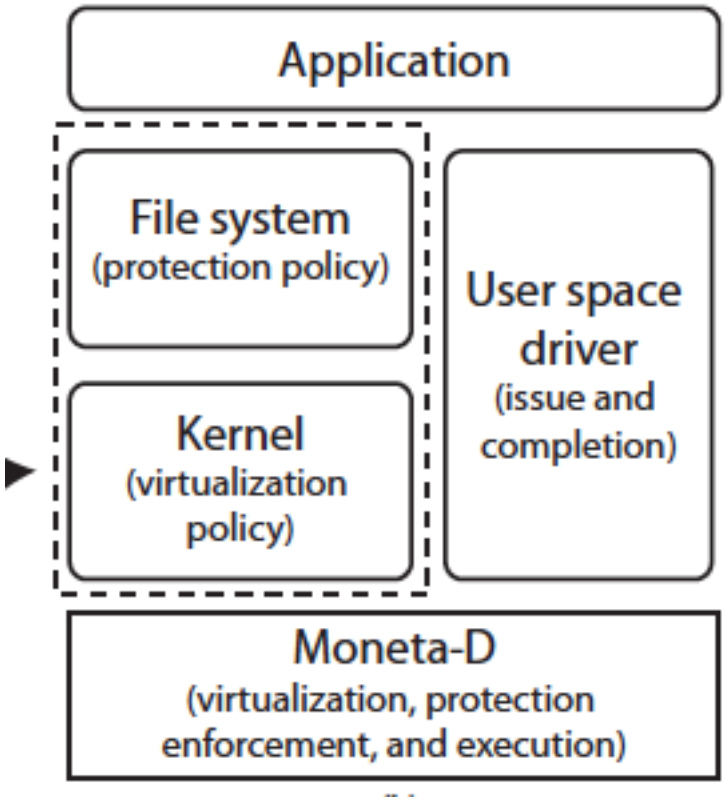
- Performance varies across SSDs, in time for a SSD (with firmware updates, depending on IO history)
- Performance varies for a given IO pattern with target size, with concurrency, with submission rate

SSD Internals



Flash Translation Layer (FTL)
Implemented on SSD controller or SSD driver

How to design for SSDs?



Moneta-D [asplos12]
Caulfield et al., UCSD

New approaches

- Layers revisited:
  - OS bypassed (e.g., Moneta-D [asplos12])
  - Avoid duplication of work (e.g., from ARIES to MARS [SOSP13])
- Cross-layer optimizations
  - Trim command [HPCA11]
  - Extending fadvise [NVM12]
- New SSD interface:
  - Communication abstraction to replace memory abstraction
  - More complex address space

CLyDE Approach

1. Keep traditional layers
2. Focus on SSD interface to support cross-layers optimization

Talk outline:

- SSD Simulation: EagleTree and LightNVM
- Multi-queue design for the Linux Block Layer
- Tree-Based SSD interface for FS/SSD co-design

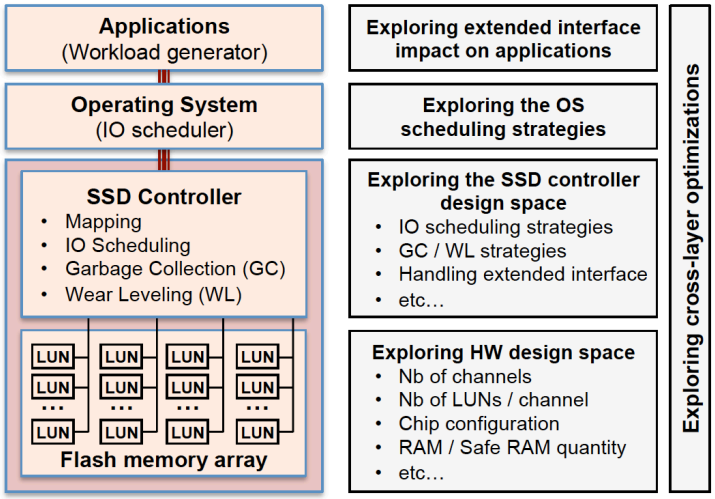
Exploring the Design Space

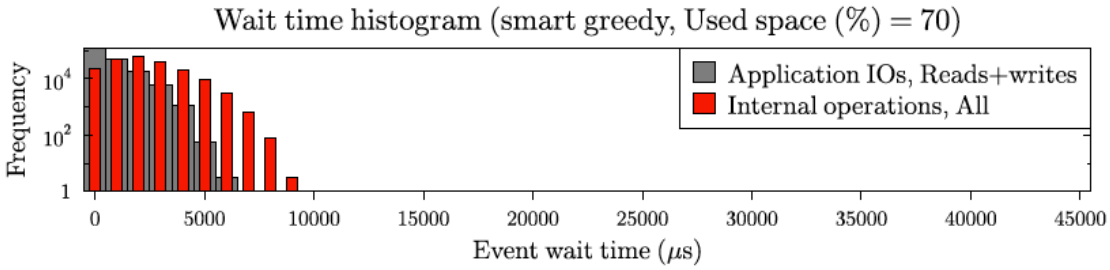
- Fundamental questions:
  - What is the relative importance of data placement and scheduling within the FTL?
  - Should scheduling adapt to HW configuration? to app workload?
- Design choices:
  - within each layer
  - across layers
- Two complementary approaches:
  - EagleTree [VLDB'13]: Simulated SSD driver/controller to experiment with actual OS/apps
    - Wall-time clock
  - LightNVM [NVMW'14]: Simulated SSD/OS/apps to experiment with
    - Dynamic run-time clock

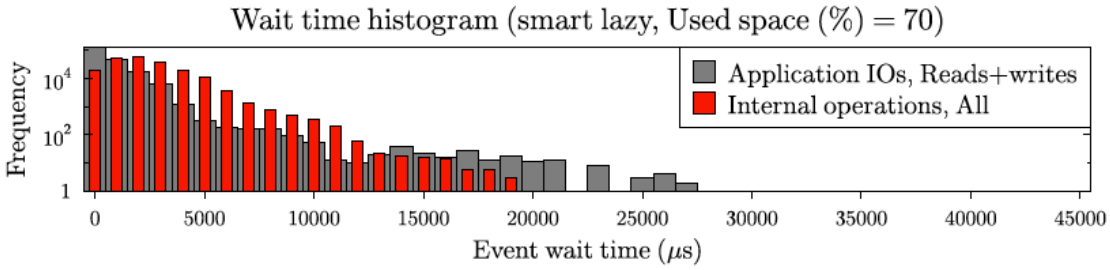
https://github.com/nivdayan/EagleTree

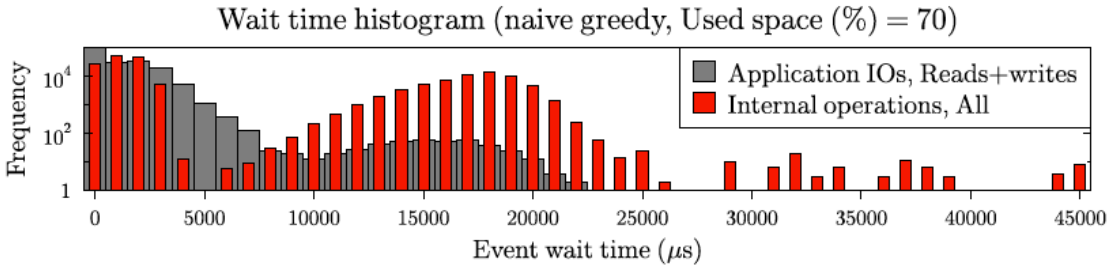
Niv Dayan: nday@itu.dk

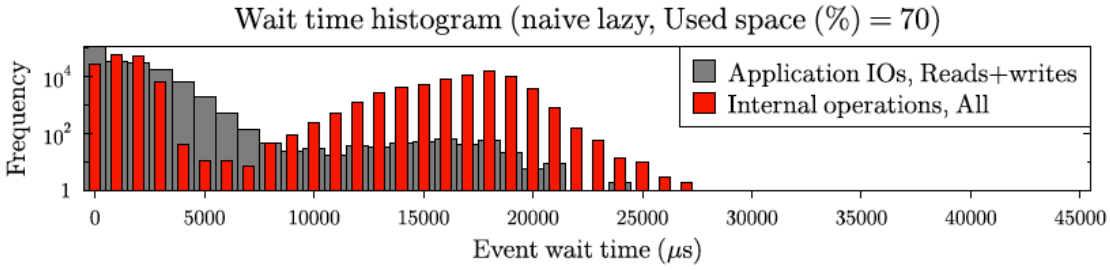
EagleTree [VLDB'13]





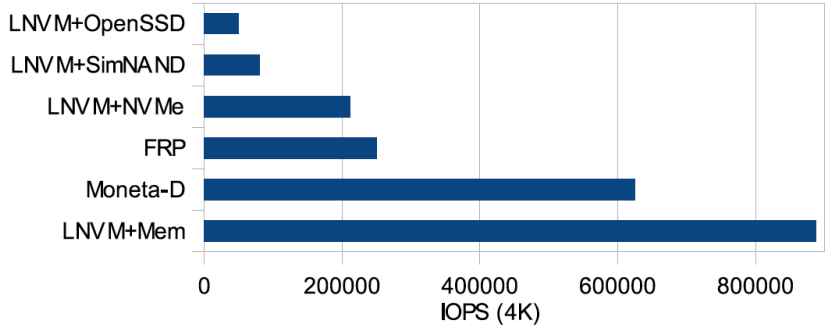


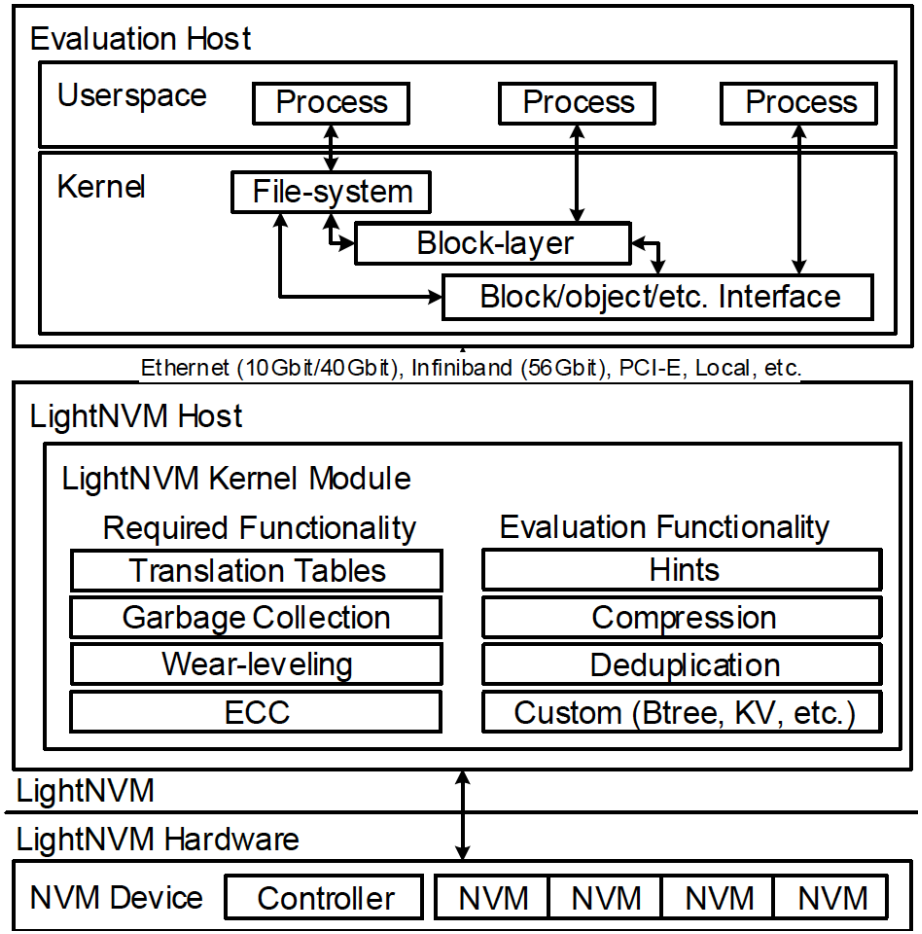




Matias Bjørling: mabj@itu.dk

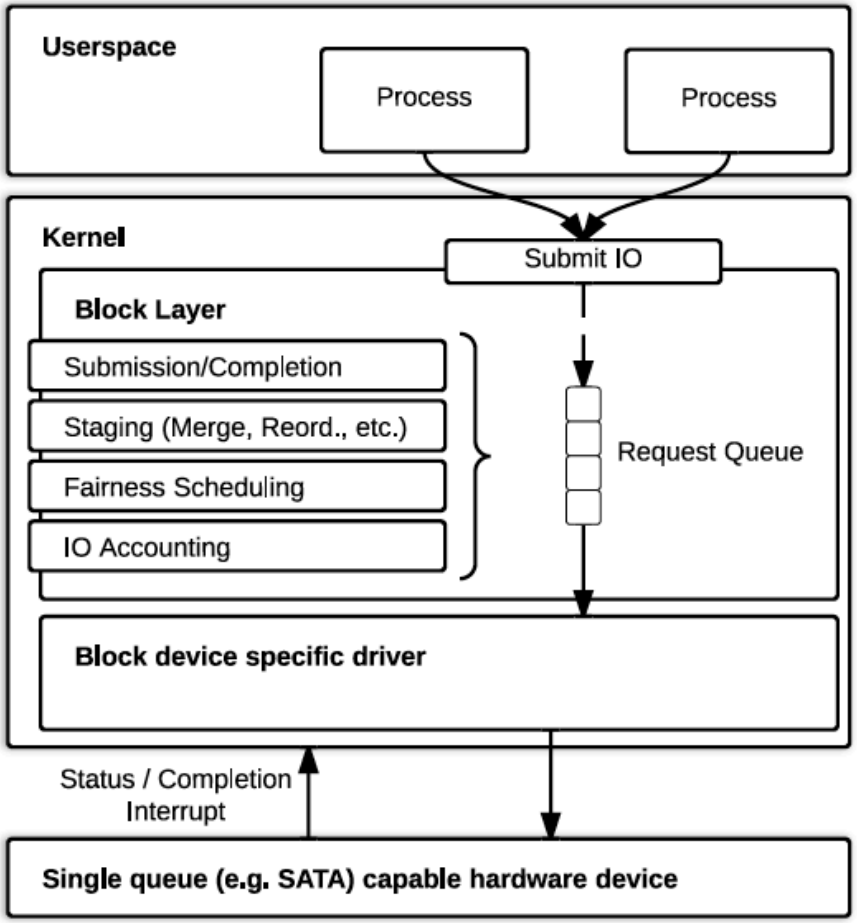
LightNVM [NVMW'14]





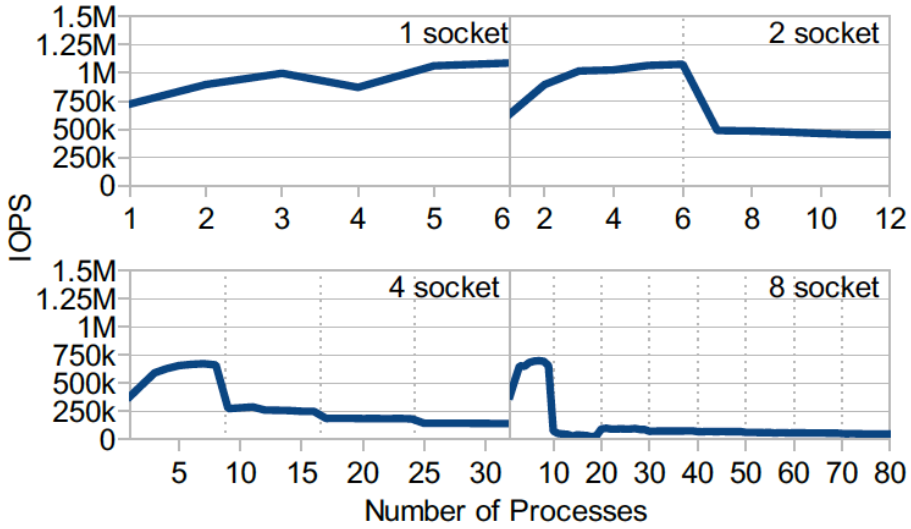
Platform	BlueSSD	VSSIM	OpenSSD	FRP	Moneta	LightNVM
Type	Custom HW	SW	HW	Custom HW	Custom HW	HW/SW
NVM	NAND	NAND	NAND	NAND	PCM	NAND/PCM/etc.
Interface	SATA	N/A	SATA	PCI-E/Net	PCI-E/Net	SATA/PCI-e/Net/Local
Cost	Low	Free	Low	High	High	Varies
Processing Power	Low	N/A	Low	High	Low	High

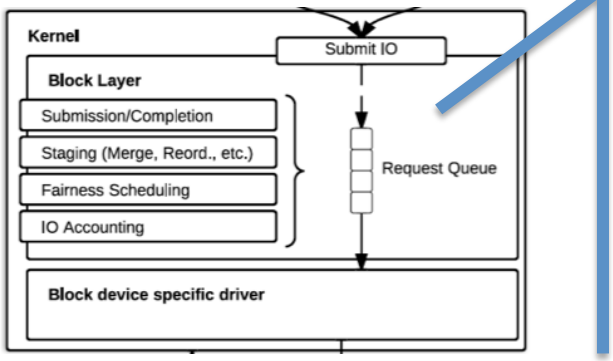
Linux Block IO

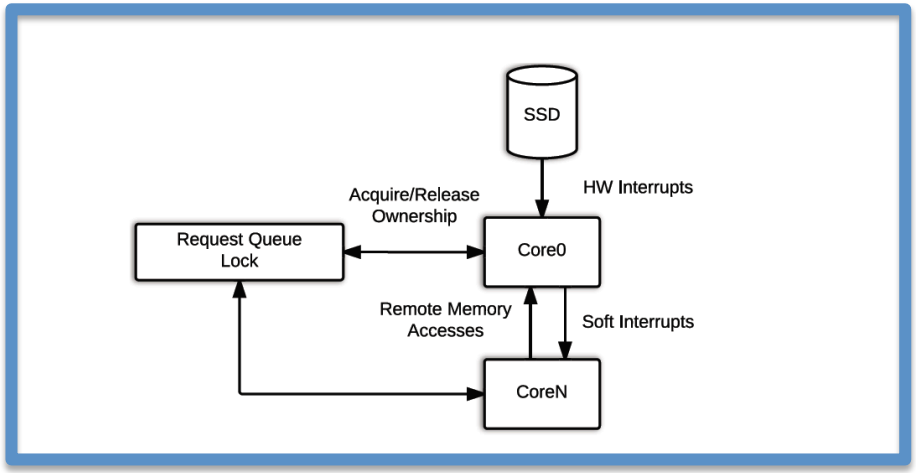


Scalability Problem

Platform (Intel)	Sandy Bridge-E	Westmere-EP	Nehalem-EX	Westmere-EX
Processor	i7-3930K	X5690	X7560	E7-2870
Num. of Cores	6	12	32	80
Speed (Ghz)	3.2	3.46	2.66	2.4
L3 Cache (MB)	12	12	24	30
NUMA nodes	1	2	4	8

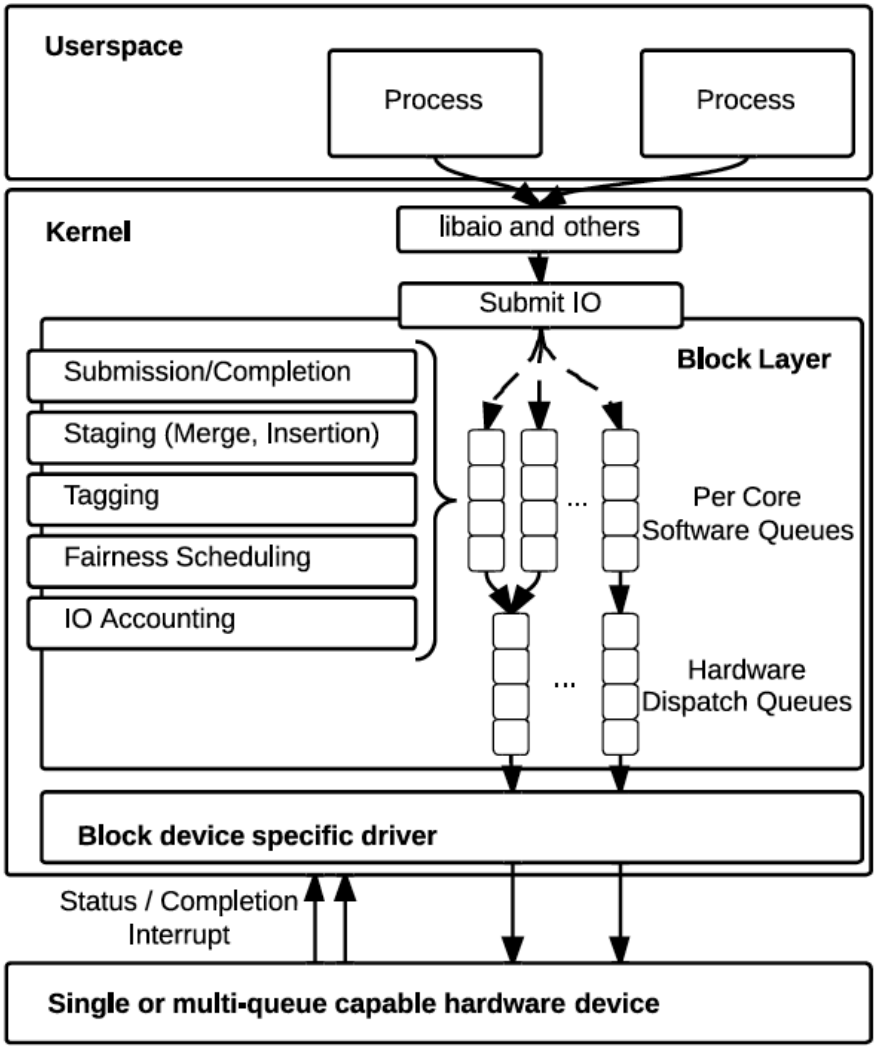




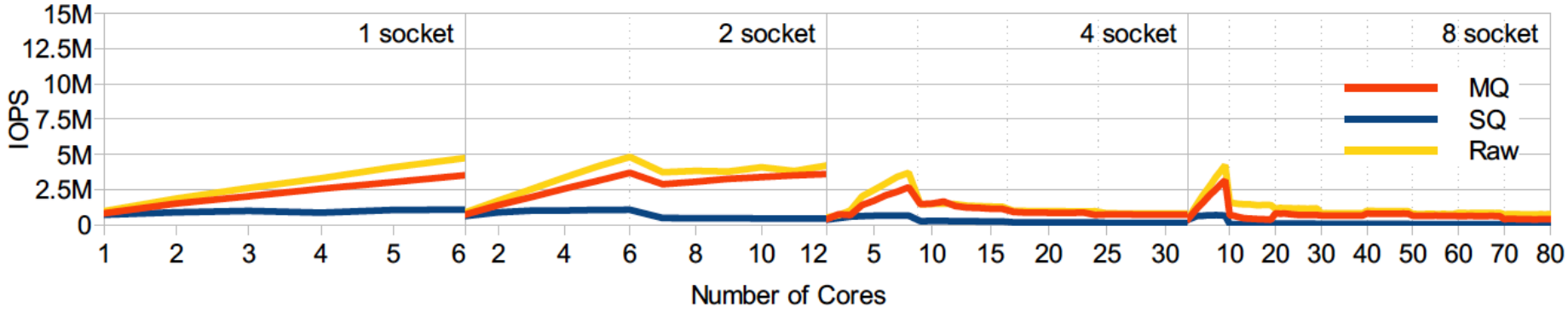


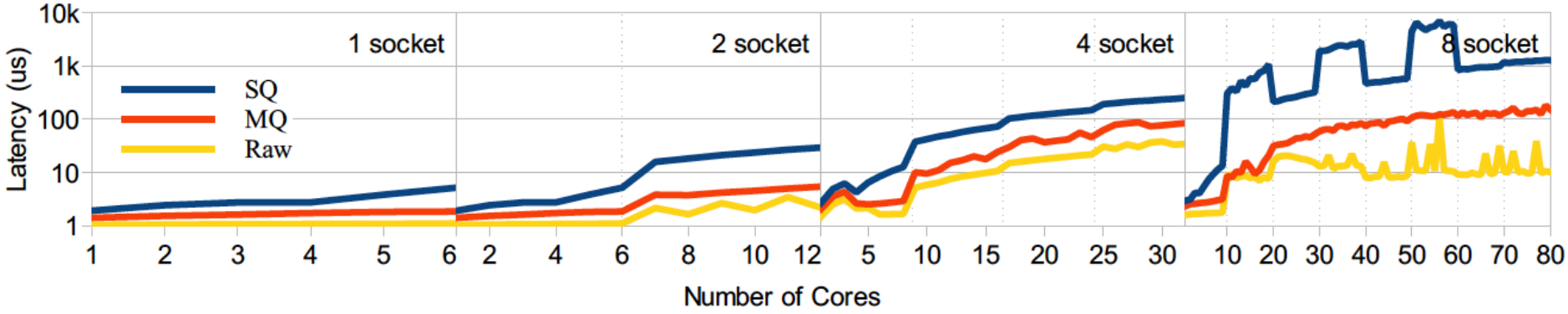
Jens Axboe's design

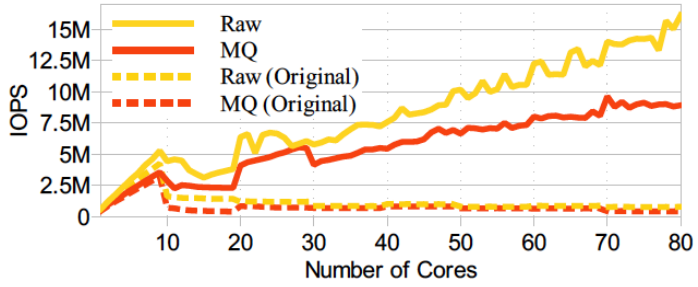
Multiqueue Approach



Experimental Results [Systor13]







Joe Williams
@williamsjoe

Follow

Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems (coming in linux 3.13) kernel.dk/blk-mq.pdf

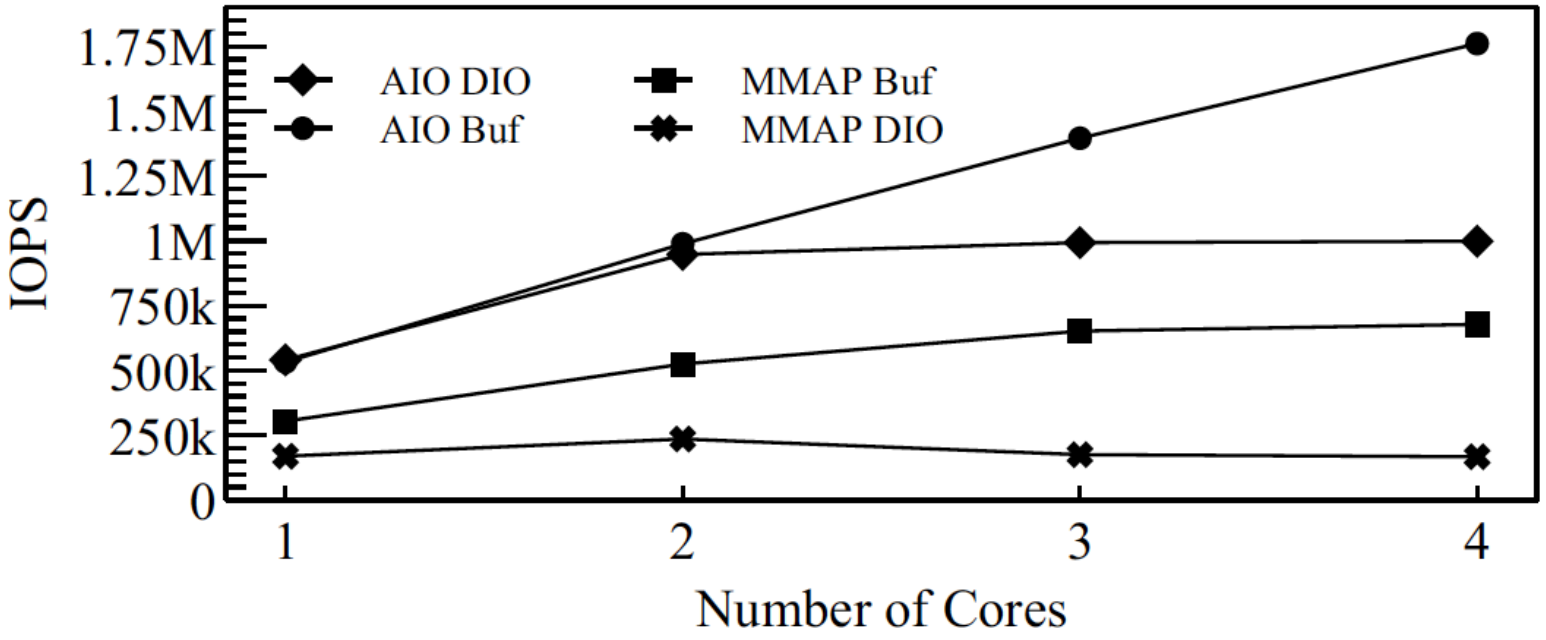
Reply Retweet Favorite More

RETWEETS 87 FAVORITES 72

8:40 AM - 16 Nov 2013

Next Bottleneck within Linux IO Stack

- Page cache



Beyond block device interface

- File System and SSD today:
  - File systems has no clear performance model for SSD
  - SSD makes un-informed assumptions about file system
- Goal: File System / SSD Co-Design
  - SSD exposes an abstraction that is well suited for File System
  - SSD takes leverages this abstraction to optimize data placement/scheduling and thus minimize GC operations

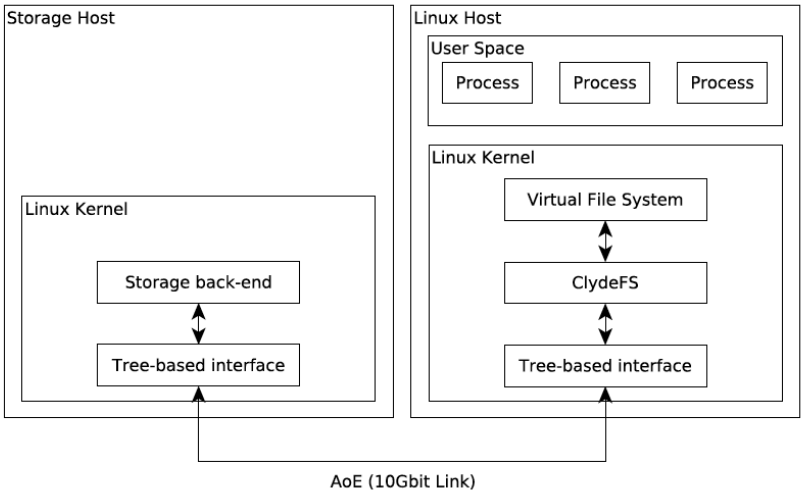
SSD Abstraction

- SSD must provide a mapping from logical to physical address spaces
- File systems in Linux abstracted by VFS (files, dentries, inodes, superblocks)
- Btrfs insight: What if every internal data structure in a file system was an entry in a B-tree?

- Idea: What if SSD provides a Tree-Based interface?
- Approach: Rely on LightNVM as experimentation platform
  - Tree-interface supported within the LightNVM driver (server-side)
  - B-link as tree structure (for a start)
  - CLyDE FS layer mapping VFS abstractions onto Tree-based SSD interface

Jesper Madsen: jmad@itu.dk

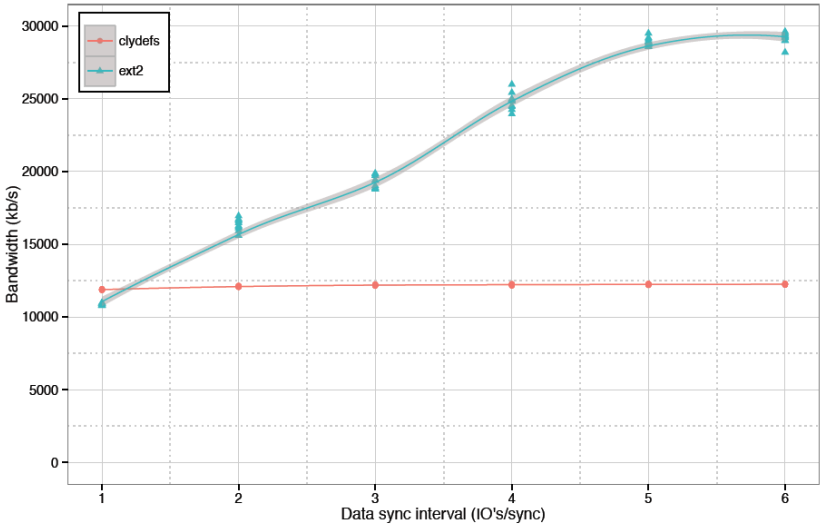
Tree-Based SSD Architecture

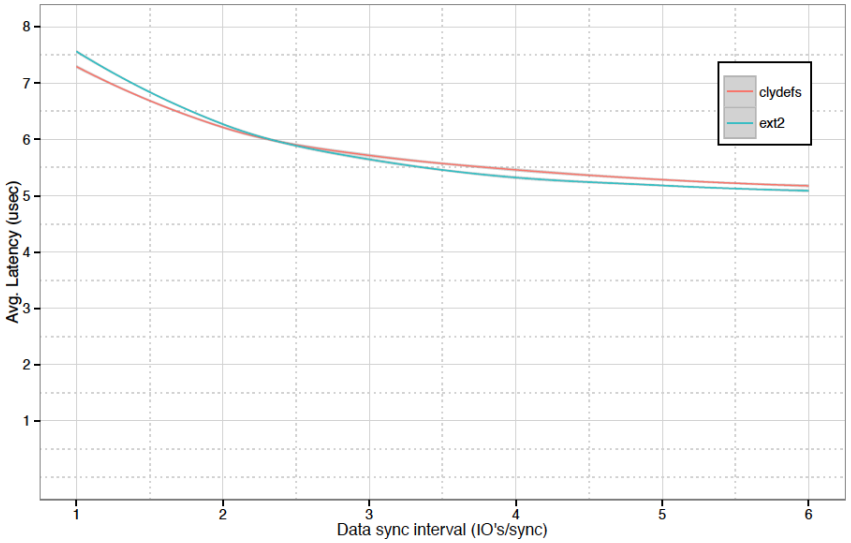


- Tree-Based interface
  - CreateTree
  - RemoveTree
  - InsertNode
  - RemoveNode
  - WriteNode
  - ReadNode
  - TruncateNode

Initial Results

Buffered IOs on RAM-disk(ext2) and in-RAM B+-tree (CLyDEFS)





Constant throughput due to no IO scheduling in current CLyDEFS incarnation

Future Work: Mapping onto NVM, comparison with Btrfs

Mid-Flight Conclusions

- Infrastructure in place to explore cross-layer design
  - EagleTree and LightNVM
- Bottleneck chasing throughout the Linux IO stack (on multicore)
  - Much similar to bottleneck chasing in Shore-MT
  - DBMS storage manager is next
- Room for cross-layer optimization
  - Tree-based structure is a good candidate as replacement for block device interface (i) to convey appropriate information from DBMS to OS to SSD, and (ii) to manage contention
- Need to address challenges of DBMS running on SSD-based virtualized environments