

The logo for Fusion-io, featuring the word "FUSION-io" in a white, sans-serif font. The "i" and "o" are lowercase, while "FUSION" is uppercase. A registered trademark symbol (®) is located at the top right of the "o". The logo is set against a dark blue background with abstract, glowing blue light trails and shapes that suggest motion and technology.

FUSION-io®

The background of the lower half of the slide is a light gray grid with faint, glowing binary code (0s and 1s) scattered across it, creating a digital or data-centric aesthetic.

Optimizing I/O Operations via the Flash Translation Layer

Gary Orenstein, VP of Products, @garyorenstein



FLASH PATH FORWARD

FUSION-io

- Flash: new media or new architecture
- Flash Translation Layer Best Practices
- Optimization examples
 - File Systems
 - Caching
 - Database
- Developer opportunities



FLASH: NEW MEDIA OR NEW ARCHITECTURE

FUSION-io

Is GPS technology a new map or new architecture?





CONVENTIONAL ARCHITECTURE

FUSION-io

Applications and File Systems

Storage Stack

Physical Device Operations



STUCK ON DRIVES AND BLOCK I/O

FUSION-io

Applications and File Systems

Block I/O

read()

write()

Physical Device Operations

Entirety of
software
and
physical
stacks
optimized
for rotating
disks



FLASH ONLY AS A FAST DISK

FUSION-io

Applications and File Systems

Storage Stack

Physical Device Operations

Flash

Flash

Flash

Flash

Flash

Disk-
centric
approach

Legacy
stacks
remain



BUT FLASH IS DIFFERENT

FUSION-io

- Asymmetric read/write latencies
- Write-impact on durability
- Unique erase characteristics



FLASH AS A NEW ARCHITECTURE

FUSION-io

Applications and File Systems

Flash-centric approach

Flash Translation Layer

Retain backwards compatibility with conventional block I/O

Flash

Flash

Flash

Flash

Flash



FLASH TRANSLATION LAYER 101

FUSION-io

Input

Logical Block Address (LBA)

Flash Translation Layer

Output

Commands to NAND flash

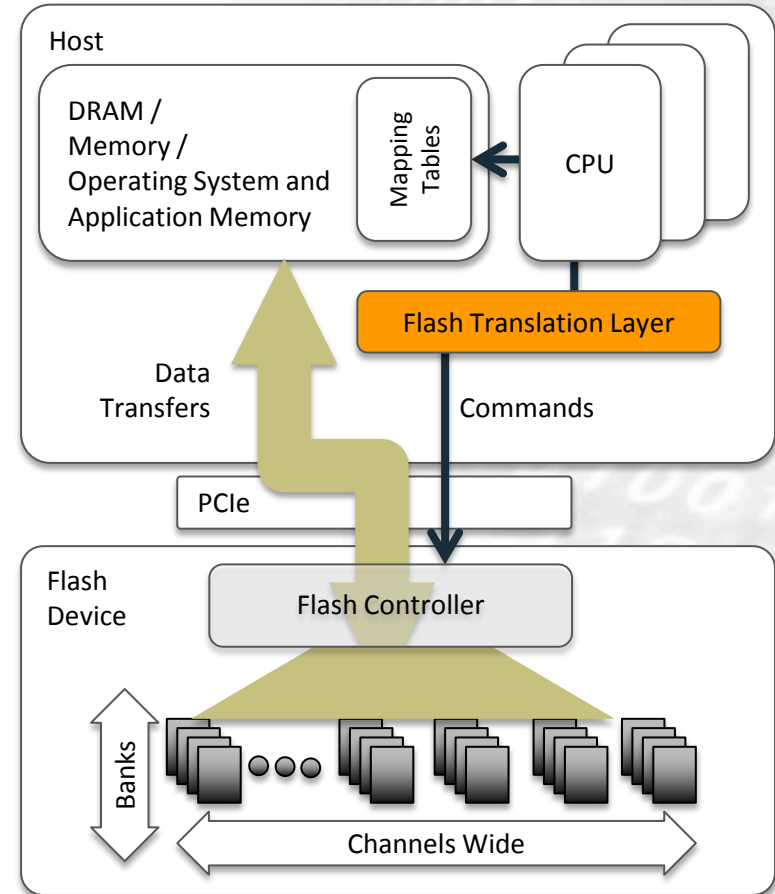
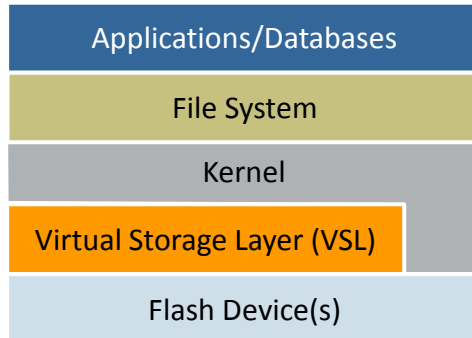


- Virtualize the storage abstraction layer
- Provide a large virtual block address space
- Be backwards compatible with conventional block I/O
- Deliver new capabilities
 - Combine virtualization with intelligent translation and allocation strategies; hide bulk erasure latencies; perform wear leveling



OPTIMIZED FTL

- Sophisticated architecture
 - maximum performance
- Intelligent software
 - advanced features





BENEFITS OF VIRTUAL ADDRESS SPACE

FUSION-io

Large, virtualized, block address space provides:

1. Client software direct access to flash memory
 - single level store fashion
 - across multiple flash memory devices
2. Frees applications and databases from details of virtual to physical flash memory pages
3. Flat, virtual block-addressed space is backwards compatible with conventional block I/O



HOST-CENTRIC APPROACH

FUSION-io

- Cooperate with hardware support
- Maintain virtual to physical mappings
- Handle multiple devices
- Log structured allocation strategy
 - Bulk erasure
 - Wear leveling
 - Bad page recovery
- Richer interface than currently available



File Systems



DFS: A FILE SYSTEM FOR VIRTUALIZED FLASH STORAGE

FUSION-io

DFS: A File System for Virtualized Flash Storage

William K. Josephson
wkj@CS.Princeton.EDU

Lars A. Rongey
larab@Princeton.EDU

David Flynn
dflynn@fusionio.COM

Kai Li

li@CS.Princeton.EDU

Abstract

This paper presents the design, implementation and evaluation of Direct File System (DFS) for virtualized flash storage. Instead of using traditional layers of abstraction, our layers of abstraction are designed for directly accessing flash memory devices. DFS has two main novel features. First, it lays out its files directly in a very large virtual storage address space provided by FusionIO's virtual flash storage layer. Second, it leverages the virtual flash storage layer to perform block allocations and atomic updates. As a result, DFS performs better and it is much simpler than a traditional Unix file system with similar functionalities. Our microbenchmark results show that DFS can deliver 90,000 I/O operations per second (IOPS) for direct reads and 71,000 IOPS for direct writes with the virtualized flash storage layer on FusionIO's iSeries. For direct access performance, DFS is consistently better than ext3 on the same platform, sometimes by 20%. For buffered access performance, DFS is also consistently better than ext3, and sometimes by over 140%. Our application benchmarks show that DFS outperforms ext3 by 7% to 25% while requiring less CPU power.

1 Introduction

Flash memory has traditionally been the province of embedded and portable consumer devices. Recently, there has been significant interest in using it in enterprise file systems for laptops as well as the servers in data centers. Compared with magnetic disk drives, flash can substantially improve reliability and random I/O performance while reducing power consumption. However, these file systems are originally designed for magnetic disks which may not be optimal for flash memory. A key systems design question is to understand how to build the entire system stack including the file system for flash memory.

Past research work has focused on building firmware and software to support traditional layers of abstractions for backward compatibility. For example, recently proposed techniques such as the flash translation layer (FTL) are typically implemented in a solid-state disk controller with the disk drive abstraction [5, 6, 26, 3]. Systems software then uses a traditional block storage interface to support file systems and database systems designed and op-

timized for magnetic disk drives. Since flash memory is substantially different from magnetic disks, the mainstay of our work is to study how to design new abstraction layers including a file system to exploit the potential of NAND flash memory.

This paper presents the design, implementation, and evaluation of the Direct File System (DFS) and describes the virtualized flash memory abstraction layer it uses for FusionIO's iSeries hardware. The virtualized storage abstraction layer provides a very large, virtualized block addressed space, which can greatly simplify the design of a file system while providing backward compatibility with the traditional block storage interface. Instead of pushing the flash translation layer into disk controllers, this layer combines virtualization with intelligent translation and allocation strategies for hiding bulk erasure latencies and performing wear leveling.

DFS is designed to take advantage of the virtualized flash storage layer for simplicity and performance. A traditional file system is known to be complex and typically requires four or more years to become mature. The complexity is largely due to three factors: complex storage block allocation strategies, sophisticated buffer cache designs, and methods to make the file system crash-recoverable. DFS dramatically simplifies all three aspects. It uses virtualized storage spaces directly as a true single-level store and leverages the virtual to physical block allocations in the virtualized flash storage layer to avoid explicit block allocations and reallocations. By doing so, DFS uses extremely simple metadata and data layout.

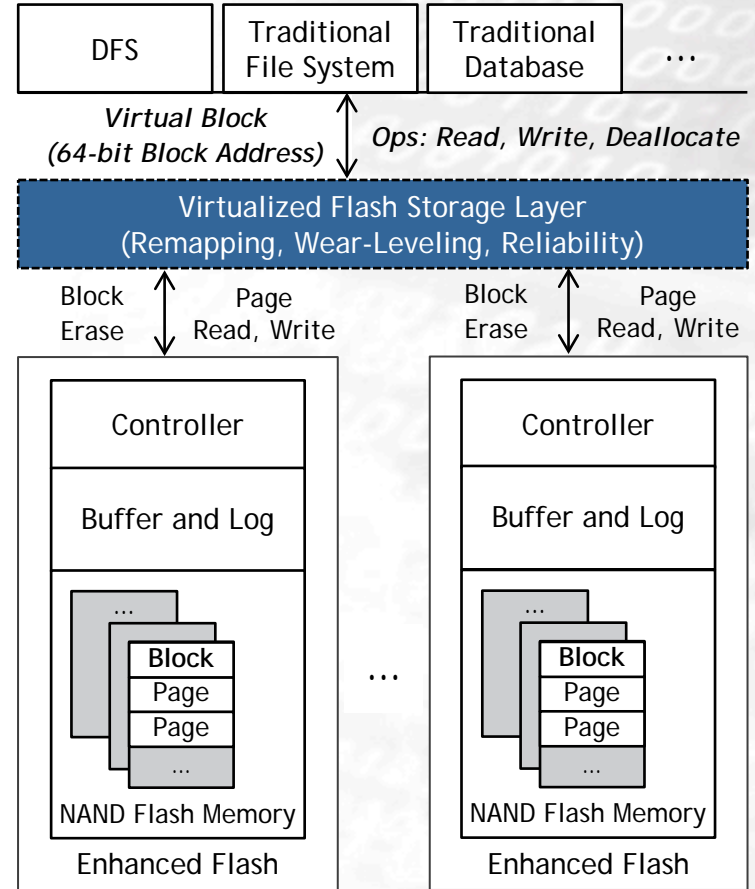
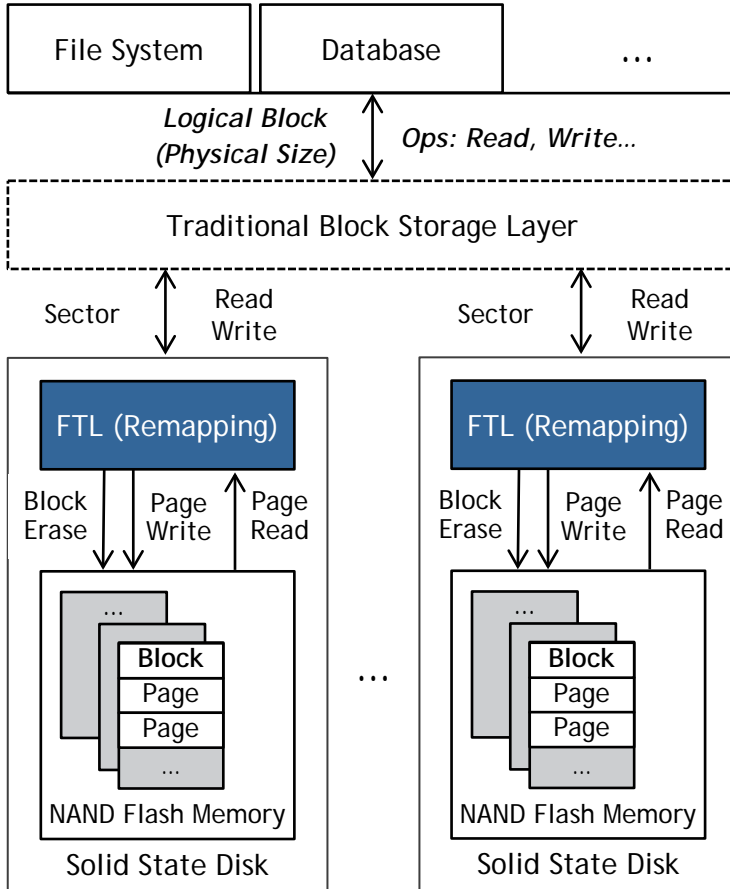
As a result, DFS has a short datapath to flash memory and encourages users to access data directly instead of going through a large and complex buffer cache. DFS leverages the atomic update feature of the virtualized flash storage layer to achieve crash recovery.

We have implemented DFS for the FusionIO's virtualized flash storage layer and evaluated it with a suite of benchmarks. We have shown that DFS has two main advantages over the ext3 filesystem. First, our file sys-

http://www.usenix.org/event/fast10/tech/full_papers/josephson.pdf



FLASH STORAGE ABSTRACTIONS





- Full fledged UNIX file system
- Employ virtualized flash storage layer's
 - Large virtualized addressed space
 - Direct flash access
 - Crash recovery mechanisms



DFS PERFORMANCE

FUSION-io

Device	Read IOPS	Write IOPS
Conventional SSD and FTL	33,400	3,120
Optimized SSD and virtual flash storage layer	98,800	71,000

http://www.usenix.org/event/fast10/tech/full_papers/josephson.pdf



DFS SIMPLICITY - LINES OF CODE

FUSION-io

Module	DFS	Ext3
Headers	392	1583
Kernel Interface (Superblock, etc.)	1625	2973
Logging	0	7128
Block Allocator	0	1909
I-nodes	250	6544
Files	286	283
Directories	561	670
ACLs, Extended Attrs.	N/A	2420
Resizing	N/A	1085
Miscellaneous	175	113
Total	3289	24708



Caching



ALL FLASH TRANSLATION LAYERS DO GARBAGE COLLECTION



1. Four pages (A-D) are written to a block (X). Individual pages can be written at any time if they are currently free (erased)



2. Four new pages (E-H) and four replacement pages (A'-D') are written to the block (X). The original A-D pages are now invalid (stale) data, but cannot be overwritten until the whole block is erased.



3. In order to write to the pages with stale data (A-D) all good pages (E-H & A'-D') are read and written to a new block (Y) then the old block (X) is erased. This last step is *garbage collection*.



INTELLIGENT CACHING LAYER

FUSION-io

Applications and File Systems

Caching

Flash Translation Layer

Physical Device Operations



INTEGRATION BENEFITS

FUSION-io

- Flash translation and garbage collection is complex
- TRIM demonstrates that flash needs information from upstream stack to perform efficiently
- TRIM is only a first step

- Caches maintain intelligence on data
- Uncoupled caching and FTL layers could be working against each other
- Linking cache intelligence to FTL can improve FTL efficiency, write performance, endurance



Databases

Atomic Writes



BEYOND BLOCK I/O: RETHINKING TRADITIONAL STORAGE PRIMITIVES

FUSION-io

Beyond Block I/O: Rethinking Traditional Storage Primitives *

Xiangyong Ouyang¹, David Nellans¹, Robert Wipfel¹, David Flynn¹, Dhabelewar K. Panda¹
¹ FusionIO and ²The Ohio State University

Abstract

Over the last twenty years the interfaces for accessing persistent storage within a computer system have remained essentially unchanged. Simply put, write, read and write have defined the fundamental operations that can be performed against storage devices. These three interfaces have endured because the devices within storage subsystems have not fundamentally changed since the invention of magnetic disks. Non-volatile (flash) memory (NVM) has recently become a viable enterprise grade storage medium. Initial implementations of NVM storage devices have chosen to export these same disk-based read/write interfaces because they provide compatibility for legacy applications. We propose there is a new class of higher order storage primitives beyond simple block I/O that high performance solid state storage should support.

One such primitive, atomic-write, batches multiple I/O operations into a single logical group that will be persisted as a whole or rolled back upon failure. By moving write-atomicity down the stack into the storage device, it is possible to significantly reduce the amount of work required at the application, filesystem, or operating system layers to guarantee the consistency and integrity of data. In this work we provide a proof of concept implementation of atomic-write on a modern solid state device that leverages the underlying log-based flash translation layer (FTL). We present an example of how database management systems can benefit from atomic-write by modifying the MySQL InnoDB transactional storage engine. Using this new atomic-write primitive we are able to increase system throughput by 3.5x, improve the 90th percentile transaction response time by 20%, and reduce the volume of data written from MySQL to the storage subsystem by as much as 67% on industry standard benchmarks, while maintaining ACID transaction semantics.

1 Introduction

Storage interfaces have remained largely unchanged for the last twenty years. The abstraction of reading and writing a 512B block to persistent media has served us well

*This work was supported in part by NSF grant CCF-0821484, CCF-0836362, and CCF-0937842.

but the advent of non-volatile memory (NVM) has produced a flood of new storage products which no longer rely on spinning magnetic media to persist data. The dominant NVM technology in use today, NAND Flash [19], has performance characteristics that are dissimilar to prior storage media. There are many benefits to NVM technologies, such as fast random reads and low static power consumption. However asymmetric read/write latency and low write-durability do not allow a simple linear mapping of a logical block address (LBA) onto a physical block address (PBA) of high throughput and enterprise class data integrity are desired.

Most high capacity solid state storage (SSS) devices implement a logical to physical mapping within the device known as a flash translation layer (FTL) [15]. The design of this FTL has direct implications on the performance and durability of the SSS device and significant effort [10, 11, 17, 20, 21] has gone into optimizing the FTL for performance, power, durability, or a combination of these properties. Optimization of the FTL is often a complex co-design of hardware and software where, at the highest level, the input to the FTL is a logical block address (LBA) and the output is commands to the NAND-flash media on which the data is stored. The LBA read/write interface to the FTL is a simple way to interact with SSS devices. However, these legacy interfaces force solid state storage to behave merely as a very fast block device, ignoring any potential value or optimizations that could be provided by utilizing unique aspects of the flash translation layer's management of the physical device. We believe the time has come for additional I/O interfaces to be defined that can leverage the FTL to provide new and interesting storage semantics for applications.

In this work we propose one such native storage interface, atomic-write, that allows multiple I/O operations to be issued as a single atomic unit with rollback support. We implement atomic-write by leveraging the log based mapping layer within an existing FTL and show that this new interface can provide additional functionality to the application with no performance penalty over traditional read/write interfaces. We target database management systems (DBMS) as a driving application in need of atomic-write and modify MySQL's InnoDB storage en-

<http://www.cse.ohio-state.edu/~zhang/hpca11-submitted.pdf>



IT IS ABOUT TRANSACTIONS

FUSION-io

- Building block of applications and databases



TRANSACTION SEMANTICS

FUSION-io

- Data Integrity
- Concurrency
- Crash Recovery



TOKEN ACID SLIDE 😊

FUSION-io

- **Atomicity**
 - Database modifications must follow an "all or nothing" rule
- **Consistency**
 - Any transaction the database performs will take it from one consistent state to another.
- **Isolation**
 - Other operations cannot access data that has been modified during a transaction that has not yet completed
- **Durability**
 - Ability to recover the committed transaction updates against any kind of system failure (hardware or software).
- <http://en.wikipedia.org/wiki/ACID>



TRANSACTIONAL SEMANTICS APPLY

FUSION-io

Across:

- Applications
- File Systems
- Databases
- Web Services
- Search Engines
- Mission Critical Computing



ATOMIC WRITES

FUSION-io

- Batch multiple I/O operations into a single logical group
- Multiple I/Os are persisted as a whole or rolled back upon failure



ATOMIC WRITES TODAY

FUSION-io

- Handled by
 - Applications
 - Databases
 - File Systems
- Guarantee the consistency and integrity of data
- Databases support atomic write through
 - Logs
 - Locks
 - Buffers
 - Process Management



NOT ALL APPLICATIONS THE SAME

FUSION-io

- Many do not fit the RDBMS model perfectly
- Opportunities exist to
 - Optimize efficient access
 - Provide more control
 - Improve application specific data layout
 - Improve application specific data access mechanisms



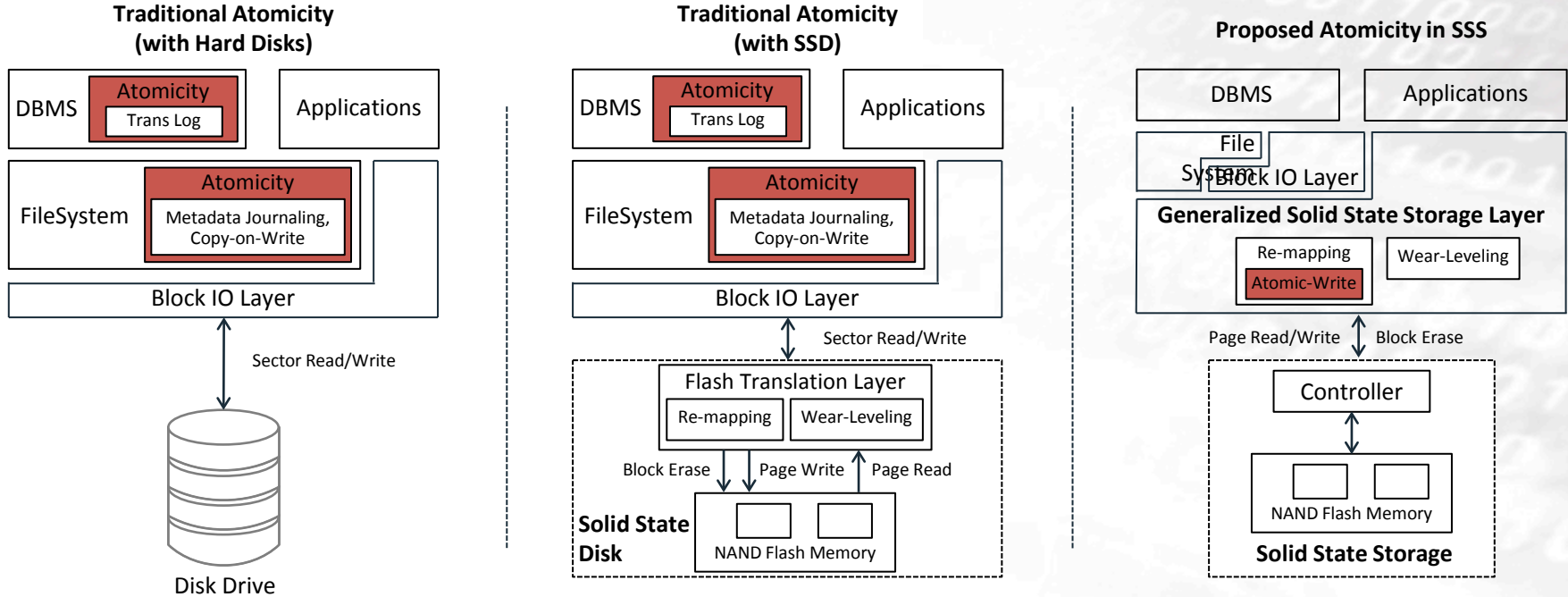
ATOMIC WRITE POTENTIAL

FUSION-io

- Leverage underlying, log-based Flash Translation Layer
- Reduce load on applications and databases
- Simplify Atomic Write execution



ATOMIC WRITES - OPTIMIZED



Moving the Atomic-Write Primitive into Storage Stack



INITIAL DEMONSTRATION

FUSION-io

- MySQL and InnoDB
- Early testing
 - 33% speedups to TPC-C and TPC-H
 - Reduced write bandwidth requirement by 43%
 - Increased endurance with write reduction



Call to Action



FLASH MERITS A NEW SOFTWARE ARCHITECTURE

FUSION-io

- Host-based FTLs integrate and scale with applications, examples include
 - File Systems
 - Caching
 - Databases
- Power of FTL no longer restricted by traditional block interfaces
- Opportunity for performance, simplicity and reliability improvements



For more

go@fusionio.com

www.fusionio.com

THANK YOU