Discriminating Hierarchical Storage (DHIS)

Chaitanya Yalamanchili,
Kiron Vijayasankar,
Erez Zadok
Stony Brook University

Gopalan Sivathanu
Google Inc.

http://www.fsl.cs.sunysb.edu/

Outline

- **Introduction**
- Background
- DPROTO Framework
- DHIS Design
- Evaluation
- Related Work
- Conclusions

Large-scale Storage Systems

EMC® Symmetrix 8830
80 333Mhz processors, 64GB RAM

NetApp FAS6080
1170 Disks, 1170TB Storage

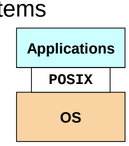
- Storage interfaces unchanged
- Processing power and memory available

Motivation

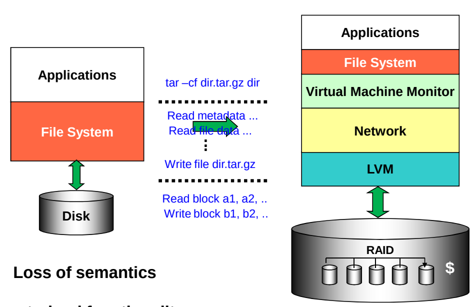
- Storage System Tradeoffs (RAID)
  - Availability
    - Replication, Multipathing, Failover
  - Cost
    - Power, Backup, Capacity
  - Performance
    - Striping, Load balancing
- Storage Management difficult
- Make better use of processing power at the storage system level to balance tradeoffs

Information Gap

- Layered System Design
  - Layers: fundamental to modern systems
    - Modularity
    - Independent innovation
  - But layers hide information
    - Information gap
    - Age-old problem in computer systems
    - Constrained functionality within layers



Information Gap (cont.)



Loss of semantics

Constrained functionality

DHIS Overview

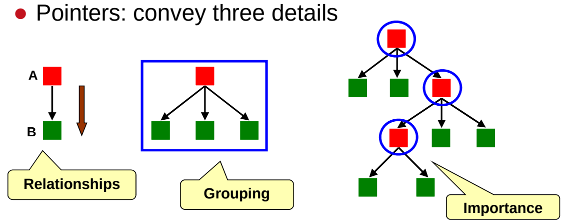
- Bridges the information gap with an extended storage system interface
- Uses hints from higher-level software (applications, file systems)
- Storage management done inside the storage system firmware
- RAID levels and NVRAM comprise the hierarchy
- Ordering in the hierarchy depends on the hints (configurable)

Outline

- Introduction
- **Background**
- DPROTO Framework
- DHIS Design
- Evaluation
- Related Work
- Conclusions

Block-Based Storage

- Two basic entities: data and pointers
- Pointers: convey three details

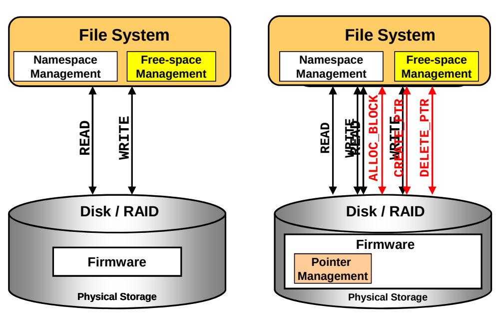


- Today's disks are unaware of pointers

Type-Aware Storage

- Bridge information gap through pointers
  - Disks aware of pointers
- Higher level software communicate pointers
  - File systems or user applications
  - Explicit disk interface extension
- Type-Safe Disks (TSDs)
  - Track pointers and enforce constraints
  - Manage free-space

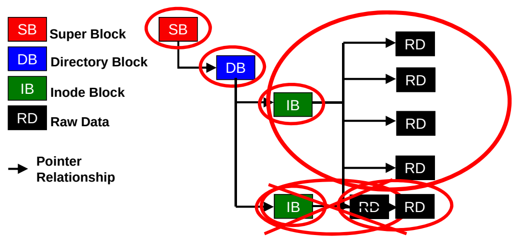
TSD Infrastructure



TSD Interface

- `READ(Block)`
- `WRITE(Block)`
- `ALLOC(Ref, Count, Hint)`
- `CREATE PTR(Src, Dest)`
- `DELETE_PTR(Src, Dest)`

TSDs and File Systems



Operation-level hints

TSD Project

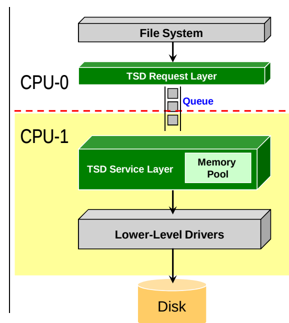
- Type-Safe Disks [OSDI 2006]
- Exploiting Type-Awareness in a Self-Recovering Disk [ACM StorageSS 2007]
- Selective Versioning in a Secure Disk System [Usenix Security 2008]

Outline

- Introduction
- Background
- **DPROTO Framework**
- DHIS Design
- Evaluation
- Related Work
- Conclusions

DPROTO Framework

- **CPU**
  - Dual CPU machine
  - Isolated CPU for disk
  - Linux "`cpusets`" interface
- **Memory**
  - Isolated memory pool
  - Pre-allocated
  - Leveraged "`mempool`"
- **Broader uses**
  - Prototype other disk-level features



Outline

- Introduction
- Background
- DPROTO Framework
- **DHIS Design**
- Evaluation
- Related Work
- Conclusions

DHIS

A **D**iscriminating **HI**erarchical **S**torage system

- Communicate data properties to disk
  - Granularity: groups of data conveyed through pointers
  - Well-defined ***attributes***
- Placement decisions based on attributes
  - Reliability
  - Performance
  - Cost
- Series of configurable hierarchies to place data in

Attributes

- Importance: High / Low
- Access Pattern: Random / Sequential
- Read-most / Write-most
- Hot / Cold
- Life-time: Temporary / Long-lived

Can add new attributes

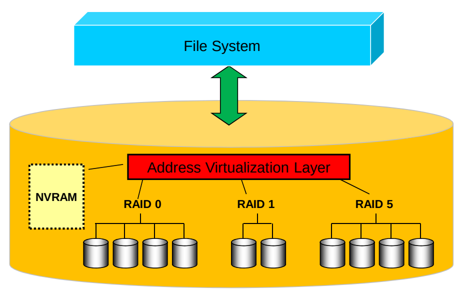
DHIS Interface

- Adds **SETATTR(Block, attr)**
  - Used by higher-level software to pass hints
- Supports the TSD interface
  - Relates blocks using logical pointers
  - Attributes inherited from parents

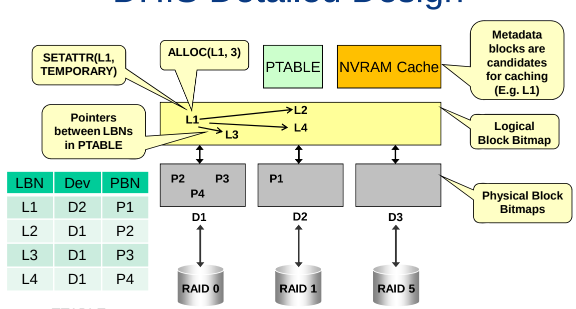
Ext2DHIS File System

- Unlike Ext2, Ext2DHIS does not perform free-space management
- Block allocation via the **ALLOC** primitive
- Issues **CREATE_PTR** or **DELETE_PTR** whenever a new pointer is added or removed for a meta-data block
- Uses **SETATTR** to set attributes

DHIS Architecture



DHIS Detailed Design



LBN	Dev	PBN
L1	D2	P1
L2	D1	P2
L3	D1	P3
L4	D1	P4

TTABLE

RAID Placement Policy

IMPORTANT	ACCESS PATTERN	READ/WRITE MOST	HOT/COLD	Preferred RAID Levels
Low	Any	Any	Any	0, 5, 1
High	Any	Any	Cold	5, 1, 0
High	Not-Set	Not-Set	Not-Set / Hot	5, 1, 0
High	Random	Not-Set / Write	Not-Set / Hot	1, 5, 0
High	Random	Read	Not-Set / Hot	5, 1, 0
High	Sequential	Any	Not-Set / Hot	5, 1, 0

Configurable, can add new RAID levels

Policies

- Temporary files
  - RAID-0
  - Placed in an isolated portion of disk
    - Reduce disk fragmentation
- Meta-data blocks
  - Identified as those having outgoing pointers
  - Placed in the RAID level of highest reliability and best random access performance (RAID 1 in our setup)

Policies (cont.)

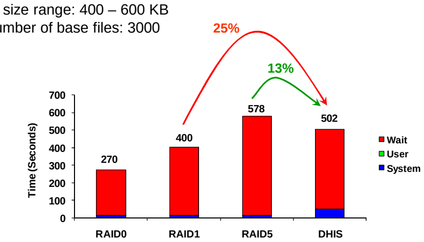
- NVRAM caching
  - Caching candidates
    - All meta-data
    - "Hot" and "write-most" data
  - Absorbs write latency
  - Sequential workloads are not chosen
  - Use block liveness info to remove freed blocks

Outline

- Introduction
- Background
- DPROTO Framework
- DHIS Design
- **Evaluation**
- Related Work
- Conclusions

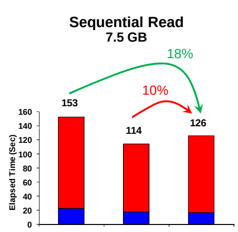
Postmark

File size range: 400 – 600 KB
Number of base files: 3000

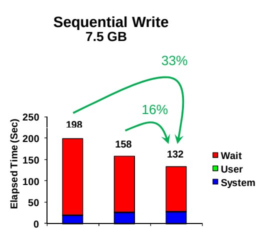


Attributes: IMPORTANT | RANDOM ➔ RAID1 policy

Micro-Benchmarks

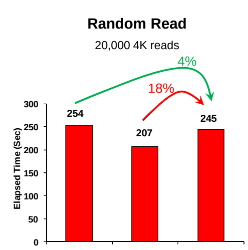


Attributes: IMPORTANT | SEQUENTIAL | READ-MOST

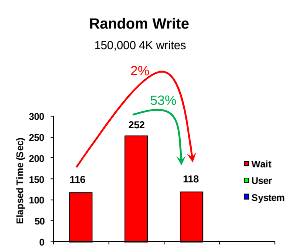


Attributes: IMPORTANT | SEQUENTIAL | WRITE-MOST

Micro-Benchmarks (cont.)

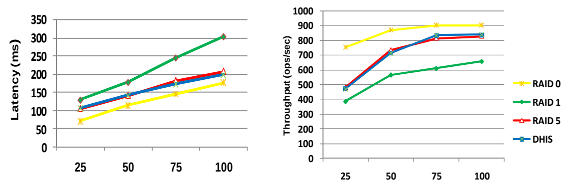


Attributes: IMPORTANT | RANDOM | READ-MOST



Attributes: IMPORTANT | RANDOM | WRITE-MOST

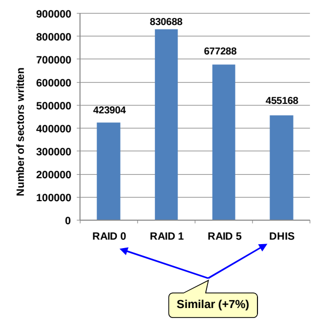
OLTP Workload



Latency and Throughput vs. no. of reader threads (FileBench OLTP)

Attributes: IMPORTANT | RANDOM | READ-MOST ➔ RAID5 policy

Kernel Compile Workload



Compiled the linux kernel 2.6.28 sources

Attribute: TEMPORARY ➔ RAID0 policy

Outline

- Introduction
- Background
- DPROTO Framework
- DHIS Design
- Evaluation
- **Related Work**
- **Conclusions**

Related Work

- **Object-Based Storage Devices**
  - Objects versus Blocks
  - Objects support attributes
  - **Problem:** Require fundamental changes to higher-level software
- **Self-* Storage**
  - Automated administration
  - Notion of supervisors, workers, and routers
  - Works better when workers are intelligent (like DHIS)
- **HP AutoRAID**
  - Newly written data placed in RAID 1
  - Data migrated to RAID 5 as it gets cold
  - **Problem:** Limited to cold/hot attribute; data migration can be costly

Related Work (cont.)

- **ExRAID**
  - Expose fault boundaries and redundancy information to the file system
  - **Problem:** Managing redundancy within the file system can be difficult, requiring the careful placement of inodes and data blocks to ensure efficient operation under failure.
- **RAIF**
  - Stackable fan-out file system
  - **Problem:** Rule management done at the file system level; extra layer adds overhead.
- **Semantically smart disks**
  - Automatically infer higher-level operations and data structures
  - **Problem:** Inference is not always accurate

Conclusions

- Enables easy storage management
  - Fine-grained policies
- Attribute association can be automated
  - Ext2DHIS file system
  - Attributes based on file extension
- Online attributes
  - Obviates need for data migration
- Future: *someone* should offer a more intelligent storage system...

Discriminating Hierarchical Storage (DHIS)

Chaitanya Yalamanchili,
Kiron Vijayasankar,
Erez Zadok
Stony Brook University

Gopalan Sivathanu
Google Inc.

Q&A

http://www.fsl.cs.sunysb.edu/