NDA: NVMe CAM attachment

M. Warner Losh

Netflix, Inc.

BSDCan 2016

http://people.freebsd.org/~imp/talks/bsdcan2016/slides.pdf

NETFLIX

How I Learned To Stop Worrying and Love CAM

http://agentpalmer.com/wp-content/uploads/2015/01/Slim-Pickens-riding-the-Bomb.jpg

Netflix

NETFLIX

- Internet Video
- Content Distribution Network (CDN)
- Operating at Scale
- Anticipating the Future

Netflix Open Connect

- According to Sandvine, Netflix streams ˜1/3 of Internet Traffic
- Netflix has own CDN (OpenConnect)
- Streams mutliple Terabits per second

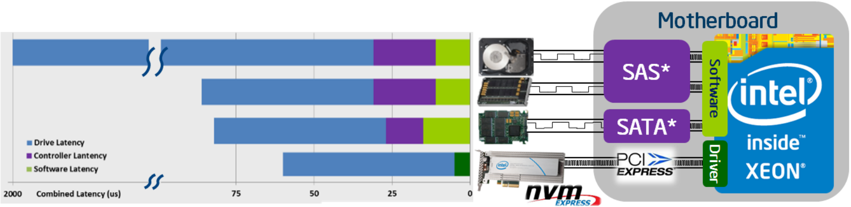
http://blog.streamingmedia.com/wp-content/uploads/2014/02/2013CDNSummit-Keynote-Netflix.pdf

Netflix OCA Trends

- Netflix Storage Appliance (Hard Disk Drive based)
- Netflix Flash Appliance (Solid State Drive based)
- Netflix (and industry) transitioning from SSD to NVMe

http://pcdiy.asus.com/2015/04/asus-z97-x99-motherboards-intel-750-series-nvme-ssds-all-you-need-to-know/

Why Move To NVMe?



- 3rd Generation NVMe designs have $\sim$ 10–15μs latency
- Full Bandwidth (3.9BG/s) from 4-lane PCIe Gen 3 NVMe
- FreeBSD needs optimization (still good at $\sim$ 30μs)

http://itpeernetwork.intel.com/intel-ssd-p3700-series-nvme-efficiency/

Motivation For nda(4) The Why

- Jim Harris of Intel wrote nvme(4) with nvd(4) disk front end
- No easy way to add I/O scheduling to nvd(4) driver
- Netflix buys cheaper drives
    - Lowers cost/GB of storage
    - More drives increases redundancy
    - Low cost drives are quirky
    - Quirkiness gets in the way of smooth, reliable performance
- CAM I/O Scheduler smooths out performance quirks

The How

- FreeBSD I/O stack overview
- CAM basics
- Structure of CAM periph (with examples from nda)
- Structure of CAM XPT (changes needed for nda)
- Structure of CAM SIM (using nvme_sim)
- Wrap up

Outline

FreeBSD I/O Stack

CAM

- Code Flow
- Important Data Structures
- XPT Probe Driver Details
- Periph driver details
- XPT Details
- SIM drivers

Summary

Outline

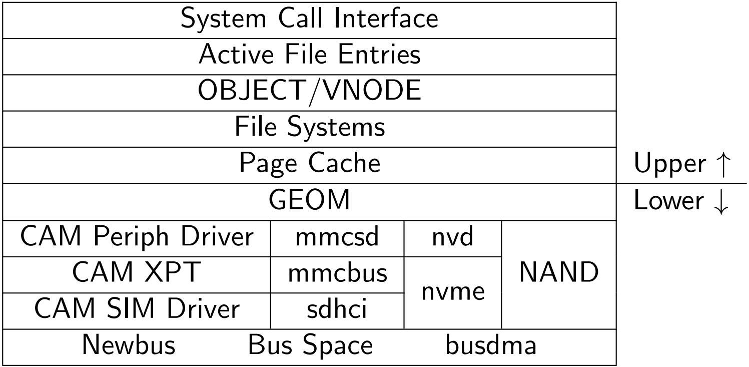
FreeBSD I/O Stack

CAM

- Code Flow
- Important Data Structures
- XPT Probe Driver Details
- Periph driver details
- XPT Details
- SIM drivers

Summary

FreeBSD I/O Stack



System Call Interface

Active File Entries

OBJECT/VNODE

File Systems

Page Cache

Upper ↑

Lower ↓

GEOM

CAM Periph Driver | mmcsd | nvd | NAND

CAM XPT | mmcbus | nvme

CAM SIM Driver | sdhci

Newbus Bus Space busdma

After Figure 7.1 in The Design and Implementation of the FreeBSD Operating System, 2015.

FreeBSD I/O Stack

- Upper half of I/O Stack focus of VM system
  - Buffer cache
  - Memory mapped files / devices
  - Loosely coupled user actions to device action
- GEOM handles partitioning, compression, encryption
  - Filters data (compression, encryption)
  - Muxes Many to one (partitioning)
  - Muxes One to Many (striping / RAID)
  - Limited Scheduling
- CAM handles queuing and scheduling
  - Shapes flows to device
  - Limits requests to number of slots
  - Enforces rules (eg tagged vs non-tagged)
  - Multiplexes shared resources between devices

CAM I/O Scheduler

- Written at Netflix to serve video better during ”fill” periods
- Generic scheduler that allows arbitrary trade offs
- Gathers many real–time statistics on I/O performance
- Knows when drive has become congested

For more information please see my BSDCan 2015 I/O Scheduler talk and paper:

http://people.freebsd.org/~imp/talks/bsdcan2015/slides.pdf

http://people.freebsd.org/~imp/talks/bsdcon2015/paper.pdf

https://www.youtube.com/watch?v=3WqOLolj5EU

Outline

FreeBSD I/O Stack

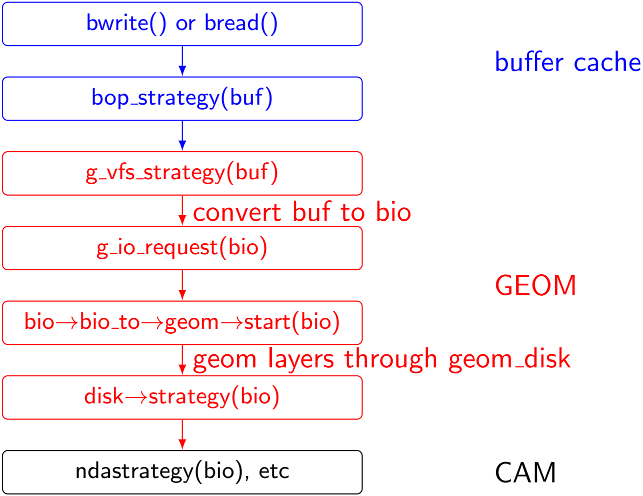
CAM

- Code Flow
- Important Data Structures
- XPT Probe Driver Details
- Periph driver details
- XPT Details
- SIM drivers

Summary

Code Flow Into CAM

File system, pager, swapper, etc



bwrite() or bread()

bop_strategy(buf)

buffer cache

g_vfs_strategy(buf)

convert buf to bio

g_io_request(bio)

GEOM

bio→bio_to→geom→start(bio)

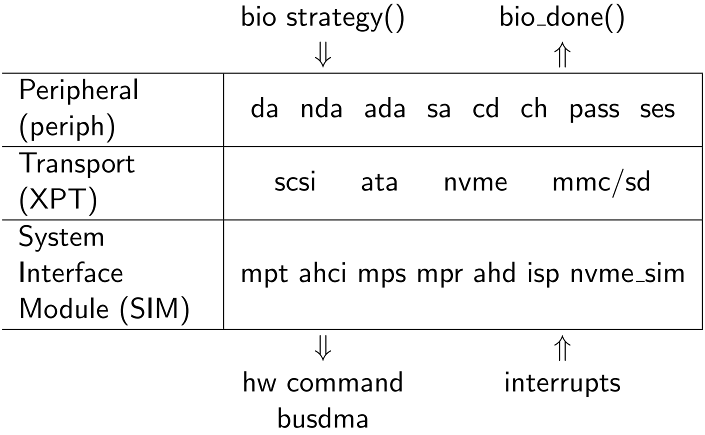
geom layers through geom_disk

disk→strategy(bio)

ndastrategy(bio), etc

CAM

CAM Overview (Simplified)



bio strategy() ⇓ bio_done() ⇑

Peripheral (periph)	da nda ada sa cd ch pass ses
Transport (XPT)	scsi ata nvme mmc/sd
System Interface Module (SIM)	mpt ahci mps mpr ahd isp nvme_sim

⇓ hw command busdma

⇑ interrupts

CAM Command Control Blocks (CCBs)

- Message passing mechanism of CAM
- One giant union of all possible messages
- Some commands immediate, others queued to SIM
- Completion routine to call
- Has completion status

CAM paths

- Describes nodes in the CAM device tree
- Glue that connects periph, xpt and SIM together
- All objects have one or more paths
- Allows multiple periph drivers to attach to the same device
- Includes refcounts on topology

```
# camcontrol devlist
<Micron_M600 MU01> at scbus0 target 2 lun 0 (pass0,da0)
<Micron_M600 MU01> at scbus0 target 3 lun 0 (pass1,da1)
#
```

CAM Async Notifications

- Paths register for an async notification
- Notifications queued
- Used for 'exceptional' events
  - device arrival
  - device departure
  - bus reset
- Sim gets notification to scan for devices
- XPT finds devices and gathers data
- XPT sends `AC_FOUND_DEVICE` and periph drivers attach

CAM devq

- Device queuing mechanism
- One slot per slot on device
- Dynamically resizable
- Controls transactions (CCBs) sent to device
- Can be frozen for error recovery

CAM Peripheral (periph) Drivers

- Participate in device enumeration
- Take block commands via `strategy` function
- Convert to protocol blocks
- Send them to the SIM via the XPT
- Notifies up the stack when SIM signals completion

CAM Transport (xpt) Drivers

- Enumerates devices on transport
- Passes CCB requests from periph to SIM
- Passes CCB completions from SIM to periph
- Answers common CCBs

CAM System Interface Module (SIM) Drivers

- Not SCSI Interface Module
- Accepts protocol blocks from periph driver
- Writes CDB to host adapter
- Sets up busdma for data associated with CCB
- Signals completion of CCB when hw completion interrupt fires
- Answers CCBs about the path to the device (speed, width, mode, etc)

SIM Creation (Done In foo_attach)

- Create a devq with `cam_simq_alloc`
- Create a SIM with `cam_sim_alloc`
  - sim_action routine to receive aysnc CCBs
  - sim_poll routine for dump CCBs
  - devq
  - name / unit #
- Register each bus with `xpt_bus_register`
- Create a path for device enumeration with `xpt_create_path`

But Where Does XPT Get Created?

- `xpt_bus_register` associates the xpt to the bus
- `XPT_PATH_INQ` CCB used to get transport type
- A giant switch statement maps the transport sub-flavors to scsi, ata, or nvme transport.
- No actual xpt object is created, just a pointer to a `struct xpt_xport` of function pointers.

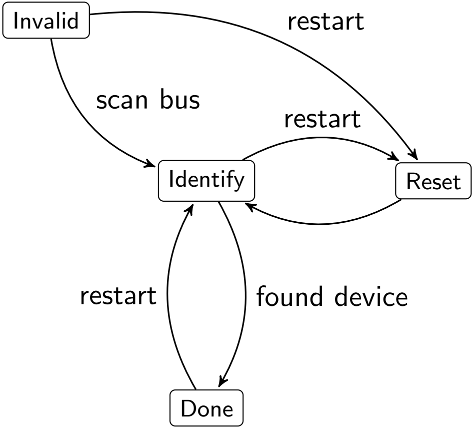
How are periph discovered?

- Each xpt driver registers “probe” device.
- Part of the path creation process queues an `AC_PATHREGISTERED` notification.
- When interrupts enabled, all AC_PATHREGISTERED notifications processed.
- These turn into `XPT_SCAN_BUS` calls.
- After the probe state machine runs for each device found, the xpt layer sends AC_FOUND_DEVICE async message
- Probe devices receive these messages
- They do a `XPT_PATH_INQ` to discover details about the devie.
- If the details match the class of device they service, a new peripheral is added which will handle the device.

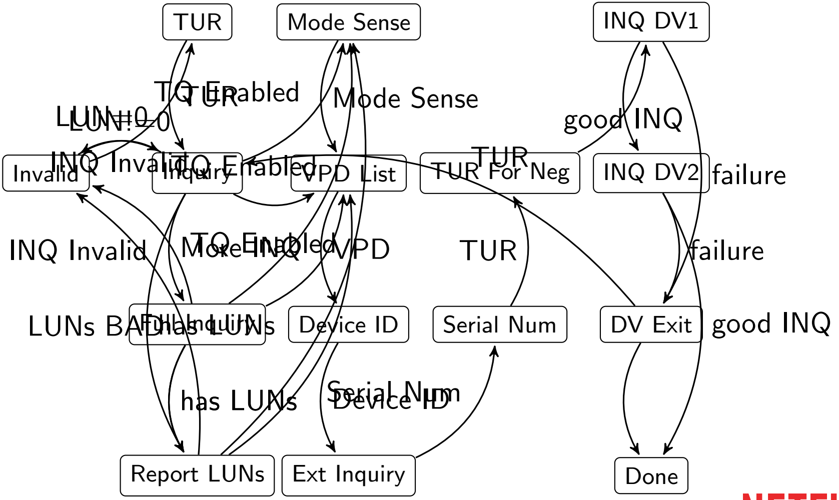
Probe state machine?

- xpt probes can't block
- xpt probes often need to send queries to the device
- State machine sends the query, when it's done the results are looked at an the next state is entered.
- For each state, a command is sent, the completion routine clocks to the next state
- Probing is done when entering the device specific done state.

NVME XPT Probe State Machine



SCSI XPT Probe State Machine



Periph driver attaching

- `AC_DEVICE_FOUND` sent to all devices from xpt probe
- Periph's async handler claims devices (beware: multiple can)
- Periph creates new instance of the device with `cam_periph_alloc`
- device's 'register' routine called
  - Allocates softc
  - Initializes I/O Scheduler
  - Matches quirks and applies them
  - Uses Inquiry or Identify Data to choose flavor of device
  - Negotiates with SIM details of the device
  - Creates disk or char device
  - Saves Identity information
  - Registers async for interesting events
  - calls `xpt_schedule` to get things started

Required Routines

- open – Called when device is opened
- close – Called on last close
- strategy – Called for bio I/O
- start – Called when room for work
- dump – Crash dumps
- getattr – Get attributes
- gone – Drive has departed
- done – CCB has finished

xpt_schedule

- Checks to see if there's room in devq
- If there is, it allocates a CCB and calls periph's `start` routine
- Can also make sure there's room in the simq for SIMs with concurrent transaction limitations beyond those of the device.

xpt_action

- Pushes the I/O to XPT or SIM

xpt_done

- Finishes a CCB up and calls its completion routine
- Also calls `xpt_schedule`
- Requeue it if there's errors

Strategy

- System presents I/O to driver in a `struct bio`
- Driver queues the I/O
- Drive calls `xpt_schedule` to maybe do I/O

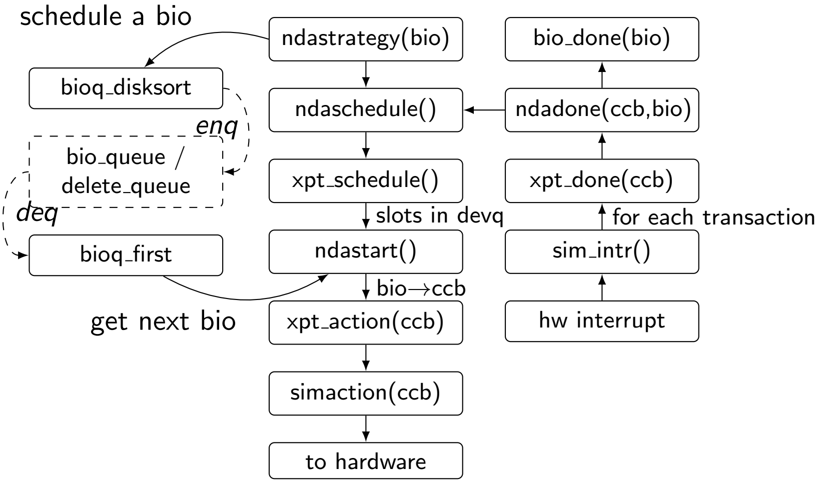
Start

- You know you have a slot
- Must either complete CCB or submit it to SIM for I/O
- Must call `xpt_schedule` at the end
- Restrictions on I/O enforced here (eg, no TRIM while other I/O outstanding, etc)

Done

- Called by the SIM as part of xpt_done processing after it's processed the I/O
- Responsible for completing the bio up the stack
- Calls xpt_schedule since there's now a slot in drive that's opened up.

CAM I/O Code flow



SIM Routines

- simaction
- simpoll
- IRQ or Timer for completions
- created in foo_attach

simaction

- Processes the CCBs queued with `xpt_action`
- Queued CCBs return w/o setting the status
- Immediate CCBs do the action and set status

simpoll

- Checks to see if the CCB has completed
- Called only during dumping when interrupts are disabled

sim IRQ

- Called when an I/O completes
- Finishes the CCB associated with the I/O with `xpt_done`

Outline

FreeBSD I/O Stack

CAM

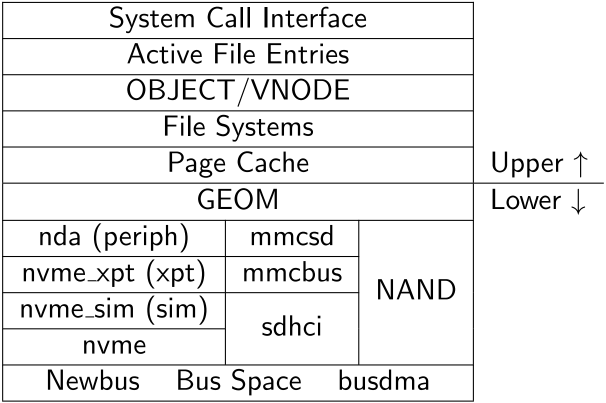
- Code Flow
- Important Data Structures
- XPT Probe Driver Details
- Periph driver details
- XPT Details
- SIM drivers

Summary

Key Points

- XPT means Transport
- SIM scans the bus for devices (explicitly, or in response to `AC_PATHREGISTERED`
- XPT probes device using special “probe” devices
- XPT probing state machine driven
- Once probed, XPT tells periph drivers by sending `AC_FOUND_DEVICE`
- periph drivers create instances based on discovered paths (may be many to 1)
- CCBs drive everything

FreeBSD I/O Stack nda World



<table>
<tr><td colspan="3">System Call Interface</td><td></td></tr>
<tr><td colspan="3">Active File Entries</td><td></td></tr>
<tr><td colspan="3">OBJECT/VNODE</td><td></td></tr>
<tr><td colspan="3">File Systems</td><td></td></tr>
<tr><td colspan="3">Page Cache</td><td>Upper ↑</td></tr>
<tr><td colspan="3">GEOM</td><td>Lower ↓</td></tr>
<tr><td>nda (periph)</td><td>mmcsd</td><td rowspan="4">NAND</td><td></td></tr>
<tr><td>nvme_xpt (xpt)</td><td>mmcbus</td><td></td></tr>
<tr><td>nvme_sim (sim)</td><td rowspan="2">sdhci</td><td></td></tr>
<tr><td>nvme</td><td></td></tr>
<tr><td colspan="3">Newbus Bus Space busdma</td><td></td></tr>
</table>

After Figure 7.1 in The Design and Implementation of the FreeBSD Operating System, 2015.

Questions

Questions?
Comments?

Warner Losh

wlosh@netflix.com
imp@FreeBSD.org

http://people.freebsd.org/~imp/talks/bsdcon2016/slides.pdf