NVMe based Video and Storage Solutions for Edge based Computational Storage

John Plasterer
NETINT Technologies
Introduction to Video Encoding at Scale
Video Distribution in the 60s
Video Distribution in the 90s / 2000s
Video Distribution in 2010s

![Netflix](https://via.placeholder.com/150)

![YouTube](https://via.placeholder.com/150)
Video Distribution Now
Video Experience Distribution

Broadcast Video Experiences

Personalized Videos Experiences

Viewers per stream

# of Streams
Video Transcoding for end application

- Video needs to be distributed in many formats
  - Instantaneous viewing at multiple resolutions
Video Edge Encoding and Storage in the Video Cloud

**Use Cases with primary video flows**

![Diagram showing video streaming, video surveillance, and interactive video use cases with latency considerations.]

Source: NETINT adapted from LF Edge, and IHS Markit, NFV Strategies, Global Service Provider Survey, June 2017
Video Encoding Alternatives Compared: Density and Power

Approximated infrastructure required for 80x 1080p30 Encoding Streams, or 40x Typical Encoding Ladders.

- NVMe Interface
- ASIC-based Transcoding
- Highest Density
- Least Rack Space
- Lowest Power

Software Encoding on Compute Server

GPU Accelerators Hosted in Server

FPGA Accelerators Hosted in Server

ASIC encoders in storage form factors
Implementation of Video Encoding using NVMe
Solution Requirements

- Fast time to market to capture fast moving live video market
- Needs to use robust, highly tested infrastructure as much as possible
- Needs to be deployable quickly by customers
Why use a storage form factor?

• Using storage interface allows scaling using standard server infrastructure

• Transcoding U.2 modules plug into SSD slots of NVMe Server
Why use NVMe?

- Easily combine storage and video into the same PCI-Express Interface
- Leverage significant amount of industry investment in NVMe
  - Kernel
  - Drivers
  - Hardware
Application of NVMe to control SSD and video processing
# Video Transcoder – Software Integration

- **FFmpeg integration** achieved by installing FFmpeg Codec Lib and SDK into host server
  - Seamlessly abstracts FFmpeg video transcoding functions from 1 or more transcoder modules

- Video transcode functions controlled through standard NVMe protocol

## Diagram

![Diagram showing FFmpeg integration with NVMe device driver and ASIC video processor](image)

<table>
<thead>
<tr>
<th>PCIe 4.0 x4 Interface</th>
<th>NVMe</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASIC Video Processor</strong></td>
<td></td>
</tr>
<tr>
<td>H.265/HEVC Encode/Decode</td>
<td>H.264/AVC Encode/Decode</td>
</tr>
<tr>
<td>4K @ 60 fps</td>
<td>4K @ 60 fps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Server Qty</th>
<th>1 Host Server</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Video Inputs</th>
<th>Qty=1, N</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 / H.265</td>
<td></td>
</tr>
<tr>
<td>(File or Real-Time Streaming)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Video Outputs</th>
<th>Qty=1, N</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 / H.265</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vendor Specific Commands

- Vendor specific commands allow a “simpler” implementation
  - Advantages:
    - Simple to architect
    - Simple to implement
  - Challenges:
    - IOCTL path in kernel/driver is not optimized for performance
    - Requires administrative privileges
    - Windows only recently supported vendor specific commands and behavior does not match Linux
    - Is not currently supported by NVMe over Fabrics
## Types of commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Encoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xcoder Open</td>
<td>Open a decoder/encoder instance</td>
</tr>
<tr>
<td>Xcoder Close</td>
<td>Close a decoder instance</td>
</tr>
<tr>
<td>Xcoder Query</td>
<td>Query xcoder for current status</td>
</tr>
<tr>
<td>Xcoder Write</td>
<td>Transfers data from host to codec for decode/encode</td>
</tr>
<tr>
<td>Xcoder Read</td>
<td>Transfers data from codec to host for decode/encode</td>
</tr>
</tbody>
</table>
Example Command Structure

**Command:**

- Opcode
- CDW10
- CDW...
- CDW15

**Response:**

- DW0

- **Xcoder Open – Opcode 0xC1**
  - CDW10: Xcoder ID, configuration data
  - Completion: Xcoder instance

- **Xcoder Write – Opcode -0x83**
  - CDW10: Decoder id, instance, format and stream
  - CDW11: Size of data
IO Commands for video

- IO commands (block level read write) allow high speed access
  - Advantages:
    - Kernel is highly optimized for block level access
    - Very low latency, high priority
  - LBA structure does not align with the structure of our data
    - Need to “hack” the usage for our device, create new definitions of LBA regions, and access patterns
    - No ability within the command to send configuration information
Codec Directly Interacting with Storage

- Challenges with direct interaction:
  - SSD is LBA based but applications are file based
  - How will the internal SSD know the file system of the OS above?
- Without significant changes at the application layer / OS layer direct storage is not practical
- Requires standardization and changes to kernel for optimal solution
Challenges with Memory Management

- Memory movement is the largest contributor of CPU cycles with this solution

- IOCTL Challenges with Memory Management:
  - IOCTL will perform a memory copy if data is not 512 Byte aligned
  - Memory copy consumes significant CPU usage

- Need to optimize overall memory movement from library to host systems
Managing SSD and Video Together

- Video codec and SSD compete for same resources
- Need to guarantee quality of service for both SSD and transcoder
- How to guarantee QOS?
QOS Criteria / Prioritization

- Live stream / real time video
  - Requires uninterrupted service and guaranteed frame rate (i.e. 30fps)
- SSD
  - Requires predictable performance
  - Requires QOS (including 99.99% latency)
- Best effort encoding (Non-Real-time)
Queues for Priority Management

NVMe Host

Non-Real Time Video

Queue C

SSD

Queue B

Live Stream Video

Queue A

NVMe Controller

Resource Monitor
Priority Management Internal

Host

CPU 3
Queue C

CPU 2
Queue B

CPU 1
Queue A

Controller

Resource Monitor
Queues

Admin Queue
Why we need to Standardize Computational Storage?

- Both vendor specific and block command approach with current NVMe is sub-optimal
  - Is a better approach possible?
- Should rethink OS queues for computational storage.
  - Should computational storage elements get a different queue?
- Items like identification, classification provide host system more information
  - Look like a formal device to host with exposed functionality
- Can we build the hooks to allow file based interactions without host interactions?
Scaling Video Encoding in the Cloud
Scaling-out Video Transcoding with NVMe-Over-Fabrics

- Work with proven NVMe and NVMe-oF device drivers
- Composable infrastructure
- Just a Bunch of Transcoders (JBOT)
- Sharing video transcoding resources among servers
Virtualization for Cloud – Hypervisor
Virtualization for Cloud – Containers

Container Engine

Operating System

PCIe NVMe for Encoding

Decoder Module

Encoder Module

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

Operating System

Container Engine

PCle NVMe for Encoding

Decoder Module

Encoder Module

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N

instance 1
instance 2
instance N
Virtualization for Edge with SR-IOV
Share One among Virtual Machines

Host

Hypervisor

Transcoder + SSD

PCIe Physical Function 0
NVMe Controller SSD

Flash Translation Layer
NAND Flash Interface
NAND Flash
Namespace 0

VM1

PCIe Virtual Function(0,1)
NVMe Controller Encoding

Decoder
instance 1
instance 2
instance N

Encoder
instance 1
instance 2
instance N

Namespace 1

VM2

PCIe Virtual Function(0,2)
NVMe Controller Encoding

Decoder
instance 1
instance 2
instance N

Encoder
instance 1
instance 2
instance N

Namespace 2

2019 Storage Developer Conference. © NETINT Technologies. All Rights Reserved.
Questions?