Integrating Flash-based SSDs into the Storage Stack

Raja Appuswamy, David C. van Moolenbroek,
Andrew S. Tanenbaum

Vrije Universiteit, Amsterdam

April 19, 2012

Introduction: Hardware Landscape

- $/GB of flash SSDs is still much higher than HDDs
  - Flash-only installations are prohibitively expensive
- Hybrid Storage Architectures (HSA) - a viable alternative
  - Use high-performance SSDs in concert with high-density HDDs
- Caching HSAs
  - Extends the two-level RAM/HDD memory hierarchy
  - SSDs used as intermediate caches
- Dynamic Storage Tiering HSAs
  - Establishes tiers of devices based on performance
  - SSDs used for primary data storage

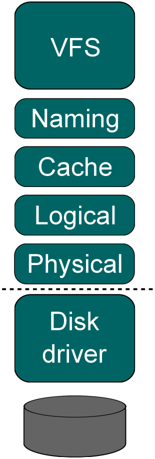
Introduction: Complex Workloads

- The duality of storage workloads
  - Isolated: well-defined app-specific access patterns
  - Virtualized: disjoint I/O requests blended into a single stream
- Two fundamental questions need to be answered
  - How do DST/Caching systems fare under such workloads?
  - Is the “one-architecture-per-installation” approach correct?
- We need a modular, flexible hybrid storage framework
  - Perform side-by-side comparison of various architectures
  - Understand the impact of design alternatives

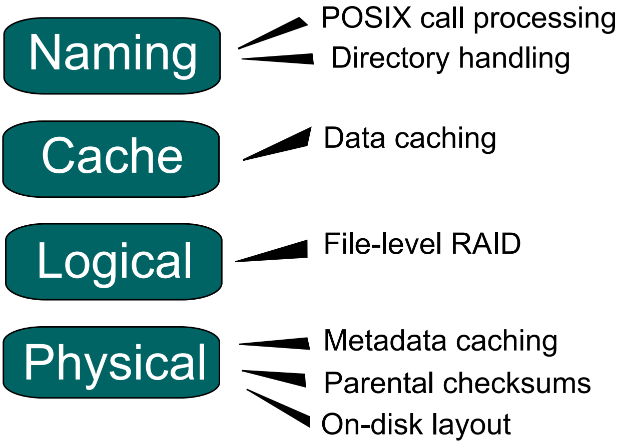
The Loris Storage Stack - Layers and Interfaces

- File-based interface between layers
  - Each file has a unique file identifier
  - Each file has a set of attributes
- File-oriented requests:

create	**truncate**
delete	**getattr**
read	**setattr**
write	**sync**

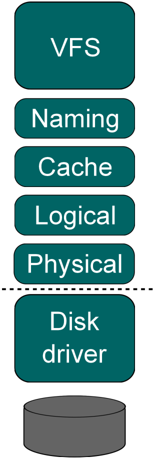


Loris - Division of Labor



Loris: A Hybrid Storage Framework

- Functionalities required to support DST/Caching
  - Collecting access statistics to classify data
  - Transparent background migration
- We extended the Logical Layer
  - Exploiting the logical file abstraction
  - Access statistics reflect “real” storage workload
- Flexible, modular plugin-based realization
  - Data collection plugin gathers statistics
  - Migration plugin handles transparent migration



Loris: Data Collection Plugin

- Several access statistics proposed by prior research
  - Extent-level IOPS and bandwidth statistics (EDT-DST)
  - Block-level access frequency (Azor-Caching)

- Our current implementation is based on Inverse Bitmaps
  - Compute a bitmap value for each read/write operation

$$b = 2^{6 - \lfloor \log_2(N) \rfloor} \tag{1}$$

  - Bitmap value added to a per-file, in-memory counter
  - Counter value indicates "hotness" of each file
    - Prioritizes small, random I/Os over large, sequential ones

Loris-based Hybrid Systems

- All Loris-based hybrid systems are file based
  - Migration/Caching at whole-file granularities
  - File granularity is only a limitation of the current prototype
- Inverse Bitmaps used for "hot" data identification
  - Data collection techniques are architecture neutral
- All hybrid systems share the SSD cleaner implementation
  - Cleaning is triggered reactively
    - Side effect of writes that encounter a lack of space
    - Both foreground and background writes trigger cleaning

Loris-based Conventional Hybrid Systems

- Popular DST/Caching variants
  - Interval-driven Hot-DST
    - Migrates "hot" files to SSD tier periodically
    - Migration interval is an important design parameter
  - On-demand Write-through Caching
    - Caches "hot" files in SSD tier as a side effect of read operation
    - Updates both copies on writes

Loris-based Unconventional Hybrid Systems

- On-demand Hot-DST
    - Migrate “hot” files as a side-effect of read operation
    - Absence of a data copy in contrast to Caching
- Interval-driven Write-through Caching
    - Periodically cache “hot” files in SSD
    - Low-overhead SSD cleanup in contrast to DST
- On-demand Cold-DST
    - Initially allocate all files in SSD
    - SSD cleaner evicts “cold” files to accommodate new files
    - Migrate back once-cold, but now-hot files from HDD

Benchmarks and Workload Generators: Quirks

- Used variety of benchmarks/workload generators
  - File Server, Web Server, and Mail Server workload types
  - Parameters: file size, dir depth, r/w ratio, etc.
- **Benchmarks lack locality by default**
  - Uniform/random access pattern
  - Grossly underestimates effectiveness of DST/Caching
  - Need to extract workload properties from file system traces
- **Beware of transaction-bound benchmarks**
  - PostMark unlike FileBench is transaction bound
  - Interval-based systems might fail to reach equilibrium

Results: Caching vs Hot-DST

- Caching excels in read-heavy workloads (WebServer)
  - Interval-driven/on-demand Caching faster than DST
  - Cheap SSD cleanup by cached copy invalidation
- Hot-DST excels in write-heavy workloads (FileServer)
  - Interval-driven/on-demand DST faster than Caching
  - No expensive synchronization of cached copy
- Is write-back caching worth the complexity?
  - Complicates consistency/availability maintenance
  - But offers Cache-like read and DST-like write performance

Results: Caching vs Hot-DST (2)

- On-demand migration/caching systems outperform their interval-driven counterparts
  - Quick responsiveness in read-heavy workloads
  - But why is this the case in write-heavy workloads?
- "Append-Read"–Inverse Bitmap interaction
  - An append operation first reads last file block
  - A single block read results in high increment to access counter
  - Actual write buffered in OS cache
  - On-demand migration migrates/caches file to/in SSD
  - SSD services the write operation at a later time
- Need for semantic awareness
  - In the long run, append reads fill SSD with write-only logs
  - Being file aware, Loris can isolate append reads

Results: Cold-DST

- Cold-DST outperforms rest under most workloads
  - Buffering allocation writes in the SSD tier boosts performance
  - Scan-resistant Inverse Bitmaps retains hot files in the SSD tier
  - Scales better as it avoids unnecessary background migration
  - Configuration free unlike Interval-driven systems
- Workload patterns also favor Cold-DST architecture
  - 90% of newly created files are opened less than 5 times
  - Proactive cold migration can exploit SSD parallelism to improve performance

Results: Cold-DST (2)

- Cold-DST systems share several advantages with write-back caching without their disadvantages
  - No synchronization overhead for maintaining consistency
  - Efficient space utilization
  - Admitting allocation writes and sieving "cold" data writes
- However, more research is required to address
  - Accelerated SSD wear due to excessive writes
    - Are recent SW/HW reliability techniques sufficient?
  - Performance deterioration caused by using all SSD capacity
    - Can over provisioning solve this problem?
  - Performance deterioration caused by high-cost random writes
    - Is random write performance still an issue with modern SSDs?

Conclusion

- Hybrid storage systems are effective and efficient
- No one architecture fits all workloads - Not yet!
  - Can Cold-DST be the last word in hybrid architectures?
  - How does Cold-DST stack up wrt write-back caching?
- Pairing workloads with ideal architectures
  - Preliminary results under virtualized workloads are encouraging
- More results/details in the paper