A Prototype Implementation of the T10 OSD Specification on OpenSolaris

Ed Nadolski, LSI Logic, Inc.
ed.nadolski@lsi.com

May 2007
Overview

• Original Motivation: Build a Petascale HPC system
  – DARPA research funded project
• Object Storage considered essential to achieve IO bandwidth and scalability requirements
• Standards-based interfaces considered highly desirable
Review: Block I/O Path in Solaris

System Call/VFS
- QFS
- ZFS
- UFS

Block Interface
- sd
- st

SCSA/MPXIO
- Leadville Fibre Channel
- iSCSI
- mpt

User
Kernel
Filesystem Layer
Target Driver Layer
HBA Driver Layer
Storage Devices
T10 OSD Support: Primary Components

• Object-aware QFS filesystem
  – Understands T10 OSD semantics: Objects, Partitions, Attributes, etc.
  – Distributed and Scalable: Support many parallel clients and storage devices
• New OSD I/O Interface
  – Replaces `bdev_strategy()` for I/O requests
  – Filesystem to target driver communication path
• New target driver: `sosd`
  – Understands the T10 OSD CDB, sense data, etc.
• SCSA and HBA driver enhancements
  – Support large CDBs, bidirectional data transfers (BIDI), descriptor sense data
  – Support multisegment DATA IN and DATA OUT buffers
    • Not handled in current implementation
• Link generator to build Solaris device nodes.
  – Each OSD LUN maps to a single minor device
  – Character device node (for now)
  – Does not present ‘partition’ semantics
**OSD I/O Interface**

- **Goal:** Present T10 OSD command level semantics to the filesystem
  - CREATE, REMOVE, READ, WRITE, APPEND, SET ATTRIBUTES, etc.
  - Don’t limit an object-aware filesystem to block semantics.
- **New data structure:** `osd_iotask(9S)`
  - Contains all information needed to describe an OSD command
  - Analogous to `buf(9S)` struct for block I/O, but richer
  - Created & initialized by filesystem, passed to the driver to perform an OSD operation, returned to filesystem with relevant status & data.
- **New interface functions:**
  - Create/delete an `osd_iotask(9S)`: `osd_iotask_alloc(9F), osd_iotask_free(9F)`
  - Initiate an OSD I/O operation: `osd_iotask_start(9F)`
  - Completion callback function: `osd_iotask.ot_iodone()`
    - Filesystem function, called by driver when the OSD I/O operation has completed
  - Open or close an OSD minor device: `osd_open_by_dev(9F), osd_close(9F)`
    - Provides access handle (`osd_handle_t`) to represent the OSD LUN
- **New command status and error reporting information**
  - Beyond EIO
OSD Filesystem-Driver Interface (continued)

• **osd_iotask(9S) struct:**
  - Key Members
    • **ot_partition_id**: 64-bit OSD Partition_ID
    • **ot_iiodone**: pointer to filesystem’s completion callback function
    • **ot_error, ot_errtyp**: error reporting codes and information.
    • Pointers to DATA IN and DATA OUT segments
    • **ot_client_private**: pointer to filesystem-private data area
  - Setup macros
    • One for each OSD Service Action
    • Set up the Service Action specific parameters & options
      - User_Object_ID, Starting Byte Address, etc.
      - Native endian – no byte swapping to be performed by filesystem
    • New macros to be added as new Service Actions are defined in the spec
  - Reusable: no need to free/realloc after a single I/O
• Caveat: New Solaris interface, not committed/supported
sosd Target Driver

• Peer to existing Solaris SCSA target drivers (sd, st, ses)
• Derived from Solaris sd target driver for disk devices
  – Use new OSD filesystem interface
  – Fully rework command and error recovery code paths
  – Add code to generate OSD CDBs and understand sense data descriptors
  – Includes uscsi(7) command path
    • Uses conventional open/close/ioctl interfaces
    • Accepts OSD + SPC commands
    • Testing
      • Management (FORMAT OSD, CREATE/REMOVE PARTITION)
• Binds to dtype 0x11 in INQUIRY data.
• Export entire OSD LUN as one minor device
  – Other schemes debated
Solaris I/O Path with OSD
Released as an OpenSolaris Project

- Email: osd-discuss@opensolaris.org
- Files
  - osd-bin-20061211.sparc.tar.bz2 - OSD project binary files
  - osd-src-20061211.sparc.tar.bz2 - OSD project source code files
- Limitations
  - Not all T10 OSD commands/features supported (see sosd.c)
    - No security, capabilities
  - SPARC only (x86 DMA issues)
  - No BIDI or multisegment data transfers
    - Solaris Express: kernel is a moving target
  - Object QFS changes not yet released for OpenSolaris
- Userland test program available (uscsxi-based)
- Now includes code for OSD target device implementation
  - iSCSI target mode
  - BIDI support in SCSA
  - Solaris Express build 55 required (download/build/install)
Issues and Concerns

• Presenting Object Device namespace to consumers
  – What access level(s) will clients want? (LUN? Partition?)
  – Solaris minor node scheme too limiting (18 bits max.)?
• BIDI: Not readily supported in Solaris today
  – Requires changes across kernel, drivers, and DDI interfaces
• Multisegment DATA IN and DATA OUT buffers
  – Understand multisegment semantics at most layers in the I/O path
  – DMA breakup (i.e., `PKT_DMA_PARTIAL`) becomes difficult
    • Works today for block devices on x86 systems (no IOMMU)
    • Drivers lack relevant context to break up OSD commands as needed to meet hardware transfer limits (size, alignment)
      – What to do when (not “if”!) an intermediate transfer or DMA mapping attempt fails?
    • Avoid burdening filesystems with managing HW limits and/or system resources
  – Doable, but involved
    • Need to handle multipathing? striping?
    • Probably should be designed along with BIDI support.
Issues and Concerns (continued)

• Error Recovery: it’s a correctness issue
  – IO & performance are (relatively) easy, robust & reliable recovery can be hard
  – Error reporting thru SCSI Status, Check Conditions and/or sense data:
    • Lots of places to check
    • Hard for filesystems to consume
    • Harder for drivers to map onto a straightforward but useful set of error codes
    • Transport level error cases also have to be handled.
  – When is it safe/appropriate to retry a failed command?
    • OSD Commands may be stateful (APPEND, attribute updates)
    • How much of the command actually completed at the LUN?
      – Section 4.7.2: Command function ordering
      – Section 4.13: Command Interactions: Caveat emptor!
      – Section 4.14.2.1: Command functions bits
      – Initiator cannot always tell reliably (timeouts, aborts, resets)
  - Recovery actions: what are you gonna do about it?
    - Which initiator(s) have ownership/responsibility?
    - Is it ever OK to reset an OSD LUN?
    - How thoroughly can recovery code be exercised/tested?
      - Concerns both initiators and targets

• Performance?
Potential Changes/Enhancements for OSD

- **Set Attribute from data in CDB**
  - For small attributes (8 bytes or less)
  - Saves mapping & transfer of additional DATA OUT buffer
- **64-bit CDB field alignments?**
  - Unaligned fields generate exceptions on SPARC
  - Can occur in other data structs (e.g. Attributes pages)
  - Debatable, but better if this issue can be prevented/avoided
- **Transactional Semantics?**
  - Ensure against data loss and/or inconsistency
  - Improve recovery from errors & crashes
Future Work

• Support x86 Solaris
• Resolve DMA/BIDI issues
  – Remove operational & performance constraints
• Support the full OSD command set
• Security & Capabilities
• Performance Testing
  – Initiators and Targets
• Define specific Attribute sets
  – QoS, ILM, Regulatory…. 
  – XAM API Compliance support
• Need more target devices and transports
• Incorporate upcoming spec changes
Questions?
Thanks!