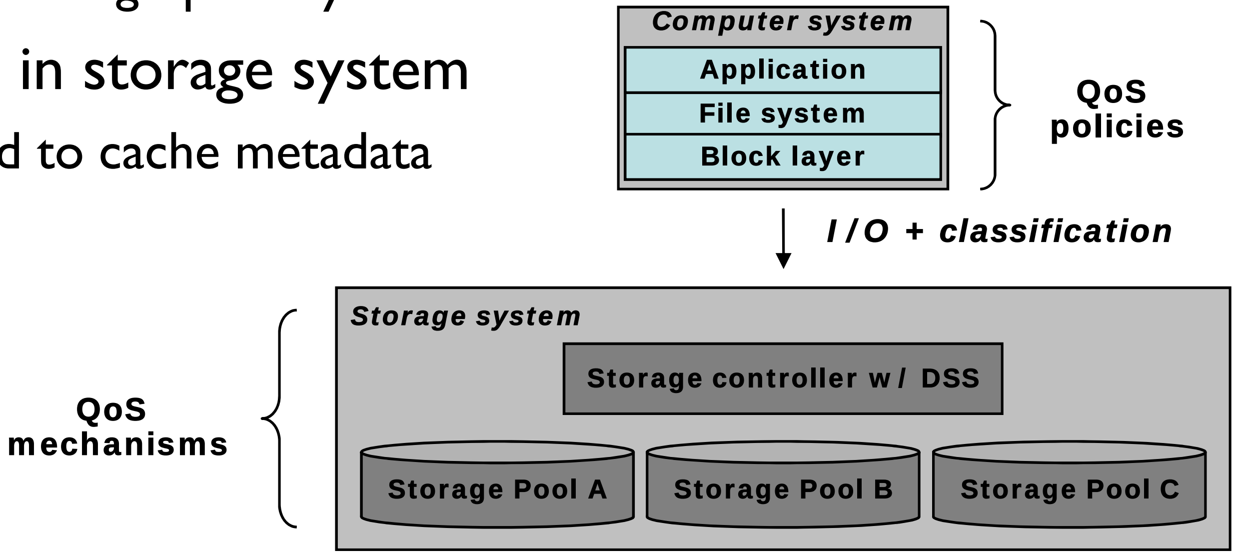
Differentiated Storage Services

Brian McKean, LSI

Michael Mesnier, Intel

Differentiated Storage Services

- A QoS framework for file and storage systems
  - I/O differentiated (classified) by file system
    - E.g., metadata versus data
  - QoS policies set by admin/software for each class
    - E.g., metadata is high-priority
  - QoS enforced in storage system
    - E.g., SSD used to cache metadata



Motivation – why now?

- Disruptive effects of flash is a key driver
- Flash-based (NAND) devices are:
  - Very high performance
  - Much higher cost per GB than disk
- How to most ***efficiently*** use flash storage?
  - Use as fast volume (let FS allocate it)
  - Use as fast cache/tier (let storage allocate it)

Our approach

- Use SSD as a fast cache/tier
  - Knowing what to cache is the challenge!
- Want storage to allocate the SSD efficiently
  - Based on the I/O patterns of each class
  - Based on the QoS policies of each class

DSS provides the class information

(Hint: we exploit the SCSI Group Number)

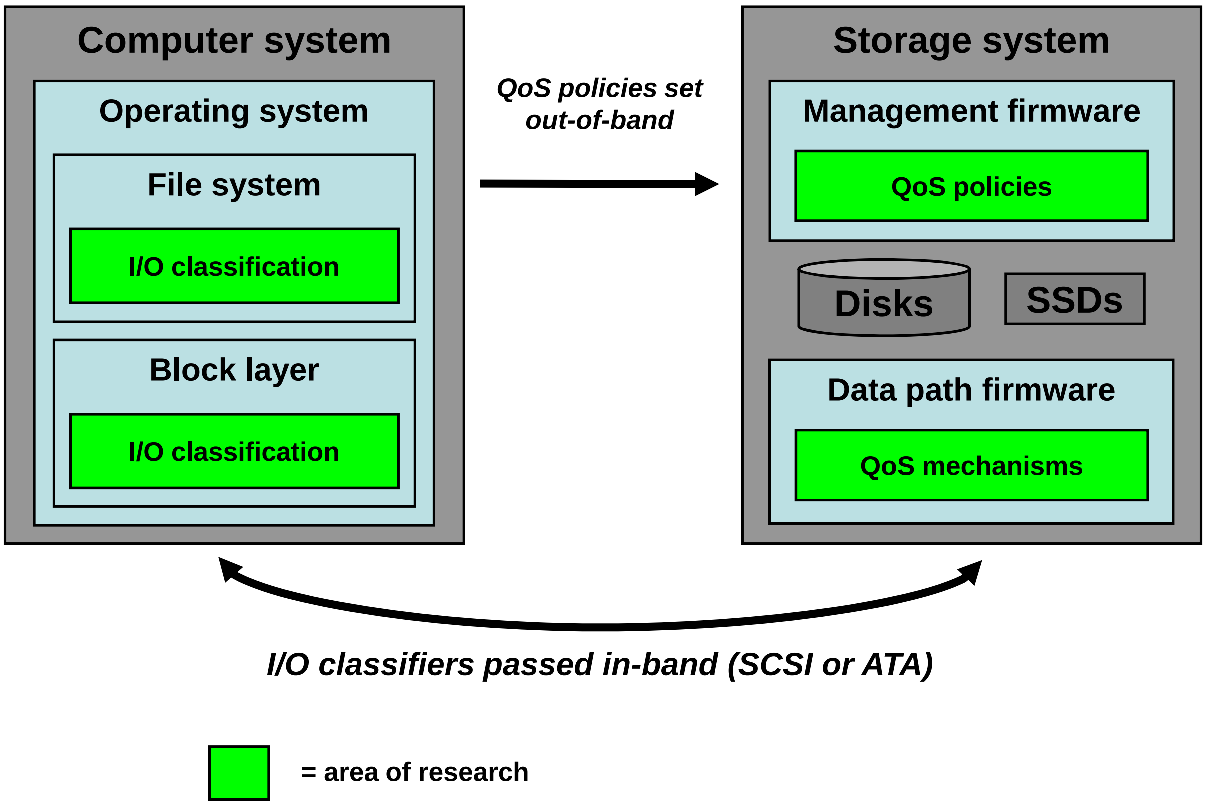
Standard research disclaimers

- DSS is not a product – yet
- This is joint research

Topics

- Overview
- **Project Details**
- Benchmark results
- Summary

High-level architecture



A classification scheme for Ext3

- Initial classification scheme for Ext3:
  - Metadata blocks (inodes, bitmaps, etc.)
  - Directory blocks
  - Regular file data
    - Immediate blocks (first 4KB of each file)
    - Small offset (4 KB to 16KB)
    - Medium offset (16 KB to 64 KB)
    - Large offset (64 KB to 256 KB)
    - Bulk offset (beyond 256 KB)
  - Journal

In-band I/O classification (SCSI)

- Block-level interface for I/O classification
  - Leverage SCSI Group Number field in CDB
  - Used for transmission of I/O class information

Table 38 — READ (10) command

Byte \ Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (28h)							
1	RDPROTECT			DPO	FUA	Reserved	FUA_NV	Obsolete
2	(MSB)	LOGICAL BLOCK ADDRESS						
5								(LSB)
6	Reserved			GROUP NUMBER				
7	(MSB)	TRANSFER LENGTH						
8								(LSB)
9	CONTROL							

SCSI Group Number

- FS maps classes to SCSI Group Numbers. E.g.,

Ext3 class	SCSI Group Number
Metadata	0
Journal	1
Directory	2
Immediate data	3
Small offset	4
Medium offset	5
Large offset	6
Bulk offset	7

Out-of-band policy setting

- QoS policies set out-of-band
  - Software PoC: module parameter
  - Hardware PoC: RAID controller console
- Why out-of-band?
  - Separates classification from policy
    - Classification is a one-time change to FS
    - QoS policies may vary by storage system
  - Easier to configure/tune
    - Can change QoS policies without modifying FS

Setting QoS polices

- Administrator sets QoS on each class. E.g.,

Ext3 SCSI Group Number	QoS Policy
0 (Metadata)	High priority
1 (Journal)	High priority
2 (Directory)	High priority
3 (Immediate data)	High priority
4 (Small offset)	Low priority
5 (Medium offset)	Low priority
6 (Large offset)	Low priority
7 (Bulk offset)	Low priority

Settings will vary by FS and storage system - finding optimal settings is part of our research

Enforcing QoS in storage

- E.g., SSD used as cache for high-priority I/O:

QoS Policy	QoS Mechanism
High Priority	Cache in SSD
Low priority	Don't cache

FS and administrators are abstracted from the QoS mechanisms (and there may be many)

Summary of DSS architecture

- Step 1: I/O classification occurs in the FS
  - E.g., Metadata, dirs, journal, file extents
- Step 2: Admin. associates QoS with each class
  - E.g., Metadata class is high priority
- Step 3: FS is mounted
  - OS includes class information with each I/O
  - Storage system differentiates by class
- We're developing two proofs-of-concept
  - Both targeting performance differentiation
  - SSD reserved for highest-priority classes

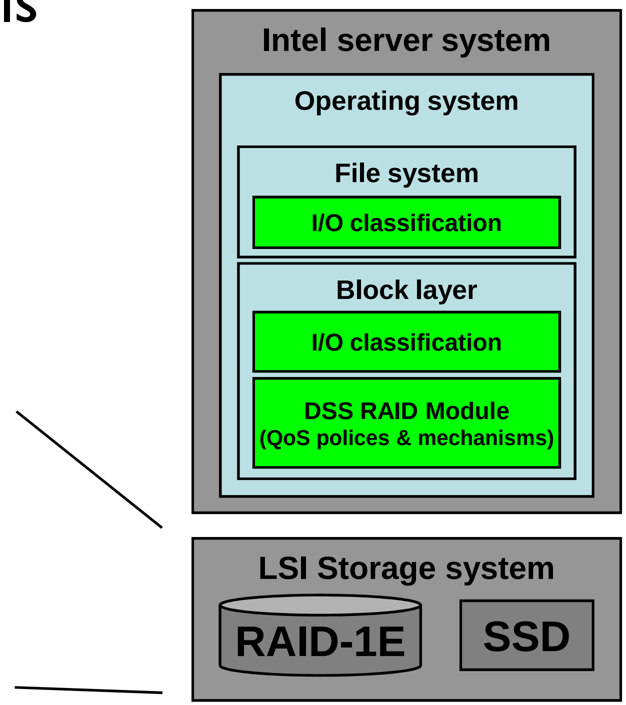
Required changes to the OS

- Changes to linux kernel (very minor)
  - Modified I/O routines to include classification
  - Added classification field to bio structure
    - A 5-bit field use to classify each I/O request
  - Modified I/O merging functions
    - Only merge bios with same classification
  - Modified SCSI disk driver (sd.c)
    - Copies 5-bit class from bio into CDB
- Journal I/O classified in journaling block device
  - A separate module from Ext3

Software PoC (Intel)

- Host-based Linux RAID module (RAID-9)
  - Base tier: LSI enhanced mirror (RAID-1E)
  - Fast tier (cache): Intel Enterprise SSD
    - DSS daemon cleans cache as it fills
    - LRU eviction
- Configurable caching policies
  - Explored in evaluation

Research @ Intel Demo



HW PoC (LSI)

- HW PoC based on shipping RAID array
  - Cache algorithms and I/O paths are optimized
  - SSD cache metadata will be persistent
  - Code to handle failure cases
    - E.g., power loss, SSD failure
  - Support for large, multi-SSD caches

External RAID w/ DSS

Topics

- Overview
- Project Details
- **Benchmark results**
- Summary

Benchmark setup

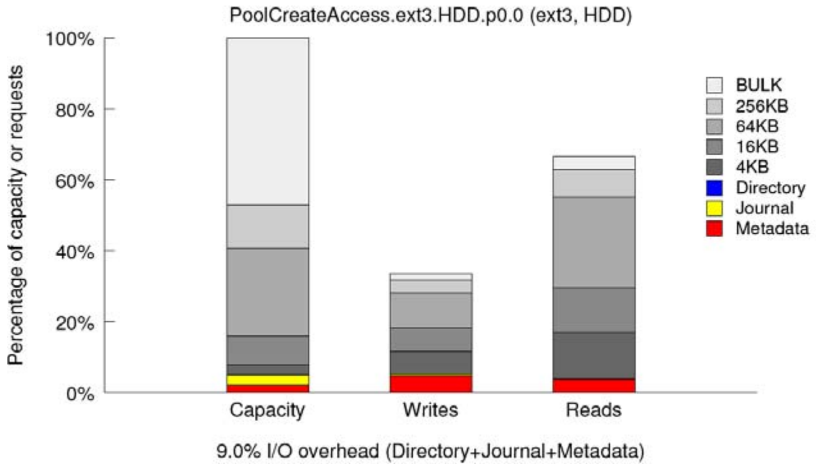
- Xeon-based server system
  - Quad-core DP, 1 GB RAM, Linux 2.6.24
  - DSS SW RAID module (Intel)
    - Base tier: LSI array (5-disk SATA RAID-1E)
    - Fast tier: Intel Enterprise SSD
- Various file system benchmarks
  - File pool manipulation, mail, tape archive
- Each benchmark uses ~5 GB of disk space

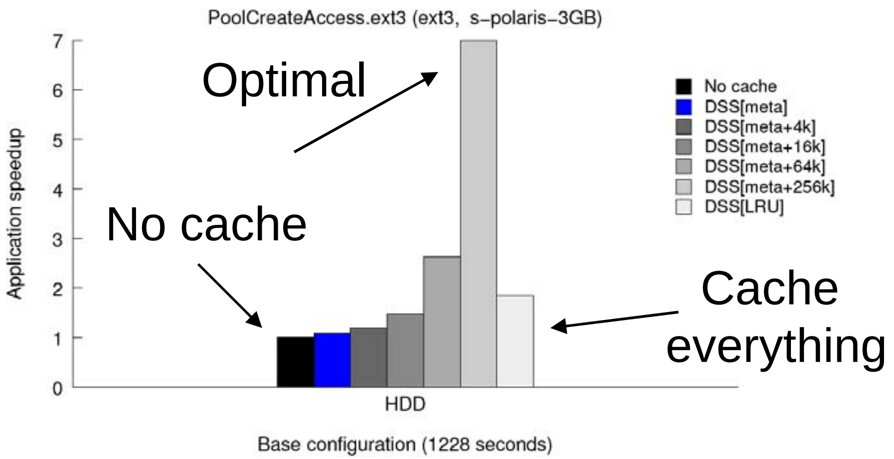
DSS setup

- SSD as write-allocate disk cache (3 GB)
  - DSS syncer daemon cleans as cache fills
    - Syncer copies blocks from SSD to HDD
    - Starts cleaning at 75% dirty, stops at 25%
- Testing methodology
  - Base cases: cache nothing, cache everything
  - DSS cases: ***selective*** caching
    - Test 1: cache metadata, journal, and dirs.
    - Test 2: cache first 4 KB of each file
    - Tests 3-5: cache 16, 64, and 256 KB of each file

File pool manipulation

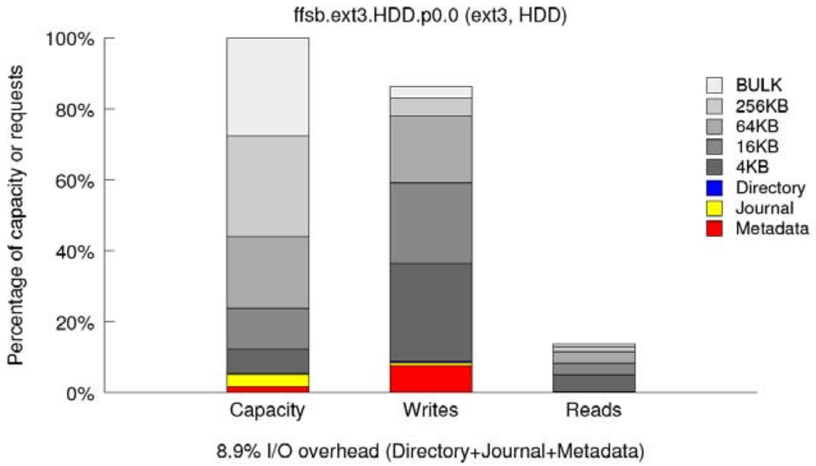
- Workload characteristics
  - Phase 1: Postmark
    - 32K files, 181 dirs.
    - Files 1KB to 128 KB
  - Phase 2: Pollute cache
    - Create archive of pool
  - Phase 3: Access pool
    - Randomly read all files
  - Working set: ~5 GB
  - 33% writes
- **DSS speedup: 3.8x**

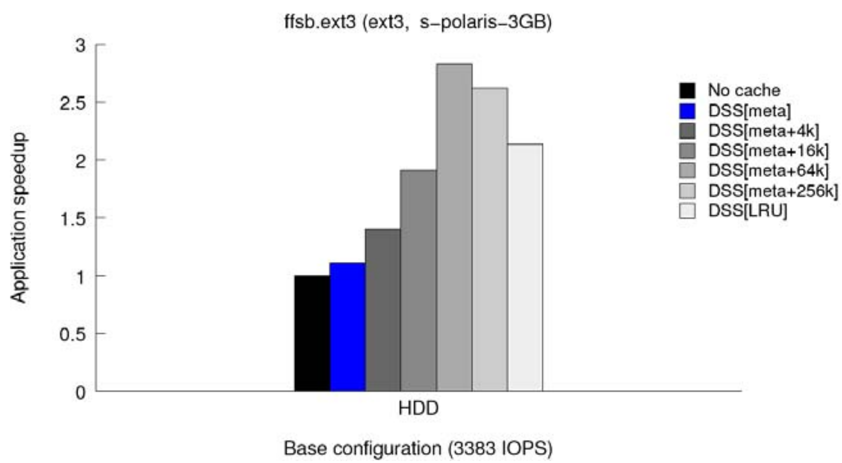




Mail server workload (FFSB)

- Workload characteristics
  - Phase 1: pool creation
    - 50K files, 100 dirs.
    - 1 KB to 1 MB file size
    - 4 KB request size
    - ~5 GB storage capacity
  - Phase 2: transactions
    - Read/write
    - Create/delete
  - 86% writes*
- **DSS speedup: 33%**

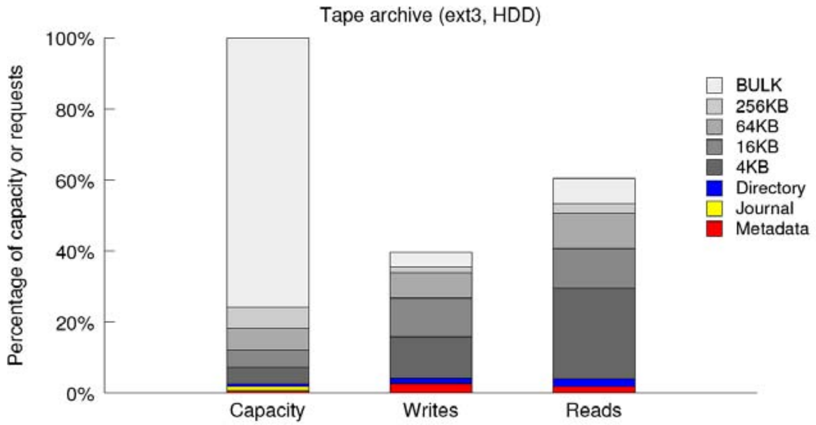




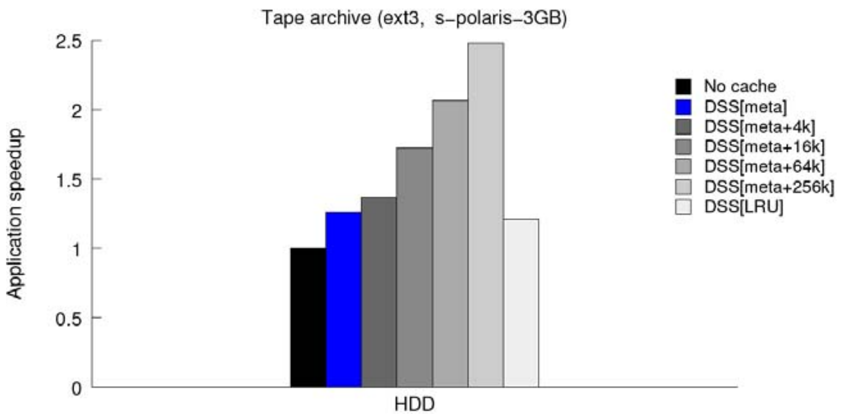
* Includes creation of initial file pool

Untar+tar of Fedora 8 /usr

- Workload characteristics
  - Phase 1: untar archive
    - 90K files, 7K dirs.
    - Files 0K B to 77 MB
  - Phase 2: create archive
  - ~5 GB storage capacity
  - 40% writes
- **DSS speedup: 2x**



8.1% I/O overhead (Directory+Journal+Metadata)



Base configuration (412 seconds)

Remaining research

- Self-tuning aspects of selective caching. E.g.,
  - Should we always cache all metadata?
  - How much of each file should be cached?
- Multi-server and cluster environments
  - Implications/opportunities for resource sharing
    - Benchmarks shown were run in isolation
    - Can DSS improve how storage is shared?
- Managing numerous tiers (not just fast and slow)
  - DRAM, NAND, various RAID levels, disks

Other areas to explore

- Other file systems (NTFS?) and databases
- Standardization
  - QoS classification interfaces (e.g., T10)
  - QoS policy interfaces
- Implications for NAS (NFS / CIFS)
- Effects on server memory size
  - Can DSS help reduce DRAM requirements?

Summary of DSS

- A QoS framework for file and storage systems
  - File systems classify I/O requests
  - Storage systems differentiate by class
- DSS separates policy from mechanism
  - File systems establish QoS policies
  - Storage systems enforce QoS
- DSS enables new storage system optimizations
  - E.g., *selective* caching of data in SSDs

Benchmark summary

- Speedup over LRU (with cache pressure):
  - File Pool – 3.8x improvement
  - Mail Server – 33% improvement
  - Untar & Tar – 2x Improvement

Questions?