Best Practices for OpenZFS
L2ARC in the Era of NVMe

Ryan McKenzie
iXsystems
Brief Overview of OpenZFS ARC / L2ARC
- Key Performance Factors
- Existing “Best Practices” for L2ARC
  - Rules of Thumb, Tribal Knowledge, etc.
- NVMe as L2ARC
  - Testing and Results
- Revised “Best Practices”
ARC Overview

- **Adaptive Replacement Cache**
- Resides in system memory
- Shared by all pools
- Used to store/cache:
  - All incoming data
  - “Hottest” data and Metadata (a tunable ratio)
- Balances between
  - Most Frequently Used (MFU)
  - Most Recently Used (MRU)
L2ARC Overview

- Level 2 Adaptive Replacement Cache
- Resides on one or more storage devices
  - Usually Flash
- Device(s) added to pool
  - Only services data held by that pool
- Used to store/cache:
  - “Warm” data and metadata (about to be evicted from ARC)
  - Indexes to L2ARC blocks stored in ARC headers

FreeBSD + OpenZFS Server

ARC

NIC/HBA

OpenZFS

SLOG vdevs

Data vdevs

L2ARC vdevs

zpool
### ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally
ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally
ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally
ZFS Writes

- All writes go through ARC, written blocks are “dirty” until on stable storage
  - async write ACKs immediately
  - sync write copied to ZIL/SLOG then ACKs
  - copied to data vdev in TXG
- When no longer dirty, written blocks stay in ARC and move through MRU/MFU lists normally
ZFS Reads

- Requested block in ARC
  - Respond with block from ARC

- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC

- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - “ARC PROMOTE”: read block(s) stay in ARC and move through MRU/MFU lists normally
ZFS Reads

- Requested block in ARC
  - Respond with block from ARC
- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC
- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - “ARC PROMOTE”: read block(s) stay in ARC and move through MRU/MFU lists normally
ZFS Reads

- Requested block in ARC
  - Respond with block from ARC
- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC
- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - “ARC PROMOTE”: read block(s) stay in ARC and move through MRU/MFU lists normally

OpenZFS

FreeBSD + OpenZFS Server

ARC

Data vdevs

SLOG vdevs

L2ARC vdevs

NIC/HBA

Prefetch more blocks to avoid another read miss?
ZFS Reads

- Requested block in ARC
  - Respond with block from ARC
- Requested block in L2
  - Get index to L2ARC block (from ARC)
  - Respond with block from L2ARC
- Otherwise: read miss
  - Copy block from data vdev to ARC
  - Respond with block from ARC
  - “ARC PROMOTE”: read block(s) stay in ARC and move through MRU/MFU lists normally
ARC Reclaim

- Called to maintain/make room in ARC for incoming writes

L2ARC Feed

- Asynchronous of ARC Reclaim to speed up ZFS write ACKs
- Periodically scans tails of MRU/MFU
  - Copies blocks from ARC to L2ARC
  - Index to L2ARC block resides in ARC headers
  - Feed rate is tunable
- Writes to L2 device(s) sequentially (rnd robin)
**ARC Reclaim**

- Called to maintain/make room in ARC for incoming writes

**L2ARC Feed**

- Asynchronous of ARC Reclaim to speed up ZFS write ACKs
- Periodically scans tails of MRU/MFU
  - Copies blocks from ARC to L2ARC
  - Index to L2ARC block resides in ARC headers
  - Feed rate is tunable
- Writes to L2 device(s) sequentially (rnd robin)
L2ARC “Reclaim”

- Not really a concept of reclaim
- When L2ARC device(s) get full (sequentially at the “end”), L2ARC Feed just starts over at the “beginning”
  - Overwrites whatever was there before and updates index in ARC header
- If an L2ARC block is written, ARC will handle the write of the dirty block, L2ARC block is stale and the index is dropped
L2ARC “Reclaim”

- Not really a concept of reclaim
- When L2ARC device(s) get full (sequentially at the “end”), L2ARC Feed just starts over at the “beginning”
  - Overwrites whatever was there before and updates index in ARC header
- If an L2ARC block is written, ARC will handle the write of the dirty block, **L2ARC block is stale and the index is dropped**
Some Notes:

**ARC / L2ARC Architecture**

- ARC/L2 “Blocks” are variable size:
  - =volblock size for zvol data blocks
  - =record size for dataset data blocks
  - =indirect block size for metadata blocks

- Smaller volblock/record sizes yield more metadata blocks (overhead) in the system
  - may need to tune metadata % of ARC
Some Notes:
ARC / L2ARC Architecture

- Blocks get into ARC via any ZFS write or by demand/prefetch on ZFS read miss
- Blocks cannot get into L2ARC unless they are in ARC first (primary/secondary cache settings)
  - CONFIGURATION PITFALL
- L2ARC is not persistent
  - Blocks are persistent on the L2ARC device(s) but the indexes to them are lost if the ARC headers are lost (main memory)
Some Notes:
ARC / L2ARC Architecture

- Write-heavy workloads can “churn” tail of the ARC MRU list quickly
- Prefetch-heavy workloads can “scan” tail of the ARC MRU list quickly
- In both cases…
  - ARC MFU bias maintains integrity of cache
  - L2ARC Feed “misses” many blocks
  - Blocks that L2ARC does feed may not be hit again - “Wasted L2ARC Feed”
Agenda

✓ Brief Overview of OpenZFS ARC / L2ARC
❏ Key Performance Factors
❏ Existing “Best Practices” for L2ARC
    ❏ Rules of Thumb, Tribal Knowledge, etc.
❏ NVMe as L2ARC
    ❏ Testing and Results
❏ Revised “Best Practices”
Key Performance Factors

- Random Read-Heavy Workload
  - L2ARC designed for this, expect very effective once warmed
- Sequential Read-Heavy Workload
  - L2ARC not originally designed for this, but specifically noted that it might work with “future SSDs or other storage tech.”
  - Let’s revisit this with NVMe!
Key Performance Factors

- Write-Heavy Workload (random or sequential)
  - ARC MRU churn, memory pressure, L2ARC never really warmed because many L2ARC blocks get invalidated/dirty
    - “Wasted” L2ARC Feeds
  - ARC and L2ARC are designed to primarily benefit reads
  - Design expectation is “do no harm”, but there may be a constant background performance impact of L2ARC feed
Key Performance Factors

- Small ADS (relative to ARC and L2ARC)
  - Higher possibility for L2ARC to fully warm and for L2ARC Feed to slow down
- Large ADS (relative to ARC and L2ARC)
  - Higher possibility for constant background L2ARC Feed
Agenda

✓ Brief Overview of OpenZFS ARC / L2ARC
✓ Key Performance Factors
☐ Existing “Best Practices” for L2ARC
   ☐ Rules of Thumb, Tribal Knowledge, etc.
☐ NVMe as L2ARC
   ☐ Testing and Results
☐ Revised “Best Practices”
Existing “Best Practices” for L2ARC

- Do not use for sequential workloads
- Segregated pools and/or datasets
  - At config time
- “l2arc noprefetch” setting
  - Per Pool, Runtime tunable
- “secondarycache=metadata” setting
  - primarycache and secondarycache are per dataset, Runtime tunable
  - all, metadata, none
Existing “Best Practices” for L2ARC

- Do not use for sequential workloads
  - Segregated pools
    - Global, At config time
  - “l2arc noprefetch” setting
    - Per Pool, Runtime tunable
  - “secondarycache=metadata” setting
    - primarycache and secondarycache are per dataset, Runtime tunable
    - all, metadata, none
Existing “Best Practices” for L2ARC

- All are more or less plausible given our review of the design of L2ARC
- Time for some testing and validation!
Agenda

✓ Brief Overview of OpenZFS ARC / L2ARC
✓ Key Performance Factors
✓ Existing “Best Practices” for L2ARC
  ✓ Rules of Thumb, Tribal Knowledge, etc.
❏ NVMe as L2ARC
❏ Testing and Validation
❏ Revised “Best Practices”
Solution Under Test

- FreeBSD+OpenZFS based storage server
  - 512G main memory
  - 4x 1.6T NVMe drives installed
    - Enterprise-grade dual port PCIeG3
      - Just saying “NVMe” isn’t enough...
    - Tested in various L2ARC configurations
  - 142 HDDs in 71 mirror vdevs
  - 1x 100GbE NIC (optical)
  - Exporting 12x iSCSI LUNs (pre-filled)
  - 100GiB of active data per LUN (1.2TiB total ADS)
Solution Under Test

- ADS far too big for ARC
  - Will “fit into” any of our tested L2ARC configurations
- Preconditioned ARC/L2ARC to be “fully warm” before measurement (ARC size + L2 size >= 90% ADS)
  - Possibly some blocks aren’t yet in cache, esp. w/random
- System reboot between test runs (L2ARC configs)
  - Validated “balanced” read and write activity across the L2ARC devices
Solution Under Test

- **Load Generation**
  - 12x Physical CentOS 7.4 Clients (32G main memory)
  - 1x 10Gbe NIC (optical)
  - Map one LUN per client
  - VDBENCH to generate synthetic workloads
    - In-House “Active Benchmarking” Automation
  - Direct IO to avoid client cache

- **Network (100GbE Juniper)**
  - Server -> direct connect fiber via 100Gbe QSFP
  - Clients -> aggregated in groups of 4 via 40Gbe QSFP
Random Read-Heavy

- 4K IO Size, Pure Random, 100% Read
OPS Achieved - Thread Scaling

More than 6x the OPS compared to NOL2ARC

1xNVMe and 2xNVMe performance not scaling as expected, need to investigate...
Avg. R/T (ms) - Thread Scaling

Reading from NVMe
L2ARC has benefit of sub-ms latency

More L2ARC devices show latency benefit at higher thread counts
NVMe Activity - 1 in L2ARC
16 Thread Load Point (“best”)

Client Aggregate Read BW ~200MiB/s
1xNVMe servicing ~100MiB/s

Still some errant L2ARC Feeding - synthetic random workload
NVMe Activity - 2 in L2ARC
16 Thread Load Point ("best")

Client Aggregate Read BW ~200MiB/s
2xNVMe servicing a bit more than 100MiB/s

Still some errant L2ARC Feeding - synthetic random workload
NVMe Activity - 4 in L2ARC

16 Thread Load Point ("best")

Client Aggregate Read BW ~450MiB/s

4xNVMe servicing a bit more than 250MiB/s

Very few errant L2ARC Feeding - synthetic random workload

Even less than before - L2ARC feed rate is per device...
L2ARC Effectiveness

OPS vs. Response Time

16 Threads Measurement Period ("best")

<table>
<thead>
<tr>
<th></th>
<th>NOL2ARC</th>
<th>4xNVMe</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPS Achieved</td>
<td>14610.2333</td>
<td>112526.2694</td>
</tr>
<tr>
<td>% increase OPS</td>
<td></td>
<td>670.19%</td>
</tr>
<tr>
<td>Avg. R/T (ms)</td>
<td>13.139</td>
<td>1.7057</td>
</tr>
<tr>
<td>% decrease R/T</td>
<td></td>
<td>-670.30%</td>
</tr>
</tbody>
</table>
"Moving the Bottleneck"
A Cautionary Tale

CPU no longer waiting on high latency disk reads, is now copying buffers like crazy!
Sequential Read-Heavy

- 128K IO Size, Sequential, 100% Read
- 1xNVMe results omitted (invalid)
  ○ No system time available to re-run
**Bandwidth - Thread Scaling**

Roughly 3x the bandwidth compared to NOL2ARC

L2ARC benefit becomes more pronounced at higher thread counts where sequential read workload appears like random reads to the server.
Avg. R/T (ms) - Thread Scaling

Sequential 128 KiB 100% Read - resp

Reading from NVMe L2ARC has significant latency benefit even with larger reads.
Prefetch ARC reads consist of about 20% of Total ARC reads.

So ZFS only marks about 20% of our accesses as “streaming”.

As we add more threads/clients the sequential read requests from them begin to arrive interleaved randomly at the server.
NVMe Activity - 2 in L2ARC
16 Thread Load Point ("best")

Client Aggregate Read
BW ~3,000MiB/s
2xNVMe servicing
~2,800MiB/s

Almost no L2ARC Feeding
NVMe Activity - 4 in L2ARC
16 Thread Load Point ("best")

Client Aggregate Read
BW ~3,300MiB/s

4xNVMe servicing a bit more than ~3,000MiB/s

Almost no L2ARC Feeding
L2ARC Effectiveness

Bandwidth vs. Response Time

16 Threads Measurement Period ("best")

<table>
<thead>
<tr>
<th></th>
<th>NOL2ARC</th>
<th>4xNVMe</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiB/s Achieved</td>
<td>1155.3823</td>
<td>3290.7132</td>
</tr>
<tr>
<td>% increase MiB/s</td>
<td>184.82%</td>
<td></td>
</tr>
<tr>
<td>Avg. R/T (ms)</td>
<td>20.7702</td>
<td>7.2923</td>
</tr>
<tr>
<td>% decrease R/T</td>
<td></td>
<td>-64.89%</td>
</tr>
</tbody>
</table>

64 Threads Measurement Period ("max")

<table>
<thead>
<tr>
<th></th>
<th>NOL2ARC</th>
<th>4xNVMe</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiB/s Achieved</td>
<td>1545.3837</td>
<td>4852.7472</td>
</tr>
<tr>
<td>% increase MiB/s</td>
<td>214.02%</td>
<td></td>
</tr>
<tr>
<td>Avg. R/T (ms)</td>
<td>62.1173</td>
<td>19.7816</td>
</tr>
<tr>
<td>% decrease R/T</td>
<td></td>
<td>-68.15%</td>
</tr>
</tbody>
</table>
“All my data fits in L2ARC!”
A Cautionary Tale

Different Test Case (Smaller Server) - 128k Sequential Reads
Same iSCSI Setup - 480GiBADS - 128GiB RAM - 1x 400GiB SSD L2ARC

Having a “fully warm” L2ARC is only a good thing if your cache device(s) are “fast enough”
L2ARC with Sequential Reads

- vfs.zfs.l2arc_noprefetch=0
  - As tested on this server
  - ⇒ blocks that were put into ARC by prefetcher are eligible for L2ARC
- Let’s test vfs.zfs.l2arc_noprefetch=1
  - ⇒ blocks that were put into ARC by prefetcher are NOT L2ARC-eligible
L2ARC with Sequential Reads

- Let’s test secondarycache=metadata
  - Only metadata blocks from ARC are eligible for L2ARC
- Both of these are strategies to keep L2ARC from feeding certain blocks
  - With older/slower/fewer L2ARC devices this made sense...
But if your cache device(s) are good, it makes no sense to keep blocks out of the L2ARC.
Summary for Write-Heavy Workloads

- **4K IO Size, Pure Random, 100% Write**
  - L2ARC constantly feeding - writes causing memory pressure
  - Up to 20% reduction in OPS vs. NOL2ARC
  - Not worth tuning for mitigation
    - Pure writes are rare in practice
    - Trying a 50/50 read write mix (which is still more writes than typical random small block use cases) and all L2ARC configs beat NOL2ARC

- **128K IO Size, Sequential, 100% Write**
  - Bandwidth differences of less than 10% +/- over NOL2ARC
    - “In the Noise” - meets design goal of “do no harm”
    - Trying a 50/50 mix again shows benefit on all L2 configs
Agenda

✓ Brief Overview of OpenZFS ARC / L2ARC
✓ Key Performance Factors
✓ Existing “Best Practices” for L2ARC
  ✓ Rules of Thumb, Tribal Knowledge, etc.
✓ NVMe as L2ARC
  ✓ Testing and Validation

❖ Revised “Best Practices”
Key Sizing/Config Metrics

● Key Ratio 1:
  ADS / (ARC Max + L2ARC Size)
  ○ <= 1 means your workload’s active dataset will likely be persistently cached in ARC and/or L2ARC

● Key Ratio 2:
  (aggregate bandwidth of L2ARC devices) / (agg. bw. of data vdevs)
  ○ > 1 means your L2ARC can stream sequential data faster than all the data vdevs in your pool
L2ARC Device Selection

● Type ⇒ Bandwidth/OPS and Latency Capabilities
  ○ Consider devices that fit your budget and use case.
  ○ Random Read Heavy Workloads can benefit for almost any L2ARC device that is faster than the devices in your data vdevs
  ○ Sequential/Streaming Workloads will need very fast low latency devices

● Segregated pools vs. mixed use pool with segregated datasets
L2ARC Device Selection

- **Capacity**
  - Larger L2ARC devices can hold more of your active dataset, but will take longer to “fully warm”
    - For Random Read-Heavy Workloads, GO BIG
    - For Sequential Read-Heavy Workloads, only go big if your devices will have enough bandwidth to benefit vs. your data vdevs
  - Indexes to L2ARC data reside in ARC headers
    - 96 bytes per block (reclaimed from ARC)
    - 256 bytes per block (still in ARC also)
    - Run “top”, referred to as “headers”, small ARC and HUGE L2ARC is probably not a good idea…
L2ARC Device Selection

- Device Count
  - Our testing shows that multiple L2ARC devices will “share the load” of servicing reads, helping with latency
  - L2ARC Feeding rate is per device, so more devices can help warm faster (or have a higher performance impact if constantly feeding)
L2ARC Feeding

● What to Feed?
  ○ Random Read-Heavy Workload: feed everything
  ○ Sequential Read-Heavy Workloads
    ■ “Slow L2”: Demand Feed - l2arc_noprefetch=1 (global)
    ■ “Fast L2”: feed everything
  ○ Write-Heavy Workloads
    ■ “Slow L2”: Feed Nothing - segregated pool
      ● or secondarycache=none (per dataset)
    ■ “Fast L2”: Feed Metadata - secondarycache=metadata (per dataset)
      ● Avoids constant L2 writes but benefits metadata reads
The Best Practice

● Know your workload
  ○ For customers, know your top level application(s) and if possible, underlying factors such as access pattern, block size, and active dataset size
  ○ For vendors, get as much of the above information as you can

● Know your solution
  ○ For vendors, know how your solution’s architecture and features interact with different customer workloads
    ■ (S31) So, You Want to Build a Storage Performance Testing Lab?
      - Nick Principe
  ○ For customers, familiarize yourself with this from the vendor’s sales engineers and support personnel that you work with
Big Picture

- With enterprise grade NVMe devices as L2ARC, we have reached the “future” the Brendan Gregg mentions in his blog and in the arc.c code:
  - Namely, L2ARC can now be an effective tool for improving the performance of streaming workloads
Agenda

✓ Brief Overview of OpenZFS ARC / L2ARC
✓ Key Performance Factors
✓ Existing “Best Practices” for L2ARC
  ✓ Rules of Thumb, Tribal Knowledge, etc.
✓ NVMe as L2ARC
  ✓ Testing and Validation
✓ Revised “Best Practices”
Thank You!

Questions and Discussion

Ryan McKenzie
iXsystems
References

- module/zfs/arc.c
  - https://github.com/zfsonlinux/zfs

- “ARC: A Self-Tuning, Low Overhead Replacement Cache”
  - Nimrod Megiddo and Dharmendra S. Modha

- “Activity of the ZFS ARC”, Brendan Gregg’s Blog (January 9, 2012)

- “ZFS L2ARC”, Brendan Gregg’s Blog (July 22, 2008)

- “FreeBSD Mastery: Advanced ZFS”
  - Allan Jude and Michael W. Lucas
  - Tilted Windmill Press, 2016