Paper Abstract: The move from direct-attach to Composable Infrastructure is being driven by large datacenters seeking increased business agility combined with lower TCO. This requires new remote-attached storage solutions which can deliver extremely high data rates with minimal latency overhead. Unfortunately, the industry-standard embedded processors used in controllers aren't fast enough to manage complex protocols at the required speed. For example, they cannot keep up with the work required to access NVMe SSDs efficiently over an NVMe-oF networked infrastructure. The solution is to add accelerators, typically built using an ASIC, FPGA, or other high-speed hardware. These accelerators offload the processing of protocols such as RDMA, TCP, and NVMe. The result is essentially the same performance for remote storage accessed over a network as for direct-attached storage. The combination provides an optimal blend of high performance, low power, and low cost to yield tremendous CAPEX and OPEX savings in the next-generation datacenter. The technology enables virtually limitless scalability, and will drive dramatically lower TCO for hyperscale and as-a-service datacenter applications.
Bryan Cowger, with over 25 years of storage industry experience, is VP Sales/Marketing at Kazan Networks, a startup developing ASICs that target new ways of attaching and accessing flash storage in enterprise and hyperscale datacenters. Kazan Networks’ products utilize emerging technologies such as NVMe and NVMe-oF. Bryan has spent his career defining and bringing to market successful high-performance storage networking ASICs for such protocols as Fibre Channel, SAS, SATA, Ethernet, PCIe, and NVMe. He has been awarded 4 patents in area of storage controller architecture. Before joining Kazan Networks, he was VP Sales/Marketing & Co-Founder at Sierra Logic, a developer of SATA-to-Fibre Channel controllers. He also spent over 10 years as a design engineer at Hewlett-Packard and Agilent Technologies. He holds a BS in Electrical Engineering from UC San Diego.
Hardware Acceleration of Storage for Composable Infrastructure

Bryan Cowger
VP, Kazan Networks
Agenda

• What is Composable Infrastructure?
• Challenges for CI using existing technologies
• NVMe over Fabrics™ overview

• Sidebar example: Metal Working Machinery (?!)
• Implementation example: Network TCP Engine
• Architectural Comparisons of available solutions
• Practical examples and results/benefits of HW acceleration
Today’s “Shared Nothing” Model 
a.k.a. DAS

Challenges:
- Forces the up-front decision of how much storage to devote to each server.
- Locks in the compute:storage ratio.
Shared Nothing Model
Option A: One Model Serves All Apps

Net utilization: 6 SSDs out of 12 = 50%
Shared Nothing Model
Option B: Specialized Server Configurations

App A: Needs 1 SSD
App B: Needs 2 SSDs
App C: Needs 3 SSDs

Dark Flash eliminated, but limits ability and future app deployments
Disaggregated Datacenter

Pool of CPUs

CPU
CPU

CPU
SSD
SSD
SSD

Pool of Storage

SSD
SSD
SSD
SSD
SSD
SSD
SSD
The Composable Datacenter

Pool of CPUs

Pool of Storage

Utilized SSDs

Spare SSDs

App A: Needs 1 SSD

CPU

SSD

App B: Needs 2 SSDs

CPU

SSD

SSD

App C: Needs 3 SSDs

CPU

SSD

SSD

SSD

Spares / Expansion Pool

- Minimize *Dark Flash*!
- Buy them only as needed
- Power them only as needed
The Composable Datacenter
Real Savings?

• Example:
  • 100k servers
  • 1M SSDs
  
  Assume 40% increase in storage utilization

• CapEx:
  • 1M SSDs * 60% = 600k SSDs
  • Savings of 400k SSDs
  • $300 per SSD
  • $120M savings

• OpEx:
  • Not powering 400k SSDs
  • Assume ~10W per SSD
  • Assume $0.10 KWH
  • $10k savings per day
  • $3.5M savings per year
The Composable Datacenter
Based on NVMe-oF

• Software-defined DataCenter
• Primary application for NVMe-oF™
• “Infrastructure as Code”
• Could be any “fabric”
  • Ethernet
  • Fibre Channel
  • Infiniband
  • Next-gen…
• Hyperscalers focusing on Ethernet
Ethernet Roadmap
“Initial Standard Completed”
The Processor Challenge

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Lamorte, O. Shacham, K. Olukotun, L. Hammond, and G. Batten
New plot and data collected for 2010-2017 by K. Rupp

https://www.karirupp.net/2018/02/42-years-of-microprocessor-trend-data/
Summary of Challenges

• Ethernet speeds going up...
• Embedded processing capabilities plateauing...
• Compute and storage disaggregating...
• ... while storage media latencies are decreasing

• One solution: Speed up how networking controllers are built by basing them on more specialized, dedicated hardware.
Why Dedicate Hardware?

• Metal-stamping machine examples

Programmable

Progressive Stamping
## Two Machines: Side-by-Side

<table>
<thead>
<tr>
<th>Feature</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versatility</td>
<td>Programmable</td>
<td>Hard-coded</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>12 parts / 8 minutes</td>
<td>1 part / 2 seconds</td>
</tr>
<tr>
<td>Latency</td>
<td>8 minutes</td>
<td>20 seconds</td>
</tr>
<tr>
<td>Size</td>
<td>18’ x 17’</td>
<td>6’ x 3’</td>
</tr>
<tr>
<td>Weight</td>
<td>14 tons</td>
<td>2 tons</td>
</tr>
<tr>
<td>Power</td>
<td>9kW</td>
<td>2kW</td>
</tr>
</tbody>
</table>
Typical NVMe-oF Deployment
Networking Controller
Typical Architecture of Inbound Path

Inbound packets

No Errors? Expected? In-order? ...

Y

Fast-path processing

Can be SW/FW or dedicated HW

N

Slow-path processing

Typically SW/FW-based
Inbound TCP Control Engine Example

- Data structure at right used to track status of each TCP frame
- Each inbound frame header must be compared against 320 bits of control information
- Fast-path / slow-path decision to be made
- Various fields must be updated and written back
Inbound TCP Parsing Example

Software-based algorithm:
• Loop 5 times:
  • Read in 64 bits from header (1 clock)
  • Read in 64 bits from data structure (1 clock if in TCM; 10+ clocks if in DRAM)
  • Make decision on multiple fields (2 or more clocks)
• Once entire data structure / header is processed, write back necessary information (5+ clocks)
• Take action to move header information to next part of protocol processing (e.g. NVMe) (5+ clocks)
• Total: 30-50 clock cycles
  • (2X higher if a 32-bit processor)

Hardware-based algorithm:

| DWord | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1     | snd_wnd | rec_wnt | rsvd | flags |
| 2     | snd_mnt | snd_una |      |      |
| 3     | snd_mnt |      |      |      |
| 4     | snd_wnt |      |      |      |
| 5     |      |      |      |      |
| 6     |      |      |      |      |
| 7     |      |      |      |      |
| 8     |      |      |      |      |
| 9     |      |      |      | t_rttseq |
| 10    |      |      |      | recovery |
| 11    |      |      |      | last_ack_sent |
| 12    | rec_wnd |      |      | snd_ssthresh |
| 13    |      |      |      | mss |
| 14    |      |      |      |      |
| 15    |      |      | t_rtt |      |
| 16    |      |      |      | t_rttvar |
| 17    |      |      |      |      |
| 18    |      |      |      |      |
| 19    |      |      |      |      |
TCP Header Processing Hardware

40 bytes of header

40 bytes of data structure

320 bit comparator

("are these equal?")

Fast path vs slow path decision
More Complex Decisions

**X bytes of header**

- n bit comparator

**X bytes of data structure**

- m bit comparator
- p bits

Enable
HW-based Finite State Machines (FSM)

• Essentially just bespoke “very long instruction word” processors

• From Wikipedia: Very long instruction word (VLIW) refers to instruction set architectures designed to exploit instruction level parallelism (ILP).
  • Whereas conventional central processing units (CPU, processor) mostly allow programs to specify instructions to execute in sequence only, a VLIW processor allows programs to explicitly specify instructions to execute in parallel.
  • This design is intended to allow higher performance without the complexity inherent in some other designs.

• Hard-coded state transitions instead of instruction set-based
Typical NVMe-oF Deployment

Server(s) → Ethernet → Fabric-to-PCIe bridge → Fanout → SSDs

JBOF / EBOF / FBOF

NVMe over Fabrics

NVMe (over PCIe)
NVMe-oF Bridge Architecture

- Approximately 150 FSMs running in parallel
- Some simple, e.g. buffer management
- Some quite complex:
  - RoCE v1, v2
  - iWARP /TCP
  - TCP
  - NVMe
  - NVMe-oF
- Slow-path implemented in FW
NVMe-oF Bridge Architecture

- Approximately 150 FSMs running in parallel
- Some simple, e.g. buffer management
- Some quite complex:
  - RoCE v1, v2
  - iWARP /TCP
  - TCP
  - NVMe
  - NVMe-oF
- Slow-path implemented in FW

Approximately 150 FSMs running in parallel
Some simple, e.g. buffer management
Some quite complex:
- RoCE v1, v2
- iWARP /TCP
- TCP
- NVMe
- NVMe-oF
Slow-path implemented in FW

3.5% of die
Purely SW Architecture
NVMe-oF Options

**High-End**
- x86 server-based
- Up to 3.3GHz
- 200W+ / 100Gb

**Mid-range**
- Network Processors
- Up to 3.0GHz
- 15-20W / 100Gb

**Low-power, low-cost**
- Dedicated hardware
- ~0.5GHz
- 7W / 100Gb
A Spectrum of Options

- Myriad choices for NVMe-oF / Composable Infrastructure target deployments
Composable Infrastructure
At Hyperscale

Units of Compute
• Processor
• Memory
• I/O (RDMA, TCP/IP)

Units of Storage
• SSDs
• Fanout
• I/O (RDMA, TCP/IP)
• Manageable
Composable Infrastructure
At Hyperscale

“We need to move the standard unit of compute from the server to the rack; we need to go from local optimization to global optimization. We can deliver higher performance and utilization through the pooling of resources, so by disaggregating independent server and storage systems, the capacity can be more finely allocated to the application.”

Diane Bryant, Intel, IDF16

Cost Optimized Compute

Cost Optimized Storage

Storage Utilization > 80%!
Industry Examples
Industry Example: Tachyon™

• Fibre Channel controller family first introduced in the 1990s
• Entirely HW FSM-based
  • No embedded processors!
• High-level protocol (SCSI) in HW
• Complex algorithms (e.g. FC-AL initialization) in HW
• Generated ~$1B in revenue during the family’s ~20 year lifecycle
Industry Example: Fuji + Optane™

Latency

<table>
<thead>
<tr>
<th></th>
<th>Read Latency (usec)</th>
<th>NVMe-oF Incremental Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Size</td>
<td>512B 4kB</td>
<td></td>
</tr>
<tr>
<td>Native</td>
<td>8.03 8.98</td>
<td></td>
</tr>
<tr>
<td>NVMe-oF</td>
<td>12.20 13.83</td>
<td>4.18 4.85</td>
</tr>
</tbody>
</table>

- **DAS Mode**
- **Disaggregated Mode**

![Diagram of the setup showing connections between Linux Host, Optane SSD, Ethernet Switch, and CPU Interface with a latency of 0.5 usec.]
**Industry Example: Fuji + Optane™**

**IOPS / Bandwidth**

<table>
<thead>
<tr>
<th>Optane Native</th>
<th>Optane NVMe-ofF</th>
</tr>
</thead>
<tbody>
<tr>
<td>4kB Rnd</td>
<td>4kB Rnd</td>
</tr>
<tr>
<td>128k Seq</td>
<td>128k Seq</td>
</tr>
<tr>
<td>571k IOPS</td>
<td>571k IOPS</td>
</tr>
<tr>
<td>2.58 GB/s</td>
<td>2.28 GB/s</td>
</tr>
<tr>
<td>546k IOPS</td>
<td>543k IOPS</td>
</tr>
<tr>
<td>2.16 GB/s</td>
<td>2.18 GB/s</td>
</tr>
</tbody>
</table>

**DAS Mode**

**Disaggregated Mode**

- **Linux Host**
  - NVMe Driver
  - Optane SSD
- **100G RNIC**
  - PCIe 1x16
  - 1x100G
- **Ethernet Switch**
  - 100G
- **FIO**
- **NVMe Driver**
- **NVMe-ofF Host Driver**
- **RNIC Driver**
- **100G RNIC**

**0.5 usec**
Takeaways

• Multiple solutions being delivered to the market this year

• Decision to make: Versatility vs optimized hardware

• Composable Infrastructure is nearing reality

• No need to sacrifice performance... while reaping upsides

• 2019 will be the year of initial NVMe-oF deployments
Q & A
Thank You!