Data-Intensive Systems in the Microsecond Era

Pınar Tözün¹, Ivan Luiz Picoli¹,², Heiner Litz³, Philippe Bonnet¹

¹IT University of Copenhagen
²Samsung Semiconductor Denmark Research (SSDR)
³UC Santa Cruz
state of the world we live in today
Z-NAND storage chips are 8x slower than Optane memory, and 15x faster than existing NAND chips.
SSDs in the µsec era

4K random read using fio - source: AnandTech

SSDs equipped with Z-NAND & Optane deliver at best 5x & 20x the read latency of the underlying storage chip, respectively.
interconnects in the µsec era

Not going to show numbers here.

Latency is of the same order as storage chip latency for fast interconnects.

Fast network-attached storage (RDMA-based) just adds to the latency of direct-attached storage (PCIe).
ftls in the µsec era ...

random writes- source: AnandTech
ftls in the µsec era …

... have even more drastic impact on throughput!

random writes- source: AnandTech
linux IOs in the µsec era

- Separation of control & data plane in Linux now, POSIX out
- Zero copy & minimized synchronization overhead

**Sources:**
- Faster IO through io_uring & Efficient I/O with io_uring & J.Axboe

**Graph:**
- 4k random reads
- 3d xPoint

**Diagram:**
- Memory copy
- Shared rings for submissions and completions
- IRQ-based polling or IRQ-based SSD polling + driver
- User space
- OS kernel
- Storage
- io_uring without polling
- io_uring with polling
- io_uring with polling
- spdk
the benefits of fast storage wasted by
- data movement overheads
  (from device to host & across network)
- black-box generic flash-translation layers
- multitude of software layers

how do we prevent these?
Put Everything in Future (Disk) Controllers (it’s not “if”, it’s “when?”)

Jim Gray

http://www.research.Microsoft.com/~Gray

Acknowledgements:
Dave Patterson explained this to me a year ago
Kim Keeton
Erik Riedel
Catharine Van Ingen

Basic Argument for x-Disks

• Future disk controller is a super-computer.
  » 1 bips processor
  » 128 MB dram
  » 100 GB disk plus one arm

• Connects to SAN via high-level protocols
  » RPC, HTTP, DCOM, Kerberos, Directory Services,…
  » Commands are RPCs
  » Management, security,…
  » Services file/web/db/… requests
  » Managed by general-purpose OS with good dev environment

• Move apps to disk to save data movement
  » Need programming environment in controller

Jim Gray, NASD Talk, 6/8/98
http://jimgray.azurewebsites.net/jimgraytalks.htm
today ...

8-core ARMv8 processor
32GB DRAM
2TB+ of NVM via M.2 slots
4x 10Gb Ethernet

Dragon Fire Card (DFC)
https://github.com/DFC-OpenSource/

Future disk controller is a super-computer.
- 1 bips processor
- 128 MB dram
- 100 GB disk plus one arm
SSD landscape

kv-store needs to change when you start pushing functionality down!

- User space
  - storage manager based on POSIX
  - storage manager based on lightkv

- OS
  - filesystem
  - block layer

- Storage
  - traditional SSD
  - open-channel SSD
  - KV-SSD
  - FPGA-based storage controller

- NIC
  - NIC-based storage controller

- PCIe
  - FTL
  - KV-FTL
  - FPGA
  - lightkv

- lightnvm

- Fabric

app-specific, but no comp. storage
static
generic FTL
app-specific FTL

kv-store needs to change when you start pushing functionality down!
open-channel SSDs

physical address space exposed
  • host can make decisions about **data placement** & **I/O scheduling**

SSD management split between
  • back-end (embedded on SSD) **block metadata** & **wear levelling** (for warrantee)
  • front-end (host-based) FTL mapping of logical to physical address spaces, overprovisioning, & **garbage collection**

separates (**application-customizable**) front-end from (**media-specific**) back-end
potential impact

WHAT?

I/O isolation
• host management of device internal resources for contention avoidance
• control latency predictability – beyond NVMe IO determinism

resource utilization
• controlled data placement to reduce write amplification – beyond NVMe streams

streamline data path
• application-specific FTL

HOW?

computational storage
• offload CPU
• shield host application from complexity of managing the physical space (e.g., flash characteristics)
• co-design of application-specific FTL and OCSSD
SSD landscape

how do we program this to have different application-specific FTLs?
programming interface

core of Ivan’s thesis work

Modularized FTL
App-specific
- Mapping
- Log
- Recovery
- GC
- Other

MLC, 3D TLC, QLC, Open-channel SSD

Upper Layer: Host Interface
Middle Layer: FTL components
Bottom Layer: NVM abstraction
OX to program storage controllers

https://github.com/DFC-OpenSource/ox-ctrl
deconstructing FTL with OX
conclusion

• need to be careful about data movement
  – computational storage would help

• application-specific FTLs would naturally allow computational storage on SSDs
  – use cases:
    • LSM (ongoing work in our lab using RocksDB)
    • BwTree (what Dave Lomet talked about during gongshow)

• open issues
  – ZNS (zoned namespaces)
  – what to push down?

thank you!