SNIA Education

ASPECTS OF DEDUPLICATION

Dominic Kay, Oracle
Mark Maybee, Oracle

SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of the SNIA Education Committee.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

Abstract

Education SNIA

- This tutorial will focus on block-level deduplication. While conceptually simple, an implementation can be quite complex as it must address multiple issues:
  - scalability - when the lookup table no longer fits in memory.
  - performance - impact of table lookups.
  - space accounting – who owns a deduped block?
  - administration - keeping the model simple.
- This tutorial will also
  - cover expanding the notion of deduplication beyond persistent storage devices to include in-memory and over-the-wire deduplication.

Deduplication Defined

- Stores first unique domain, additional copies increase reference counts
- Improves storage efficiency
- Historically used for backups
  - Now moving into archiving and primary storage
- Leads to reduced redundancy
  - A single corrupted block can have greater impact
- Can be done in-line or in post processing
- Can be done at the file or block level

Other Tutorials

This tutorial covers research & work in progress. It focuses on some of the specific features and implementation details of ZFS. For a grounding in deduplication check out this SNIA Tutorial:

Understanding Data Deduplication (Thomas Rivera)

ZFS Overview

- Pooled storage
  - Completely eliminates the antique notion of volumes
  - Does for storage what VM did for memory
- Transactional object system
  - Always consistent on disk – no fsck, ever
  - Applied universally – file, block, iSCSI, swap ...
- Provable end-to-end data integrity
  - Detects and corrects silent data corruption
  - Historically considered “too expensive” – no longer true
- Simple administration
  - Concisely express your intent

FS/Volume vs. Pooled Storage

Education SNIA

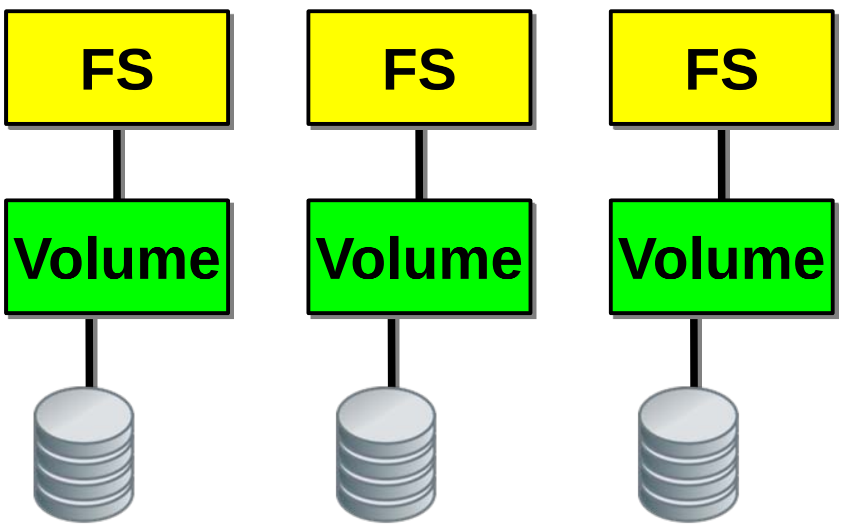
Abstraction: Virtual disk

Partition/volume for each FS

Grow/shrink by hand

Each FS has limited bandwidth

Storage is fragmented, stranded



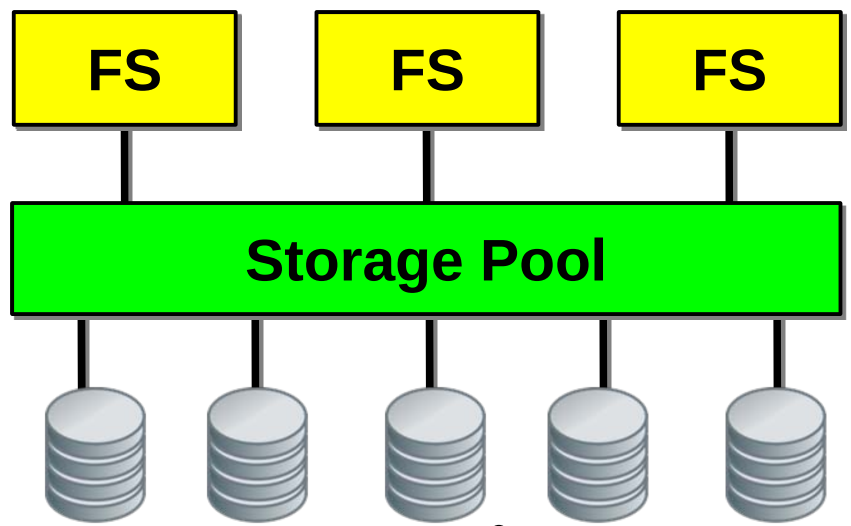
Abstraction: malloc/free

No partitions to manage

Grow/shrink automatically

All bandwidth available

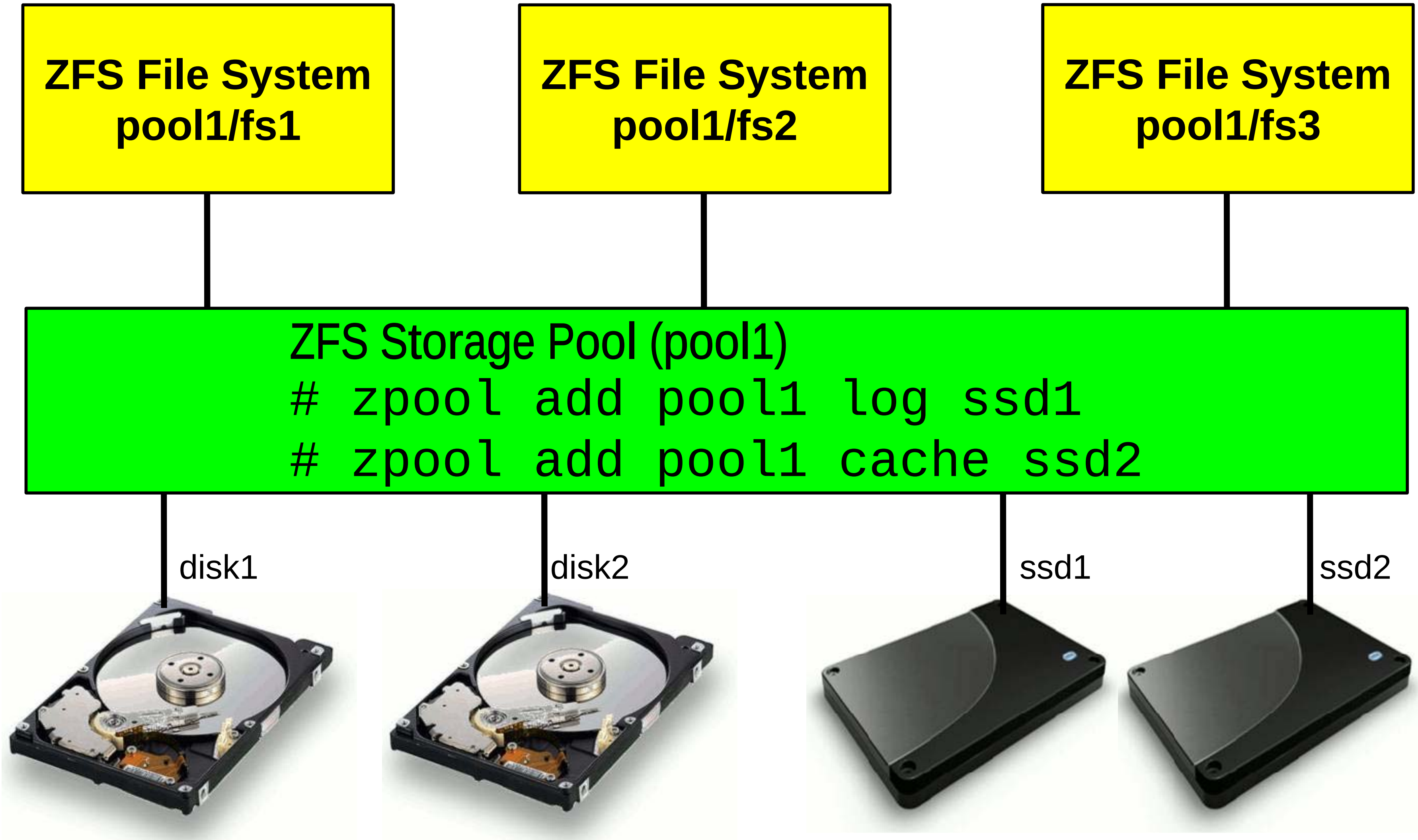
All storage in the pool shared



Administrative Interfaces

- Design goals of ZFS dictate simple admin where possible.
- The pool/filesystem model dictates the administrative interface:
  - zpool create pool1 mirror disk1 disk2
  - zfs set dedup=<on | off | checksum>[,verify] <filesystem,volume>
  - zfs get dedup <filesystem, volume>
  - zpool get dedupratio pool
- This model allows us to deal with mixed mode data stores.
  - Can be requested at the dataset level
  - Can be applied to any dataset type in the pool
  - Applied across all selected datasets in the pool

ZFS & SSD: Hybrid Storage Pool



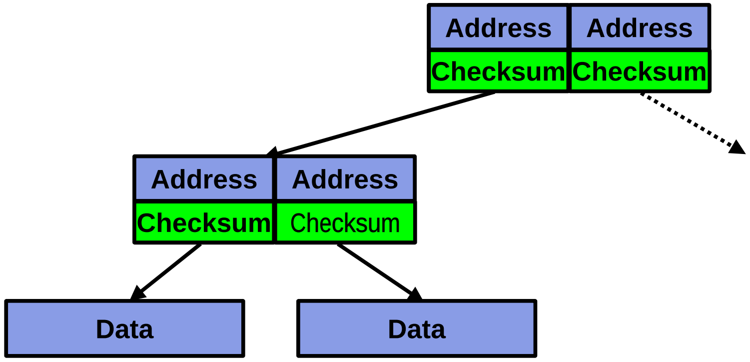
Dedup Table and its Placement

Education SNIA

- Most implementations keep the table in main memory
  - Keeps table lookups fast
  - Simplifies the implementation
  - Constrains the amount of "dedupable" data
    - Once table is full, new data blocks are not deduped
- ZFS allows dedup table to grow
  - Eventually may no longer fit in memory
  - Significant performance-vs-space tradeoff:
    - All data is deduplicated
    - May require a read to perform a table lookup
  - SDDs (as secondary cache) help to mitigate the impact
    - Lookup that misses in memory reads from SSD
    - Much faster than rotating disk

ZFS Data Authentication

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire storage pool is a self-validating Merkle tree
- ZFS validates the entire I/O path
  - DMA parity errors
  - Driver bugs
  - Accidental overwrite
  - Misdirected reads and writes
  - Bit rot
  - Phantom writes



Checksums

- The data validation checksums drive the deduplication table.

- `zfs set dedup=<on|off|checksum>[,verify]`
  - The acceptable values for the dedup property are as follows:
  - off (the default)
  - on (see below)
  - on,verify
  - sha256
  - sha256,verify
  - fletcher4,verify
  - fletcher2,verify

Ditto Blocks

- Data replication above and beyond RAID
  - Each logical block can have up to three physical blocks
    - Different devices whenever possible
    - Different places on the same device otherwise (e.g. laptop drive)
  - All ZFS metadata 2+ copies
    - Small cost in latency and bandwidth (metadata ≈ 1% of data)
  - Explicitly settable for precious user data
- ZFS Detects and corrects silent data corruption
  - In a multi-disk pool, survives any non-consecutive disk failures
  - In a single-disk pool, survives loss of up to 1/8 of the platter

Ditto Blocks & Deduplication

- Automatic-ditto data protection
- Mitigates data redundancy concerns associated with deduplication
- Creates an extra copy of the block based on reference count threshold
- Setting the automatic-ditto threshold

```
# zpool set dedupditto=200 tank
```

Variable Sized Block

ZFS supports blocks sizes from 512 bytes to 128K bytes

- Uses block size appropriate for data
  - Small blocks for small files
    - just large enough to accommodate file content
  - Large blocks for large files
- Larger blocks make dedup table more efficient
  - Better table-entry to disk space ratio
  - More deduped data can be managed by a smaller table
  - Smaller memory footprint

ZFS Compression

- The tool for space optimization prior to deduplication.
  - Leveraged to minimize zfs lookup table size
  - Important for non-dedupable meta data
- Several algorithms available
  - lzjb (default)
  - gzip-[1-9]
  - zle
- set and get (compression ratio) via filesystem properties.
  - zfs set compression=[on | off | lzjb | gzip | zle]
  - zfs get compressratio
- Apllies to data written after property is set and usual YMMV rules apply.

Dedup Integrates with Compression

Education SNIA

```
# zdb -DD tank
DDT-sha256-zap-duplicate: 110173 entries, size 295 on disk, 153 in core
DDT-sha256-zap-unique: 302 entries, size 42194 on disk, 52827 in core

DDT histogram (aggregated over all DDTs):

bucket              allocated                       referenced
______   ______________________________   ______________________________
refcnt   blocks   LSIZE   PSIZE   DSIZE   blocks   LSIZE   PSIZE   DSIZE
------   ------   -----   -----   -----   ------   -----   -----   -----
     1      302   7.26M   4.24M   4.24M      302   7.26M   4.24M   4.24M
     2     103K   1.12G    712M    712M     216K   2.64G   1.62G   1.62G
     4    3.11K   30.0M   17.1M   17.1M    14.5K    168M   95.2M   95.2M
     8      503   11.6M   6.16M   6.16M    4.83K    129M   68.9M   68.9M
    16      100   4.22M   1.92M   1.92M    2.14K    101M   45.8M   45.8M
    32      548   65.7M   34.0M   34.0M    22.4K   2.69G   1.40G   1.40G
    64      169   20.8M   11.2M   11.2M    13.8K   1.70G    940M    940M
 Total     108K   1.25G    787M    787M     274K   7.43G   4.15G   4.15G

dedup = 5.40, compress = 1.79, copies = 1.00, dedup * compress / copies = 9.67
```

Sync & Async Deduplication

- Synchronous deduplication happens "on the fly"
  - Write operation is bypassed if we hit in dedup table
  - Dedup table expanded when we miss
  - Can improve write performance if we get lots of hits
  - Can decrease write performance when we miss
- Async deduplication happens in the background
  - Improves storage efficiency
  - Often used in backup systems
  - Background task can impact performance of foreground activity

Dedup over the wire

- ZFS **send** syntax
  - `zfs send -D[vRp] [-[i|I] snapshot] snapshot`
  - -D flag requests deduplication in stream
- Applies the concept of on-disk deduplication to a backup stream.
  - Send first copy of the data, just send refs after
  - Only dedup's the data within the stream
- Concept can be extended to remote replication
  - Only send a ref if a data block is already present in the remote replica
  - Requires tight integration: resend if block is no longer present in replica
    - e.g., was present in a snapshot that has been deleted on the replica

In-memory Deduplication

- Keep only a single copy of data in cache for any block
  - Mostly just “falls out” from on-disk dedup
    - Blocks already share a common address
  - Tricky to manage multiple refs on a single cache block
  - Make copies **only** when referencer wants to modify content
- Special case the “zero block”
  - Most common block of data is empty
  - Represents a “hole” in a file, so does not need to dedup on disk
  - Map all such refs in memory to a single empty data block in cache
    - Use max file system block size

Q&A / Feedback

Please send any questions or comments on this presentation to SNIA: trackdatamanagement@snia.org

Many thanks to the following individuals for their contributions to this tutorial.

Jeff Bonwick	Bill Moore
Joel Buckley	Cindy Swearingen
Brenden Gregg	George Wilson
Adam Leventhal	Jeffrey Wright