MultiLanes: Providing Virtualized Storage for OS-level Virtualization on Many Cores

Junbin Kang, Benlong Zhang, Tianyu Wo, Chunming Hu, and Jinpeng Huai

Beihang University
Outline

• Background
• MultiLanes Design
• Evaluation
• Related Work
• Conclusion
Background

• Many-core architecture
  – Common in modern processor
  – Requests consolidation for high performance

• Virtualization
  – Efficient method to improve performance and utilization of hardware.

• Non-Volatile Memory
  – Higher percentage of software overhead
### Hypervisor-based Virtualization vs. OS-level Virtualization

<table>
<thead>
<tr>
<th>Hypervisor-based Virtualization</th>
<th>OS-level Virtualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>- E.g., Xen, KVM</td>
<td>- E.g., VServer, LXC</td>
</tr>
</tbody>
</table>

#### Deep IO stack -> Poor performance

**Container:** A virtualized environment (VE)
OS-level Virtualization

• Performance bottleneck of software
  – Especially for NVM-based fast storage

[Diagram showing containerization and virtualization layers with notes on shared data structure and interferences]
Motivation

- Scalability issues
  - FS: Ext3
MultiLanes Design

- MultiLanes
  - A storage system for OS-level virtualization that addresses the I/O performance interference between multiple VEs on many cores.

- Design Goals:
  - Simple, self-contained, transparent to applications and FS
  - Good scalability
  - Low virtualization overhead on fast storage
MultiLanes Design

- MultiLanes Architecture

  Container: a set of processes (actually)

  pVFS: partitioned VFS

  Different FSs

  vDriver: virtualized device driver

  Virtualized Storage: a file in host FS
Virtualized Storage

- **Traditional OS-level virtualization**
  - Stores VM’s data on the host FS directly
  - Results in contention on shared data structure and locks

- **MultiLanes**
  - Maps a regular file in host FS as a virtualized block device
  - Incurs overhead
vDriver

• Key work
  – Mapping virtualized resources to physical ones.

• Two components
  – **Block translation**: maps a logical block of a file to the physical blocks on the host device
  – **Request handling**: handles IO requests of virtualized device
    • A single IO request of virtualized device may be translated into multiple IO requests of host device.
Slice: a IO request on the host block device.
Block Translation

• **Cache Table**
  – Maintains the mapping between logical blocks and physical blocks

• **Job Queue**
  – Stores translation job

• **Translation Thread**
  – Block translation
    • Invokes the mapping interface of host FS to get the target physical block
    • Stores a new mapping entry in the cache table
    • Wakes up the container thread
Request Handling

• **Slice**
  – A new block I/O request on the host block device

• **BIO list**
  – Doubly-linked list
  – Each slice is submitted to host device driver in sequence

• **I/O completion**
  – Offer a I/O completion callback to the host driver
Example

- Request Mapping
Partitioned VFS

- **Hot VFS Locks**

<table>
<thead>
<tr>
<th>#</th>
<th>Hot VFS Locks</th>
<th>Hot Invoking Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>inode_hash_lock</td>
<td>insert_inode_locked()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_remove_inode_hash()</td>
</tr>
<tr>
<td>2</td>
<td>dcache_lru_lock</td>
<td>dput()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dentry_lru_prune()</td>
</tr>
<tr>
<td>3</td>
<td>inode_sb_list_lock</td>
<td>evict()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inode_sb_list_add()</td>
</tr>
<tr>
<td>4</td>
<td>rename_lock</td>
<td>write_seqlock()</td>
</tr>
</tbody>
</table>

- **Method in MultiLanes**
  - Allocate an inode hash table and a dentry hash table for each container
  - Use separate locks for guest FS
Evaluation

- **Experimental Setup**
  - Intel 16-core machine with 64GB memory
  - 16 LXC containers
  - 40GB Ram disk as fast storage device
  - FS: Ext3, Ext4, XFS, Btrfs

- **Microbenchmarks**
  - Ocrd: runs 64K transactions (create, rename, and delete files)
  - IOzone: writes 4KB data to a file that ends up with 256MB size

- **Macrobenchmarks**
  - Filebench: mail server, file server
  - MySQL
Results

- Ocrd
  - Ext3

![Graph showing throughput vs. number of containers with different file systems and locks]

<table>
<thead>
<tr>
<th>Lock</th>
<th>Ext3</th>
<th>Ext4</th>
<th>XFS</th>
<th>Btrfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>inode_hash_lock</td>
<td>1092k</td>
<td>960k</td>
<td>114k</td>
<td>228k</td>
</tr>
<tr>
<td>dcache_lru_lock</td>
<td>1023k</td>
<td>797k</td>
<td>583k</td>
<td>5k</td>
</tr>
<tr>
<td>inode_sb_list_lock</td>
<td>239k</td>
<td>237k</td>
<td>144k</td>
<td>106k</td>
</tr>
<tr>
<td>rename_lock</td>
<td>541k</td>
<td>618k</td>
<td>446k</td>
<td>252k</td>
</tr>
</tbody>
</table>

Table 3: Contention Bounces. The table shows the contention bounces using MultiLanes without pVFS.
Results

- **IOzone**
  - Sequential writes

(a) Buffered write on Ext3

(b) Buffered write on XFS

XFS: delays block allocation and metadata journaling
Results

- **IOzone**
  - Random writes

(a) Buffered write on Ext3

(c) Buffered write on XFS

XFS: competitive
Results

- Filebench

<table>
<thead>
<tr>
<th>Workload</th>
<th># of Files</th>
<th>File Size</th>
<th>I/O Size</th>
<th>Append Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varmail</td>
<td>1000</td>
<td>16KB</td>
<td>1MB</td>
<td>16KB</td>
</tr>
<tr>
<td>Fileserver</td>
<td>2000</td>
<td>128KB</td>
<td>1MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Table 4: **Workload Specification.** This table specifies the parameters configured for Filebench Varmail and Fileserver workloads.

(a) Mail server on Ext3  
(e) File server on Ext3
Results

- MySQL

(a) MySQL on Ext3

(b) MySQL on Ext4
Overhead Analysis

- **Single container**
- **Benchmarks**
  - Apache Build: I/O less intensive
  - Webserver: read intensive
  - Streamwrite: write intensive
Related Work

- **Performance isolation**
  - Space partitioning or time multiplexing hardware resources (e.g., CPU, memory, disk)
    - E.g., VSever (allocating and scheduling physical resources), Cgroup (limit, account, and isolate resource usage of process groups)

- **Kernel scalability**
  - Partitioning hardware resources
    - Hive, Barrelfish (multi-kernel model)
  - Virtualization layer
    - Diso (running multiple VMs on shared-memory multiprocessors)

- **Device virtualization**
  - Device emulation, e.g., network cards, SCSI devices
  - Para-virtualization, e.g., KVM, Xen
  - Direct device assignment
Conclusion

- OS-level virtualization suffers from storage performance interference, especially for fast storage device.
- MultiLanes
  - Provides an isolate I/O stack to eliminate contentions between multiple VEs on many cores.