Animesh Trivedi, Nikolas Ioannou, Bernard Metzler, Patrick Stuedi, Jonas Pfefferle, Ioannis Koltsidas
IBM Zurich Research
HIGH-PERFORMANCE IO

1 → 100 Gbit/sec, with ~1 usec link latencies

Rise of NVM devices, multi GBs/sec with ~10s usec device latencies

Marginal improvements
HIGH-PERFORMANCE IO

1 → 100 Gbit/sec, with ~1 usec link latencies

The notion of “fast CPU and multiple slow IO devices” is no longer valid

Marginal improvements

Rise of NVM devices, multi GBs/sec with ~10s usec device latencies
High-Performance IO

1 → 100 Gbit/sec, with ~1 usec link latencies

Traditional IO stacks built assuming slow IO and fail to deliver performance

Rise of NVM devices, multi GBs/sec with ~10s usec device latencies

Marginal improvements
WHY UNIFY NETWORK AND STORAGE IO?

- Exposing high-speed networking performance to the user application:
  - Polling, direct hardware access, OS-bypass, zero-copy data movement, RDMA, DPDK…

- Exposing NVM device performance to the user application:
  - Polling, direct hardware access, OS-bypass, zero-copy data movement, NVMe, SPDK...

Proposal: Unify network and storage IO

→ FlashNet!
The Problem Scenario

Key-Value Stores, Distributed Overlay File Systems e.g., Hadoop.
THE PROBLEM SCENARIO

Key-Value Stores, Distributed Overlay File Systems e.g., Hadoop.
THE PROBLEM SCENARIO

Key-Value Stores, Distributed Overlay File Systems e.g., Hadoop.

performance = network IO + storage IO + server time
userspace

kernel

1. TCP/IP processing
SERVER SIDE - A DETAILED LOOK: SEND

1. TCP/IP processing
2. receive request
3. receive request

userspace
kernel
1. TCP/IP processing
2. receive processing
3. receive request
4. fs translation
5. block IO

11
SERVER SIDE - A DETAILED LOOK: SEND

userspace

kernel

1. TCP/IP processing

2. receive processing

3. receive request

4. fs translation

5. block IO

6. flash IO completion
**SERVER SIDE - A DETAILED LOOK: SEND**

1. TCP/IP processing
2. receive processing
3. receive request
4. fs translation
5. block IO
6. flash IO completion
7. response transmission
8. send
9. TX done

userspace

kernel
SERVER SIDE - A DETAILED LOOK: SENDFILE

1. TCP/IP processing
2. receive processing
3. receive request
4. fs translation
5. block IO
6. flash IO completion
7. send
8. response transmission
8. TX done

userspace

kernel
SERVER SIDE - A DETAILED LOOK: SENDFILE

1. TCP/IP processing
2. receive processing
3. receive request
4. fs translation
5. block IO
6. flash IO completion
7. send
8. response transmission

userspace
kernel
THE FLASHNET APPROACH

1. TCP/IP processing

2. receive processing

3. receive request

4. fs translation

5. block IO

Eliminate application involvement

userspace

kernel
THE FLASHNET APPROACH

1. TCP/IP processing
2. receive processing
3. receive request
4. fs translation
5. block IO

Eliminate application involvement
Reduce file system overheads
THE FLASHNET APPROACH

1. TCP/IP processing
2. receive processing
3. receive request
4. fs translation
5. block IO

Eliminate application involvement
Reduce file system overheads
Enable direct network and storage interaction
THE FLASHNET APPROACH

userspace  

kernel

2. RDMA processing

1. TCP/IP processing

RDMA

 Eliminate application involvement

Reduce file system overheads

Enable direct network and storage interaction

4. fs translation

5. block IO
THE FLASHNET APPROACH

1. TCP/IP processing
2. RDMA processing
3. 
4. 
5. block IO

RDMA
contiguous FS layout

userspace
kernel

Eliminate application involvement
Reduce file system overheads
Enable direct network and storage interaction
THE FLASHNET APPROACH

Userspace

Kernel

1. TCP/IP processing

2. RDMA processing

3. Use VM

4. Contiguous FS layout

5. Block IO

Eliminate application involvement

Reduce file system overheads

Enable direct network and storage interaction
**FLASHNET IO OPERATION**

1. TCP/IP processing
2. RDMA processing
3. va to block
4. block IO
5. flash IO completion
6. response transmission
7. TX done

**userspace**

**kernel**
FLASHNET: A UNIFIED IO STACK

RDMA controller
file system

flash controller
FLASHNET: A UNIFIED IO STACK


FLASHNET: A UNIFIED IO STACK

network control setup expanding to storage


FLASHNET: A UNIFIED IO STACK

network control setup expanding to storage
data path from a flash device to a client buffer


process address space
ANATOMY OF A FLASHNET OPERATION

process address space

area1

area2

① mmap

FNFS

dir1

file1

file2

DRAM pool

LBAs

virtualized FTL

PBAs
ANATOMY OF A FLASHNET OPERATION

process address space

area1

area2

1. mmap

RNIC

3. - save vm
   - generate STag

2. reg_mr

FNFS

dir1

file1

file2

DRAM pool

LBAs

virtualized FTL

PBAs

OpenFabrics Alliance Workshop 2016
ANATOMY OF A FLASHNET OPERATION

1. mmap

2. reg_mr

3. RNIC
   - save vm
   - generate STag

4. RDMA read req.

5. resolve STag
   - request pages

process address space

area1

area2

FNFS

dir1

file1

file2

DRAM pool

virtualized FTL

LBAs

PBAs

OpenFabrics Alliance Workshop 2016
ANATOMY OF A FLASHNET OPERATION

process address space

area1

area2

RNIC

③ reg_mr

④ RDMA
read req.

⑤ -save vm
-generate STag

⑥ -resolve STag
-request pages

⑦ lookup and IO

FNFS

dir1

file1

file2

virtualized FTL

PBAs

LBAs

DRAM pool

iomem_get

1 mmap

2

OpenFabrics Alliance Workshop 2016
ANATOMY OF A FLASHNET OPERATION

1. `mmap`
2. `reg_mr`
3. RNIC
   - save vm
   - generate STag
4. RDMA read req.
5. - resolve STag
6. - request pages
7. `iomem_get`
   - lookup and IO
8. response transmitted

- RNIC
- process address space
- area1
- area2
- FNFS
- file1
- file2
- LDAs
- virtualized FTL
- PBAs
ANATOMY OF A FLASHNET OPERATION

process address space

① mmap

area1

area2

② reg_mr

RNIC

③ save vm

- generate STag

⑤ - resolve STag

- request pages

⑦ lookup and IO

⑨ iomem_put

⑧ response transmitted

④ RDMA read req.

⑥ iomem_get

FNFS

dir1

dir1

dir2

dir2

dir3

dir3

file1

file2

DRAM pool

virtualized FTL

PBAs

LBAs

virtualized FTL
How efficient is FlashNet's IO path?

9-machine cluster testbed

CPU : dual socket E5-2690, 2.9 GHz, 16 cores
DRAM : 256 GB, DDR3 1600 MHz
OS : Linux 3.19 kernel
NIC : 40Gbit/s Ethernet
Flash : 1.3 GB/sec (read), 680 MB/sec (write)

peak read IOPS: 360K, chip latency: 50μsec
PERFORMANCE - IOPS EFFICIENCY

single core @ server
4kb random access
one outstanding req
ext4/fn fs
1 server, 8 client hosts
• FlashNet reads are almost 50% more efficient
PERFORMANCE - CORE SCALING

4kb random access
one outstanding req
ext4/fn fs
1 server, 8 client hosts
128 client processes
FlashNet IO operations scale better with respect to per-core scaling
APPLICATION: RSTORE ON FLASHNET

[1] RStore: A Direct-Access DRAM-based Data Store, Trivedi et al., ICDCD’15

control setup
data access
APPLICATION: RSTORE ON FLASHNET

RStore: A Direct-Access DRAM-based Data Store, Trivedi et al., ICDCD'15
APPLICATION: RSTORE ON FLASHNET

- **4 lines** to enable FlashNet
- Flash managed by RStore's control API
- Flash accessed by RStore's data API
- Run RSort on FlashNet/Rstore
- 4 servers with direct-attached SSDs

[1] RStore: A Direct-Access DRAM-based Data Store, Trivedi et al., ICDCD'15
RSTORE ON FLASHNET: TERASORT

Time in Seconds

- DRAM-only

Data Set Size

- 16 GB
- 32 GB
- 64 GB
- 128 GB

[Graph showing time in seconds for different data set sizes]
RSTORE ON FLASHNET: TERASORT

**Time in Seconds**

- **DRAM processing + I/O time**
- **DRAM-only**

**Data Set Size**

- **16 GB**
- **32 GB**
- **64 GB**
- **128 GB**

- **DRAM processing + I/O time**
- **DRAM-only**
RSTORE ON FLASHNET: TERASORT

<table>
<thead>
<tr>
<th>Data Set Size</th>
<th>16 GB</th>
<th>32GB</th>
<th>64GB</th>
<th>128GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time In Seconds</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FlashNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRAM processing + I/O time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRAM-only</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FlashNet adds minimum overheads to RDMA-ready applications
CONCLUSIONS

- CPU-centric IO stacks incur overheads
- Solution: Apply unified path separation (ctrl/data) and RDMA access models to both storage and network IO stacks
- Implemented software prototype that benefits from unified storage/network access semantics
- Demonstrated performance gains for
  - a unified end-host network-storage stack
  - a distributed data store.
- More FlashNet benefits:
  - Client transparent
  - Byte-granular access to storage
  - Easy storage tiering
  - Obsoletes overhead of network storage access protocol
- Exploring HW implementation
THANK YOU
BACKUP
RSTORE

API calls:
1. reserve()
2. alloc/init()
3. map()
4. read/write()

Application Stacks:
- PageRank
- SSSP
- Pregel Engine
- Query Engine
- Distributed Graph Store
- Distributed KV Sorter

64-bit Distributed Namespace
Striped data access

Master → Memory Servers
High-performance RDMA Network

[1] RStore: A Direct-Access DRAM-based Data Store, Trivedi et al., ICDCD’15
PBA state says a flash Logical Block Address (LBA) is stored on a Physical Block Address (PBA) on a device not in a DRAM page
FLASHNET: FLASH CONTROLLER

Read-only, In flight

RD_GET

WR_GET

PBA

Read modify write, In flight
 FLASHERNET: FLASH CONTROLLER

Read-only, In flight

Read modify write, In flight

IO_DONE

Clean Page

PBA

TX data

TX data

RX data

IO_DONE

Dirty Page

RD_GET

WR_GET
FLASHNET: FLASH CONTROLLER

- Read-only, In flight
- Read modify write, In flight
- Dirty, In flight
- Clean Page
- Dirty Page

Transitions:
- RD_GET
- WR_GET
- RD_GET, WR_GET
- LAST_PUT
- IO_DONE