

Integrating Parallel File Systems with Object-based Storage Devices

*Ananth Devulapalli*¹ Dennis Dalessandro¹ Pete Wyckoff¹
Nawab Ali² P. Sadayappan²

¹Ohio Supercomputer Center

²Department of Computer Science and Engineering
The Ohio State University

Supercomputing 2007

Overview

- ▶ Motivation
- ▶ Technological background
- ▶ OSD Infrastructure
- ▶ Integration of OSDs with parallel filesystem
- ▶ Experimental evaluation

Vision

- ▶ Storage density is up but I/O bandwidth and latency not improving
- ▶ Large data
 - CERN LHC, Digital LoC
- ▶ Increasing cost of storage management
 - Storage: 50% cost of data centers

- ▶ Storage density is up but I/O bandwidth and latency not improving
- ▶ Large data
 - CERN LHC, Digital LoC
- ▶ Increasing cost of storage management
 - Storage: 50% cost of data centers
- ▶ Simplify storage infrastructure
- ▶ Use intelligent peripherals to improve scalability, manageability and performance of storage systems

- ▶ Storage density is up but I/O bandwidth and latency not improving
- ▶ Large data
 - CERN LHC, Digital LoC
- ▶ Increasing cost of storage management
 - Storage: 50% cost of data centers
- ▶ Simplify storage infrastructure
- ▶ Use intelligent peripherals to improve scalability, manageability and performance of storage systems

Serverless, direct-access, storage model

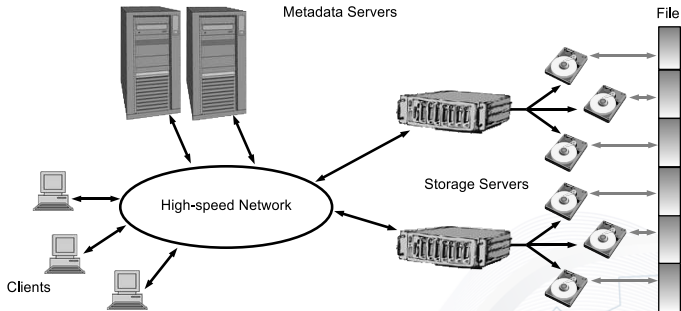
Motivation

- ▶ New storage technology: Object-based Storage Devices (*OSD*)
- ▶ Intelligent, higher-level object interface
- ▶ Secure building block for direct-access filesystems

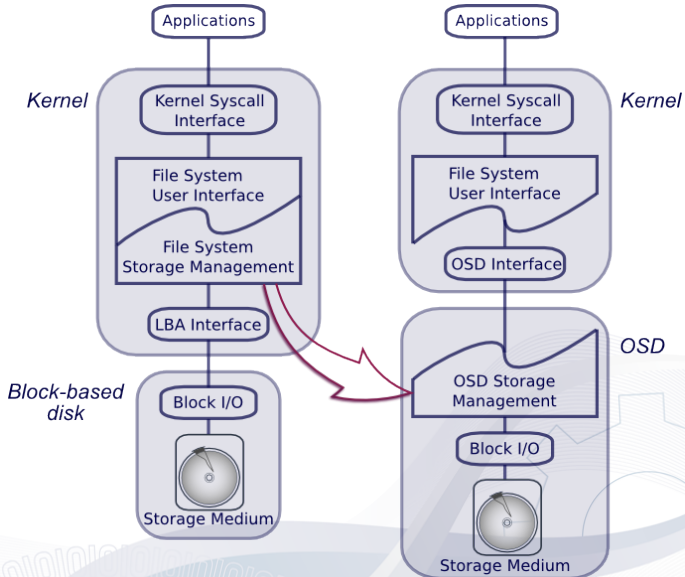
- ▶ New storage technology: Object-based Storage Devices (*OSD*)
- ▶ Intelligent, higher-level object interface
- ▶ Secure building block for direct-access filesystems

- ▶ Analyze trade-offs in using OSDs for various aspects of filesystems
- ▶ Research platform for realizing our vision

Parallel File System Architecture



Object-based Storage Device (OSD) Architecture



Object-based Storage Device (OSD) Architecture (contd)

- ▶ OSD Technology
 - SCSI Extension
- ▶ Object
 - Root, Partition, Collection, User Object
- ▶ Object encapsulation
- ▶ Attributes
 - User assigned, but device managed
 - Large space for rich metadata
 - Shared attributes
- ▶ Transport
 - iSCSI, iSCSI Extensions for RDMA (iSER), Fiber Channel

Object

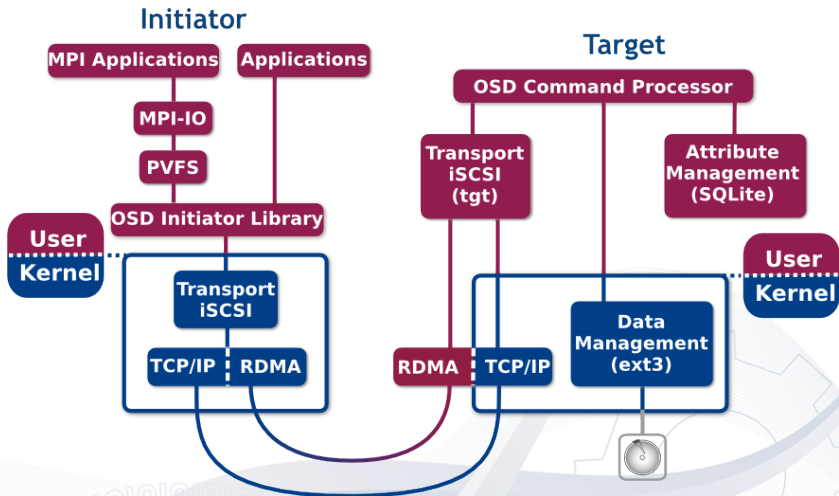
Attributes

Attribute Page	Attribute Number	Value
1	9	"foo"
2	1	1MB
3	1	01:00:00
3	2	02:00:00

Data

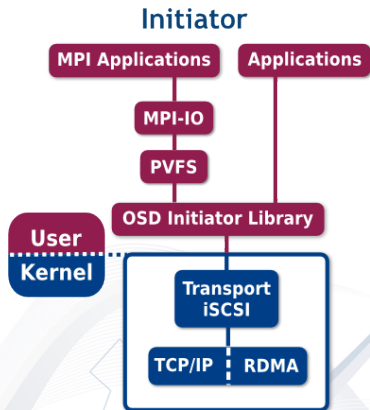
```
1010101111100010101
101010101010101010
0000000111111100101
.....
```

OSD System Design



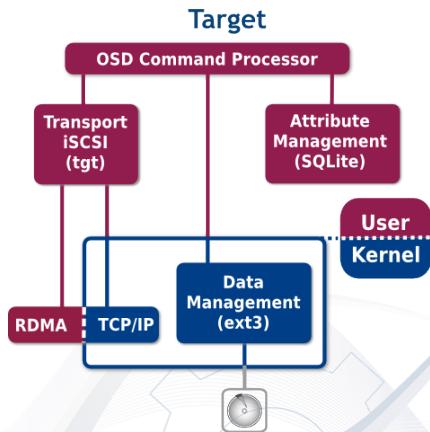
OSD Initiator

- ▶ Exports OSD interface to client applications
- ▶ Transport: iSCSI, iSER
- ▶ Generates SCSI commands, submits to “SCSI mid-layer”
- ▶ Kernel modifications
 - 200B CDB (> 16B)
 - Bidirectional commands in SCSI stack
 - Support for I/O vectors to minimize copying
- ▶ Clean interface for handling OSD buffer model
 - Multiple entities within single buffer
 - Many combination of entities
 - Offset calculation and alignment constraints



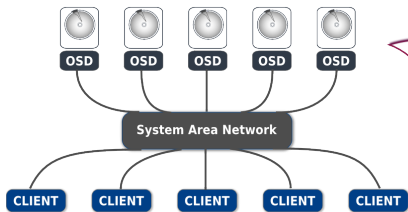
Software implementation of OSD Target

- ▶ SCSI/iSCSI layer
 - iSCSI session management, SCSI header parsing
 - Userspace *tgt*
 - iSCSI and iSER [SNAPI '07]
- ▶ OSD Command processor
 - Object, Attribute, Input/Output, Device management, Multi-object
- ▶ Data Management
 - Relies on underlying filesystem
- ▶ Attribute Management
 - SQLite-based
 - More details [MSST '07]

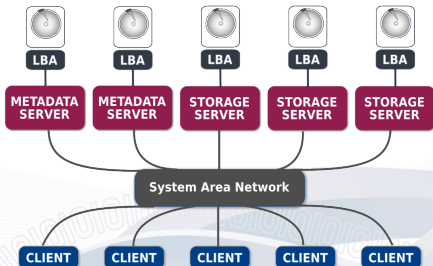


Integration of OSDs with Parallel Filesystem

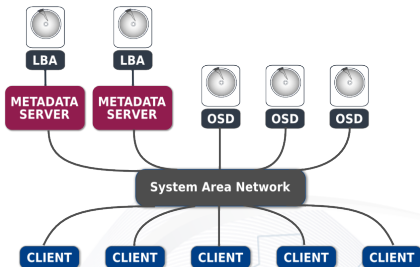
DIRECT-ACCESS MODEL



SERVER FRONTED MODEL



OSD AS STORAGE SERVER



Integration steps

- ▶ PVFS design helps in integration
 - Stateless
 - Servers don't communicate with each other
 - Servers don't initiate communication with clients
- ▶ PVFS client library modifications
 - BMI interface
- ▶ Configuration management
- ▶ Commands translations:

PVFS Command	OSD Command
PVFS PING	SCSI TEST UNIT READY
PVFS STAT	OSD GET ATTRIBUTE
CREATE file handle	OSD CREATE
REMOVE file handle	OSD REMOVE
...	...

Integration steps (contd)

- ▶ Flow control
 - Driven by the client
 - Target controls the flow
- ▶ Handle space
 - Trivial mapping from PVFS handles to OSD Object Identifiers
 - Flexible mapping, extent list for handle assignment

- ▶ PVFS design
- ▶ OSD as storage server straight forward
- ▶ Powerful OSD primitives
- ▶ Single buffer model
 - Read + Get attributes of 1MB for an object of 1B
 - Allow OSD to specify retrieved offset
- ▶ Multiple transfers to same object for non-contiguous I/O
 - MPI/IO type strided access
 - Allow gather and scatter capability for a single object
- ▶ Lack atomic operations
 - Difficult to offload metadata operations

Experiments

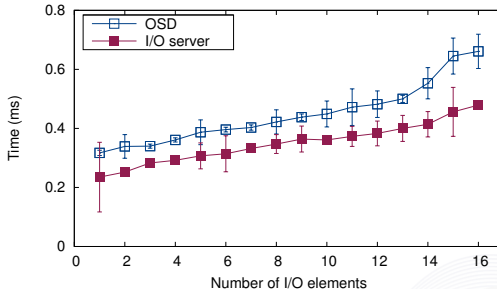
- ▶ Comparison of software OSD with a server.
- ▶ Goal of the experiments:
 - Highlight protocol costs
 - Server performance is used to gauge relative cost
 - Performance gains will mostly be due to implementation artifacts
- ▶ Experimental setup
 - 30 nodes
 - Dual AMD Opteron 250,
 - 2GB RAM, 80 GB SATA disk
 - Linux 2.6.20
 - Tigeon 3 Gigabit Ethernet NIC
 - Single SMC 8648T 48-port switch

Processing phase	Overhead
SQLite	$81.3 \pm 0.3 \mu\text{s}$
CDB	$2.2 \pm 0.5 \mu\text{s}$
iSCSI	$29.9 \pm 1.7 \mu\text{s}$
Initiator	$125.6 \pm 2.9 \mu\text{s}$
Total	$239.0 \pm 1.1 \mu\text{s}$

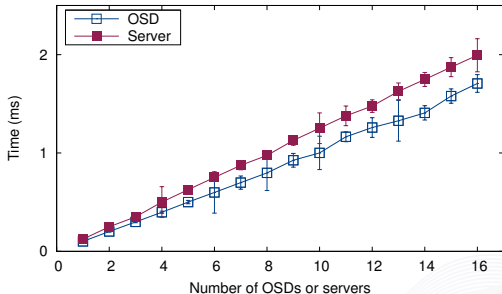
Table: Overheads in `getattr`

- ▶ PVFS not involved
- ▶ ASIC for iSCSI layer
- ▶ Content-addressable RAM for attribute operations

Stat Microbenchmark

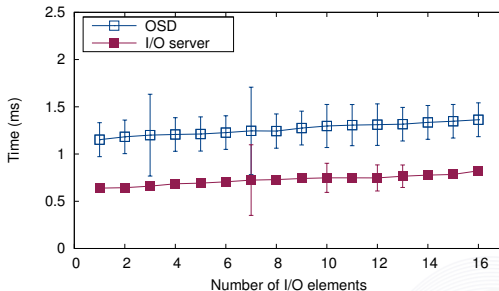


- ▶ 1 metadata server, 1 client, varying I/O elements
- ▶ OSD expensive: SQLite and protocol overhead
- ▶ Same rate of scalability



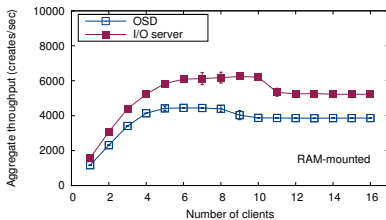
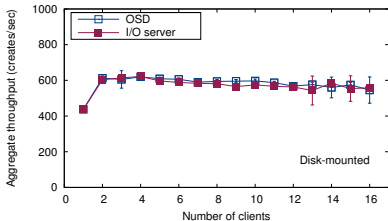
- ▶ 1 metadata server, 1 client, varying I/O elements
- ▶ Ping is handled in *tgt*
- ▶ OSD scales better

Create Latency



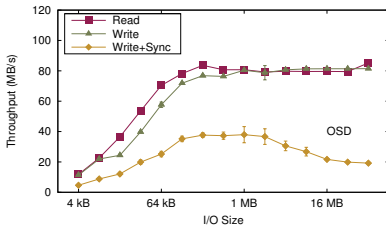
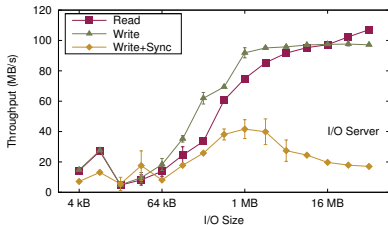
- ▶ 1 metadata server, 1 client, varying I/O elements
- ▶ RAM-mounted to eliminate disk effects
- ▶ OSD expensive by a constant amount; SQLite/protocol overhead

Create Throughput

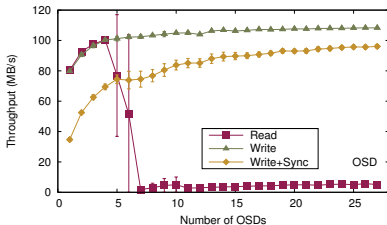
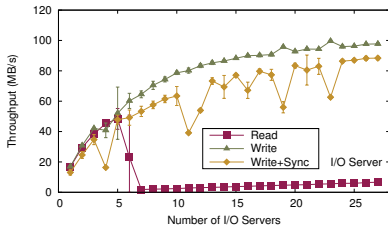


- ▶ 1 metadata server, 1 I/O element, varying number of clients
- ▶ Stress test
- ▶ Fine-grain multi-threading to improve OSD performance

I/O Throughput

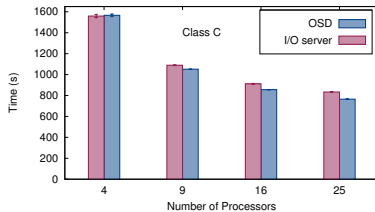
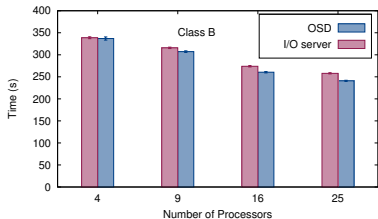


- ▶ perf benchmark
- ▶ 1 metadata server, 1 I/O element, 1 client
- ▶ Write+Sync: data flushed to disk
- ▶ OSD performs better than PVFS for small messages
- ▶ OSD writes slower than reads: extra `READY TO SEND`
- ▶ Peak throughput of OSD < I/O server: transfer limit of 256 kB, lack of pipelining in OSD



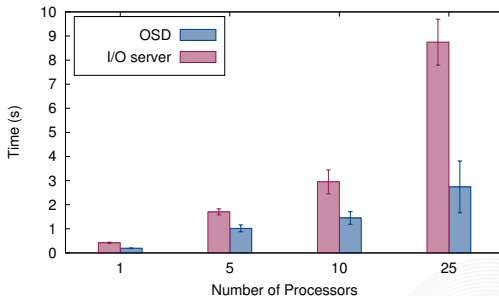
- ▶ 1 metadata server, 1 client, varying I/O elements
- ▶ Stress test, constant work per server
- ▶ OSD saturates quickly, gets better scaling than PVFS
- ▶ Congestion causes degradation in reads

NAS Block-Tridiagonal IO (BTIO)



- ▶ *full* version of NAS BTIO
- ▶ 1 metadata server, 4 I/O elements, varying (square) number of clients
- ▶ Small message sizes help OSD

Flash (Checkpoint)



- ▶ 1 metadata server, 4 I/O elements, varying number of clients
- ▶ $64 \text{ kB} \leq \text{write per server} \leq 100 \text{ kB}$
- ▶ PVFS must improve performance for small message sizes

Conclusions

- ▶ Demonstrated integration of OSDs with parallel filesystem
- ▶ Feasible to replace servers with disks
- ▶ Highlighted mismatches between parallel filesystem requirements and OSD capabilities

Future Work

- ▶ Multi-object commands ✓
- ▶ iSER ([SNAPI '07]) ✓
- ▶ Metadata offload ✓^{1/2}

Administrivia

- ▶ Acknowledgment: NSF
- ▶ iSER patch: `git://git.osc.edu/tgt`
- ▶ Browse iSER patch: `http://git.osc.edu/?p=tgt.git`
- ▶ `http://www.osc.edu/research/network_file/projects/object/index.shtml`
- ▶ Contact: `osd-group@osc.edu`