CS-580K/480K Advanced Topics in Cloud Computing

Storage Virtualization
Where we are

Cloud operating system

Virtualization Layer

Network Virtualization

Storage Virtualization

VM1 VM2 VM3

Operating System

VM1 VM2 VM3

Operating System

VM1 VM2 VM3

Operating System
How to Provide Virtualization to Storage Resources?
Data Storage Systems

What is a storage system used for?

--- Simply speaking, storing data

Data can be stored in various places in different manners

--- Hardware: CPU registers, caches, main memory and persistent storage

--- Software: File systems, object storage, databases (SQL databases and No-SQL databases.)
Storage I/O system within a single host

Persistent Storage media

**FLASH**

**HDD OR DISK DRIVE**
I/O layers within a single host

User space

Kernel space

Applications
System Call Interfaces
VFS
File System (ext3, ext4, btrfs)
Page Cache
Generic Block Layer
I/O schedulers
Block device driver
I/O layers within a VM

- Applications
- System Call Interfaces
- VFS
- File System (ext3, ext4, btrfs)
- Page Cache
- Generic Block Layer
- I/O schedulers
- Block device driver
I/O layers in Virtualization

VM1

Hypervisor/Host

Applications
System Call Interfaces
VFS
File System (ext3, ext4, btrfs)
Page Cache
Generic Block Layer
I/O schedulers
Block device driver

QEMU Device

System Call Interfaces
VFS
File System (ext3, ext4, btrfs)
Page Cache
Generic Block Layer
I/O schedulers
Block device driver
I/O layers in Virtualization

VM1

Hypervisor/Host

Question 1: Dual I/O stack – how to optimize?
I/O layers in Virtualization

Question 2: Dual Page Cache – Is it good?
I/O layers in Virtualization

Question 3: Dual I/O stack – how to virtualize a block device?
Block Device Virtualization

$sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-16.04.3-server-amd64.iso -m 1536
Block Device Virtualization

$ sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-16.04.3-server-amd64.iso -m 1536
Virtualize Block Device

- Virtual disk is stored as a single file on the real file system of the host
- Operations to the block device is emulated by QEMU
- Guest issues block reads & writes
- Qemu converts them to file operations
- All these file operations, just like traditional file operations, operate on the virtual disk file
Virtualize Storage Device

- **Guest Disk**
- **Device Driver**
- **Device Emulation**
- **Map disk operations to File operations**
- **Physical Disk**
- **Device Driver**
- **Guest OS**
- **Real Disk**

**Emulation Layer**
- **Guest virtual device**
- **A File (virtual image file)**

**Host File System**
- **Real Disk**
Disk image type matters!

- A “pre-allocated” disk image (1:1 blocks):
  - A 10 GB disk image reserves 10 GB of disk space, regardless of whether the virtual machine guests uses 1 GB or 10 GB (allocated at creation time)

- An “extensible” disk image, useful for growing on demand
  - From the VM point of view, it sees a full size disk, but the hypervisor is actually lying to the VM, and is allocating the disk blocks on the HOST side on demand
Disk images - pros / cons

• A “pre-allocated” disk image
  ▪ Pros: Fast
  ▪ Cons: Uses all space

• An extensible disk image
  ▪ Pros: Less space
  ▪ Cons: A bit overhead, fragmentation

▪ It depends on what we are trying to achieve: system design tradeoff
Discussion

• Each virtual machine (VM) is going to need a disk image. If we are only going to create a single machine, it’s pretty easy:
  • Create VM
    • (1) create disk image
    • (2) attach ISO image start VM
    • (3) install operating system
    • (4) Done!

• What if we want to install 2 machines? We could probably install a second time. What about when we have to build 5? 40? And do this every week, to teach a class?
Two Concrete Techniques

• Raw disks
  ▪ Byte-for-byte disk image, byte 0 = byte 0 of the disk

• QEMU-KVM’s “QCOW2” (Qemu Copy On Write, v.2) format
  ▪ grow-on-demand
  ▪ compression support
  ▪ encryption support
  ▪ Copy-on-write!
What is Copy-on-Write?

• Traditionally:
  ▪ When programs inside the guest VM write to the virtual disk, the changes are written to the disk image on the host (for example, this could be /var/lib/libvirt/images/master image.qcow2)

• Copy-on-write:
  ▪ Write the delta and store somewhere else (not modify original copy)
  ▪ Relink the files
Use of CoW

- A new disk image, originates from a “master” image as a backing file.
  - E.g, `qemu-img create -o backing_file=master_image.qcow2 -f guest1.qcow2 10G`
- Initially, the size of guest1.qcow2 is 0. System will start using the backing file (path master_image.qcow2), which is 10 GB.
- For reads, KVM will actually read the block from the master image.qcow2.
- For writes, KVM will write the changes to the guest1.qcow2. The file master_image.qcow2 is never written to.
Storage Area Network

• Protocols: iSCSI – Reuse Ethernet Network
Definition of iSCSI

- iSCSI is a Storage Area Network (SAN) protocol that allows for SCSI command transmission over a TCP/IP network
- iSCSI allows for the sharing of I/O devices over a long distance, especially storage devices
  - Typically high speed disk arrays
- iSCSI maintains the SCSI notion of an Initiator and Target device
iSCSI

iscsiadm -m discovery -t sendtargets -p 192.168.1.1
Starting iscsid: [ OK ]
192.168.1.1:3260,1 iqn.2015-06.com.example.test:target1

iscsiadm -m node -T iqn.2015-06.com.example:target1 –login
mkfs.ext4 /dev/sdb

... default-driver iscsi
<target iqn.2015-06.com.example.test:target1>
  backing-store /dev/vdb1
  initiator-address 10.10.1.1
</target>
Both protocols have advantages and disadvantages

- iSCSI
  - iSCSI is typically less expensive
  - iSCSI reuses existing ethernet network

- Fibre Channel
  - Fibre Channel is typically faster
  - A typical Fibre network for FC
Challenges of SAN

- Latency
- Data granularity mismatch

Non-volatile Main Memory  Flash Arrays  High-speed Networking
Block Addressability vs. Byte Addressability?
Data Granularity Mismatch

VMs/Containers

Virtual File System

File Systems

Page Cache

Generic Block Layer

Storage Server Host

Block Devices

Transmission Layer

Block-addressable

Byte-addressable
Non-aligned Write Blocking

VMs/Containers

Virtual File System

File Systems

Page Cache

Generic Block Layer

Block-addressable

I/O size NOT aligned with Page Size (4 KB)

Byte-addressable

Transmission Layer

Storage Server Host

Block Devices
Transmission Amplification

VMs/Containers

Virtual File System

File Systems

Generic Block Layer

Virtual File System

Page Cache

Block-addressable

Non-aligned writes

Non-aligned write blocking

Transmission amplification

Storage Server Host

Byte-addressable

Transmission Layer

Block Devices
Low End-to-End I/O Throughput

The Higher The Better

Ext3
Ext4
XFS
NTFS

I/O Size (bytes)
512 1024 2048 4096

Non-aligned Writes
Well-aligned Writes

Non-aligned Writes
Well-aligned Writes

IOPS
0 5,000 10,000 15,000 20,000 25,000 30,000 35,000 40,000
High Network Bandwidth Consumption

Bandwidth Usage per I/O (KB)

- Ext3
- XFS
- Ext4
- NTFS
- Dirty Bytes

The Lower The Better

I/O Size (bytes)

512
1024
2048
4096

Non-aligned Writes
Well-aligned Writes
Solutions

- Byte-addressable storage stack
A Highly-Efficient Write Solution

VMs/Containers

Virtual File System

File Systems

Generic Block Layer

Page Cache

Non-aligned writes

Client side – Return I/O Threads Immediately

I/O Stack – Flush only Dirtied I/O Data

Storage Server – Conduct Alignment

Storage Server Host

Transmission Layer

Block Devices
Byte-Addressable Storage Stack

VMs/Containers

Virtual File System

File Systems

Generic Block Layer (BASS_Block)

Page Cache (BASS_Cache)

Backend (BASS_Backend)

Storage Server Host

Byte-addressable

Transmission Layer

Block Devices
Byte-Addressable Storage Stack

Applications

One Page

Dirty Page

Page Cache
Byte-Addressable Storage Stack

**Applications**

**One Page**

**Page Cache (BASS_Cache)**

**Dirty Segment Linked-list**

**BASS Cache**: Keep track of application-level accesses at a byte granularity
Byte-Addressable Storage Stack

**BASS Cache**: Keep track of application-level accesses at a byte granularity

**BASS Block Layer**: Generate “Arbitrary-Length” I/O requests

**Too many I/O requests!**
**Byte-Addressable Storage Stack**

**BASS Cache**: Keep track of application-level accesses at a byte granularity

**BASS Block Layer**: Generate “Arbitrary-Length” I/O requests

**Packing dirty segments!**
**Byte-Addressable Storage Stack**

**BASS Cache**: Keep track of application-level accesses at a byte granularity.

**BASS Block Layer**: Generate “Arbitrary-Length” I/O requests.

**BASS Backend**: Conduct “block-level” operations to align with storage devices.

With byte-addressability, File Systems can only flush dirtied bytes!
Non-blocking Writes

- **Generic Block Layer (BASS_Block)**
- **Page Cache (BASS_Cache)**
- **Virtual File System**
- **File Systems**
- **Backend (BASS_Backend)**

**VMs/Containers**

The Synchronous part can return quickly.

The Asynchronous part does NOT block I/O threads of VMs/Containers.
Performance Gains

The Higher The Better

BASS increases the performance of non-aligned writes by 8X, compared to Legacy
BASS greatly reduces the network usage of non-aligned writes by up to 90%
Storage Media
Eventually, All Data Goes To Block Storage
HDD vs. SSD
Spinning Drives Are Slow

Random access to HDD is especially slow
HDD Performance

RANDOM 4KiB I/O PERFORMANCE COMPARISON

<table>
<thead>
<tr>
<th>Reads</th>
<th>Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>514</td>
<td>133</td>
</tr>
</tbody>
</table>

SAS HDD (146GB, 15K RPM)
Performance HDD vs. SSD

**Random 4KiB I/O Performance Comparison**

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS HDD (146GB, 15K RPM)</td>
<td>514</td>
<td>133</td>
</tr>
<tr>
<td>Consumer SSD (400GB, SATA 6Gb/s, MLC)</td>
<td>61,178</td>
<td>34,406</td>
</tr>
</tbody>
</table>
SSD Storage is Emerging

- Solid-state drives (SSDs) are widely adopted in data centers
  - Examples: EMC XtremIO Array, NetApp Sandisk, Micron P420m

- Pros of SSDs:
  - High I/O throughput
  - Low power
  - High reliability

- Cons of SSDs:
  - Wear-out

[Source: http://www.crn.com]
Flash And NAND Gates

Every NAND can be set to 0 individually

To set back to 1, an entire group needs to be reset
How does SSDs Work?

- Organized into blocks
- Each block has 64 or 128 pages of size (e.g., 4KB each)
- Three basic operations: read, write, erase
  - Read, write: per-page basis
  - Erase: per-block basis
How SSDs Work?

Out-of-place page-level write for updates:

- Write to a clean page and mark it as valid
- Mark original page as invalid
- How to deal with these invalid pages
- Garbage Collection
**Garbage Collection**

- **Garbage collection (GC)** needed to reclaim clean pages
  - Choose a block to erase
  - Move valid pages to another clean block
  - Erase the block

- **Challenges:**
  - Blocks can only be erased a finite number of times
    - SLC: 100K; MLC: 10K; 3-bit MLC: several K to several hundred
  - When a block reaches the limit, it wears out
  - Bit error rates increase as blocks wear down

---

Before GC

<table>
<thead>
<tr>
<th>Block A</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After GC

<table>
<thead>
<tr>
<th>Block A</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. write
2. erase
## Sequential vs. Random

### SSD or Flash

<table>
<thead>
<tr>
<th>Write</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EVERYTHING IS RANDOM IO FOR FLASH</strong></td>
<td><strong>ERASE + WRITE</strong></td>
</tr>
</tbody>
</table>

### HDD or Disk Drive

<table>
<thead>
<tr>
<th>Write</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SEQUENTIAL</strong></td>
<td><strong>WRITE</strong></td>
</tr>
<tr>
<td><strong>RANDOM</strong></td>
<td><strong>SEEK/SPIN + WRITE</strong></td>
</tr>
</tbody>
</table>

**SLOWER PERFORMANCE**
Revisit: Performance HDD vs. SSD

**RANDOM 4KiB I/O PERFORMANCE COMPARISON**

- **HDD**
  - Reads: 514 IOPS
  - Writes: 133 IOPS

- **Consumer SSD (400GB, SATA 6GB/s, MLC)**
  - Reads: 61,178 IOPS
  - Writes: 34,406 IOPS

Legend:
- SAS HDD (146GB, 15K RPM)
- Consumer SSD (400GB, SATA 6GB/s, MLC)
Other Issues

- Failures
- Data Storage Efficiency/Cost
- Security
- ...

58
High Reliability

• Data Replication
  ▪ Inter-disk replication
  ▪ Inter-rack replication
  ▪ Inter-datacenter replication
Data Deduplication

- Often called intelligent compression or single instance storage.
- Duplicate data is deleted leaving only one copy of the data to be stored.
- Data deduplication turns the incoming data into segments, uniquely identifies the data segments, and compares these segments to the data that has already been stored. If the incoming data is new data then it is stored on disk, but if it is a duplicate of what has already been stored then it is not stored again and a reference is created to it.
- “Only one unique instance of the data is actually retained on storage media (e.g., disk). Redundant data is replaced with a pointer to the unique data copy.”
Deduplication Methods

- **In-line** deduplication is the most efficient and economic method.
  - Hash calculations are created as the data is entered in real time.
  - If the target device identifies a block that has already been stored then it simply references to the existing block.

- **Pros**: Inline deduplication significantly reduces the raw disk capacity needed in the system since the full, not-yet-deduplicated data set is never written to disk

- **Cons**: However, “because hash calculations and lookups takes so long, data ingestion can be slower thereby reducing the backup throughput of the device.”

- What is off-line deduplication?