VM Migration, Containers (Lecture 12, cs262a)

Ali Ghodsi and Ion Stoica,
UC Berkeley
February 28, 2018

(Based in part on
http://web.eecs.umich.edu/~mosharaf/Slides/EECS582/W16/021516-JunchengLiveMigration.pptx)

Today's Paper

Live Migration of Virtual Machines, C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, NSDI'05

An Updated Performance Comparison of Virtual Machines and Linux Containers, Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio

VMWare “history”

Mendel Rosemblum (Stanford University)

Started by creators of Disco

Initial product: provide VMs for developers to aid with development and testing

- Can develop & test for multiple OSes on the same box

Actual, killer product: **server consolidation**

- Enable enterprises to consolidate many lightly used services/systems
- Cost reduction, easier to manage
- Eventually over 90% of VMWare’s revenue

Migration Motivation

Server becomes overloaded

- Multiple VMs on same server are heavily used
- Load balance the load (e.g., multiple web servers running in VMs)

Maintenance: update the configuration of a machine

- Change/upgrade HDD
- Upgrade guest OS (e.g., Xen)

Thus, need to migrate VMs on a different machine

Why VM instead of Process migration?

Avoid complex dependencies between processes and local services

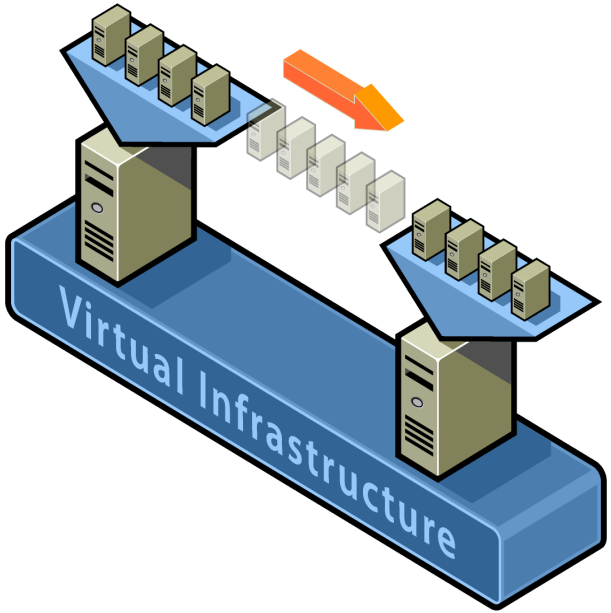
Separation of concerns between users and operators

- Users can fully control the software within their VMs
- Operators don’t care about what’s inside the VM

Live VM Migration

Move VMs across distinct physical hosts transparently

- Little or no downtime for running services
- Services unaware about migration, e.g.,
  - Maintain network connectivity of the guest OS
- VM is treated as a black box

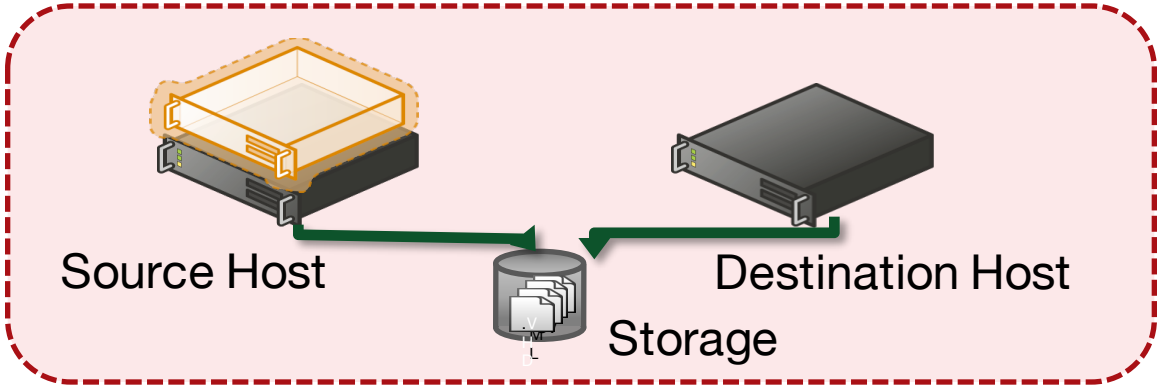


Challenges

Minimize service downtime

Minimize migration duration

Avoid disrupting running service



Handling Resources during Migration

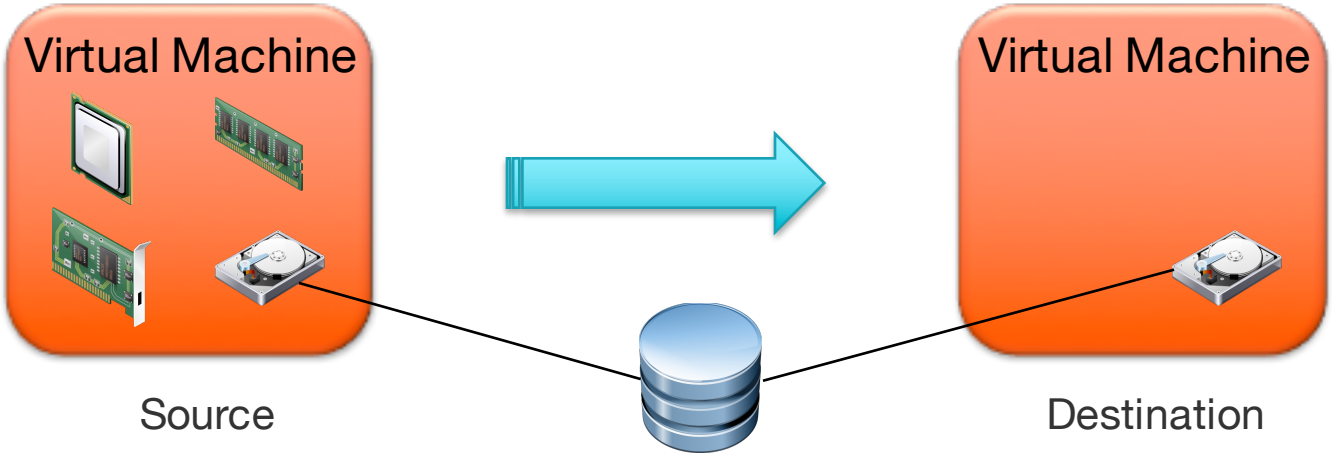
Open network connections

- Keep IP addresses while migrating VM
- Use ARP (Address Resolution Protocol) to map IP to new host MAC
- Broadcast ARP new routing information
  - Some routers might ignore to prevent spoofing
  - However, guest OS aware of migration can avoid this problem

Local storage

- Assume Network Attached Storage

Handling Resources during Migration



Migration Techniques

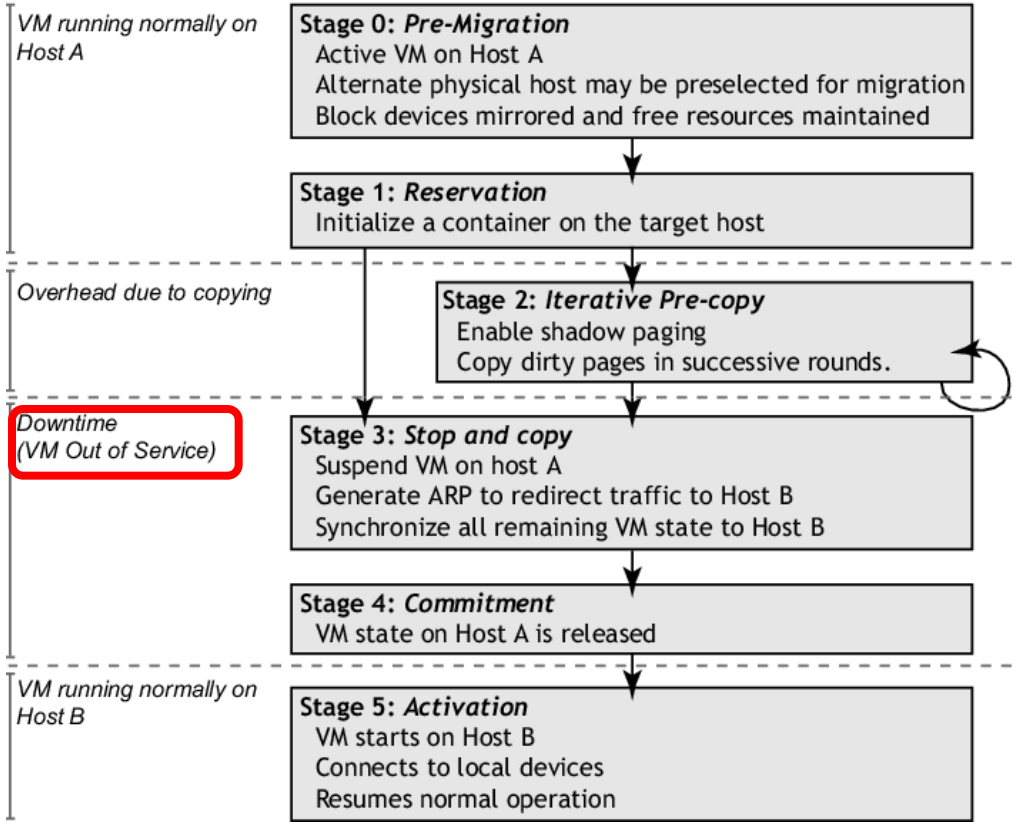
Phase	service downtime	migration duration
push	-	-
stop-and-copy	longest	shortest
pull (demand)	shortest	longest

Pre-copy: bounded

- iterative push phase, plus
- very short stop-and-copy phase

Careful to avoid service degradation

Design Overview



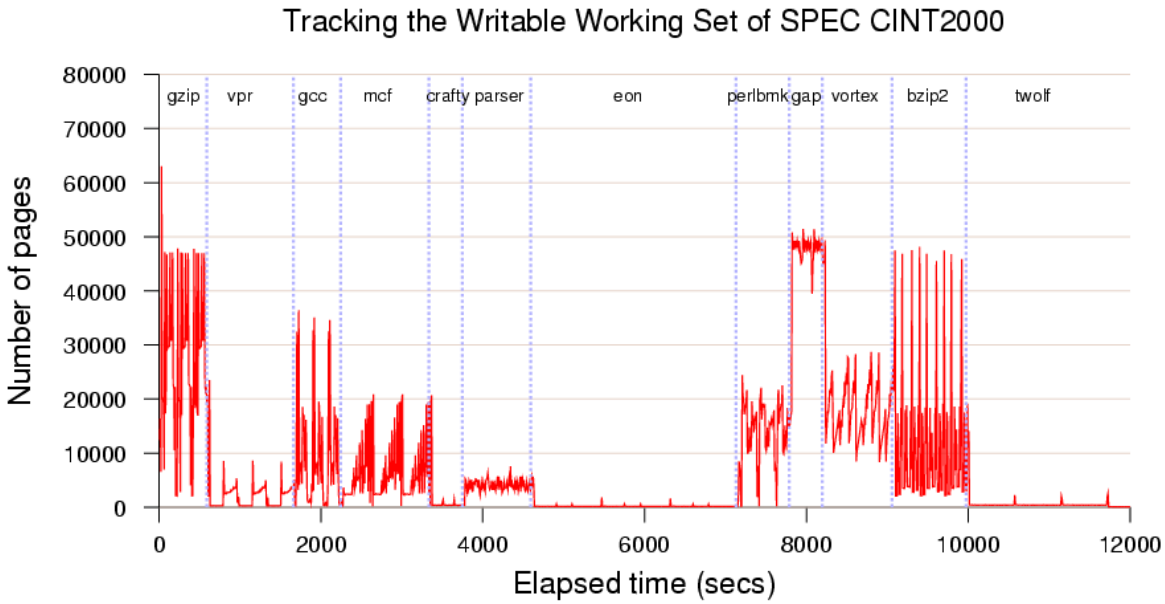
Migrate Writable Working Sets

Transfer memory pages that are subsequently modified

- Good candidates for push phase: pages seldom or never modified.
- Writeable working set (WWS): Pages are written often, and should best be transferred via stop-and-copy

WWS behavior

- WWS varies significantly between different sub-benchmarks
- Migration results depend on workload and when migration happens



WWS behavior

- WWS varies significantly between different sub-benchmarks
- Migration results depend on workload and when migration happens

Managed & Self migration

Managed migration

- Performed by a migration daemon running in management VM

Self migration

- Within migratee OS; a small stub required on destination host

Difference	Managed	Self
Track WWS	shadow page table + bitmap	bitmap + a spare bit in PTE
Stop-and-copy	suspend OS to obtain a consistent checkpoint	two-stage stop-and-copy, ignore page updates in last transfer

Managed Migration (1/2)

Use shadow page table to track dirty pages in each **push round**

1. Xen inserts shadow pages under guest OS
2. Shadow pages are marked read-only
3. If OS tries to write to a page, resulting page fault is trapped by Xen.
4. Xen checks OS's original page table and forwards appropriate write permission
5. Simultaneously, Xen marks page as dirty in bitmap.

Managed Migration (2/2)

At the beginning of next push round

- Last round's bitmap is copied, Xen's bitmap is cleared
- Shadow pages are destroyed and recreated, all write permissions lost

Self Migration

Most implementation within OS being migrated (source machine)

Migration stub must run on destination machine

- Listen for incoming migration requests, create an app

Pre-copying scheme similar

Challenge: transfer a consistent OS checkpoint

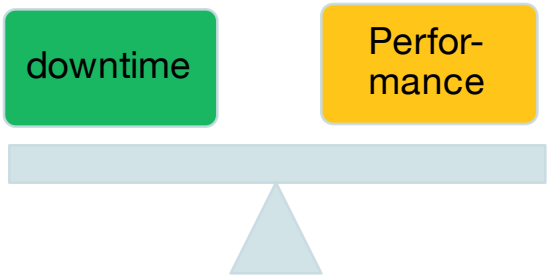
- Cannot suspend migratee (as it is doing migration!)

Logically OS Checkpointing

1. Disables all OS activity except for migration
   - Final scan of dirty bitmap; clear bit as each page is transferred.
   - Pages dirtied during final scan, copied to a shadow buffer

2. Transfer the contents of the shadow buffer
   - Page updates are ignored during this transfer.

Dynamic Rate Limiting



Tradeoff:

- More network bandwidth less downtime
- Less network bandwidth more impact on running services

Dynamically adapt bandwidth limit during each **push** round

– Set a min and a max bandwidth limit, begin with the min limit

$$bandwidth_{next} = \text{dirty } rate_{\text{current}} + constant\ increment$$

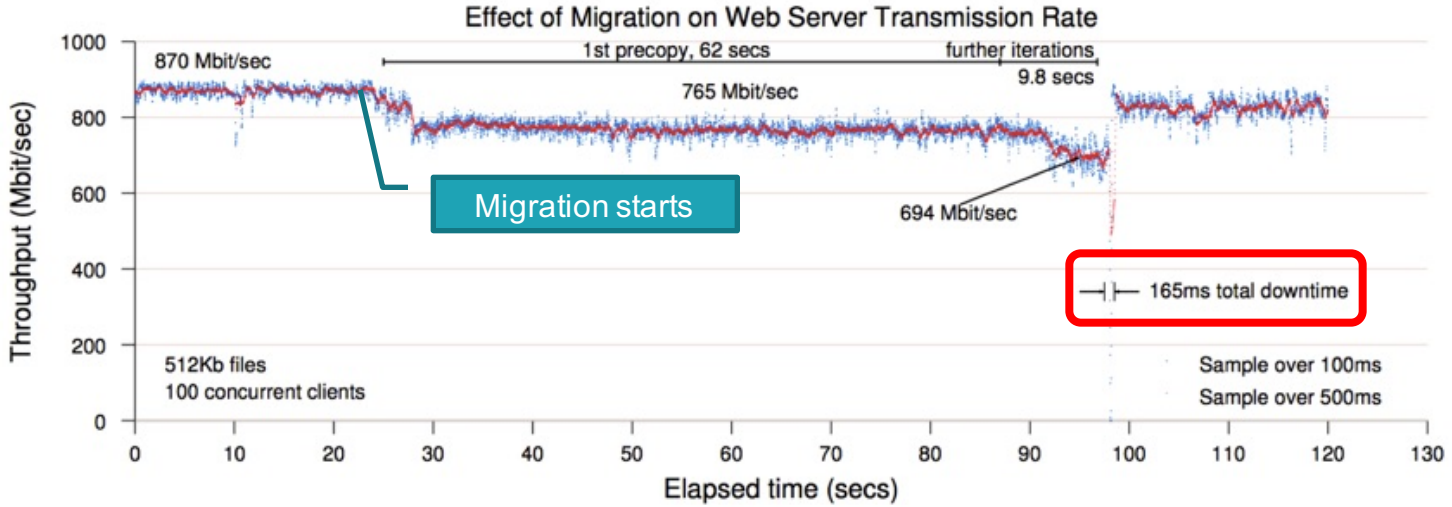
$$dirty\ rate_{current} = dirty\ pages/duration$$

When terminate push, and switch to **stop-and-copy**?

$$dirty\ rate_{\text{current}} > bandwidth_{max}$$

$dirty\ pages < threshold$ (e.g., 256 KB)

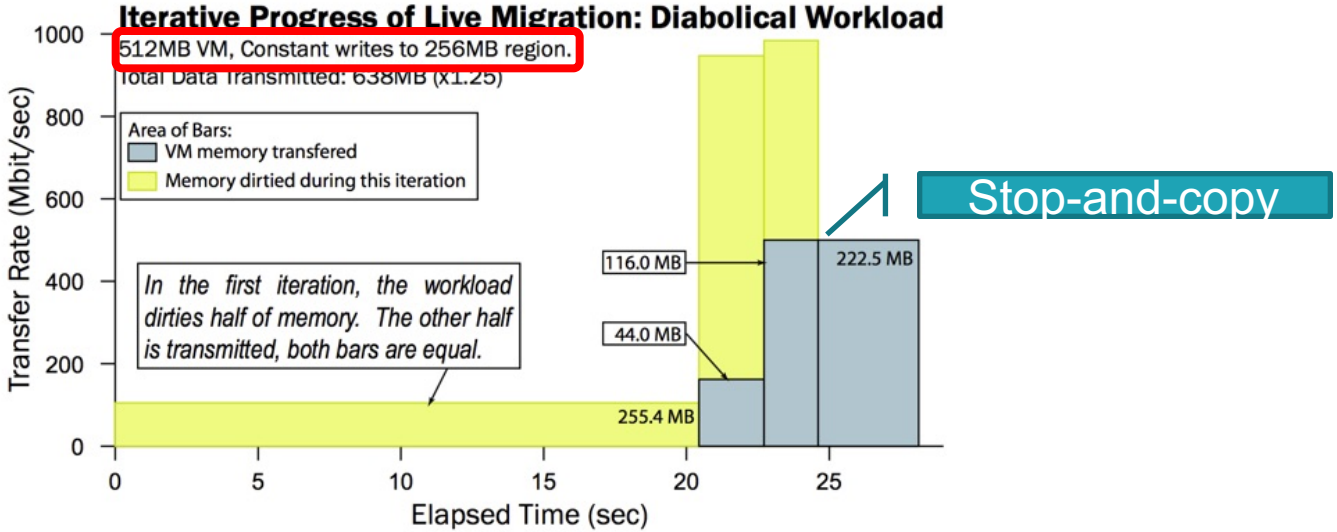
Evaluation-simple web server



A highly loaded server with relative small WWS

- Controlled impact on live services
- Short downtime

Evaluation-rapid page dirtying

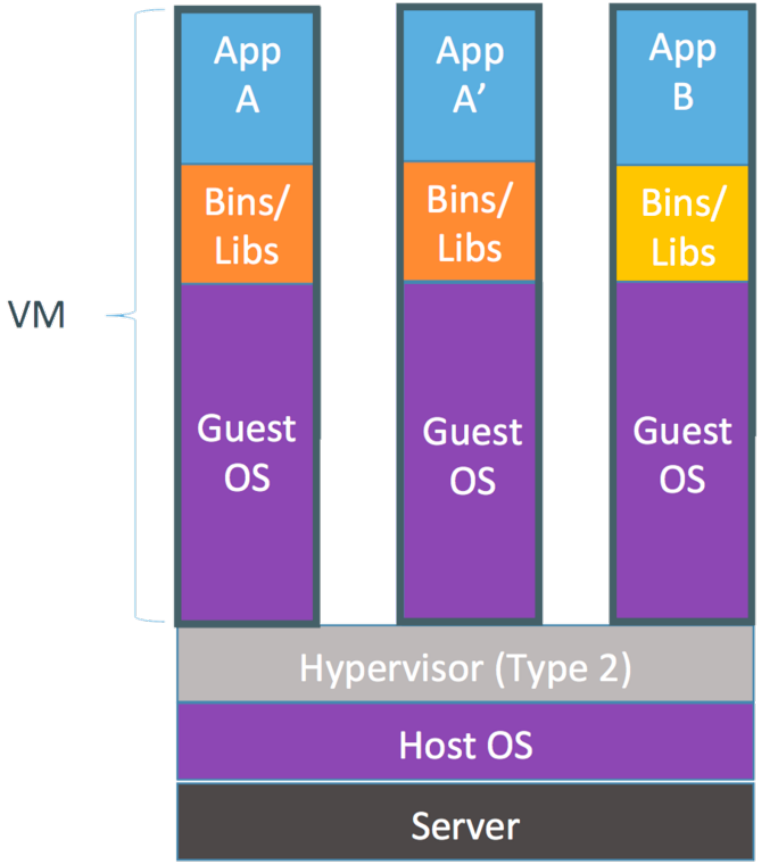


- In the third round, the transfer rate is scaled up to 500Mbit/s (max)
- Switch to stop-and-copy, resulting in 3.5s downtime
- Diabolical workload may suffer considerable service downtime

Conclusion

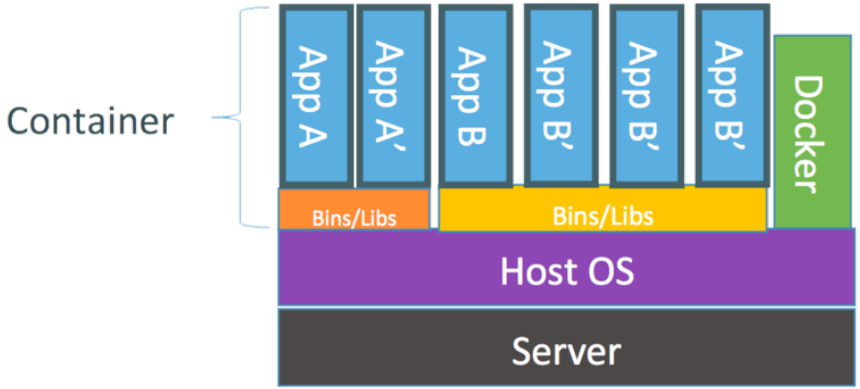
- OS-level live migration
- Pre-copy: iterative push and short stop-and-copy
- Dynamically adapting network-bandwidth
  - Balance service downtime and service performance degradation
- Minimize service downtime and impact on running service

VMs vs Containers



Containers are isolated, but share OS and, where appropriate, bins/libraries

…result is significantly faster deployment, much less overhead, easier migration, faster restart



Linux Containers (LXC)

Linux kernel provides the “control groups” (cgroups) functionality

- Allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc)

“namespace isolation” functionality

- Allows complete isolation of an applications' view of the OS, e.g.,
  - Process trees
  - Networking
  - User IDs
  - Mounted file systems.

LXC Features

Runs in the user space

- Own process space
- Own network interface
- Own /sbin/init (coordinates the rest of the boot process and configures the environment for the user)
- Run stuff as root

Share kernel with the host

No device emulation

Near-zero overhead

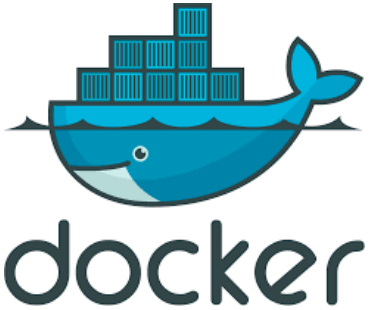
Processes are isolated, but run straight on host

CPU performance = native performance

Memory performance = a few % for (optional) accounting

Network performance = small overhead; can be reduced to zero

Docker



Standard format for containers

Allows to **create and share** *images*

- Standard, *reproducible* way to *easily* build *trusted* images (Dockerfile)

Public repository of Docker images

- https://hub.docker.com/

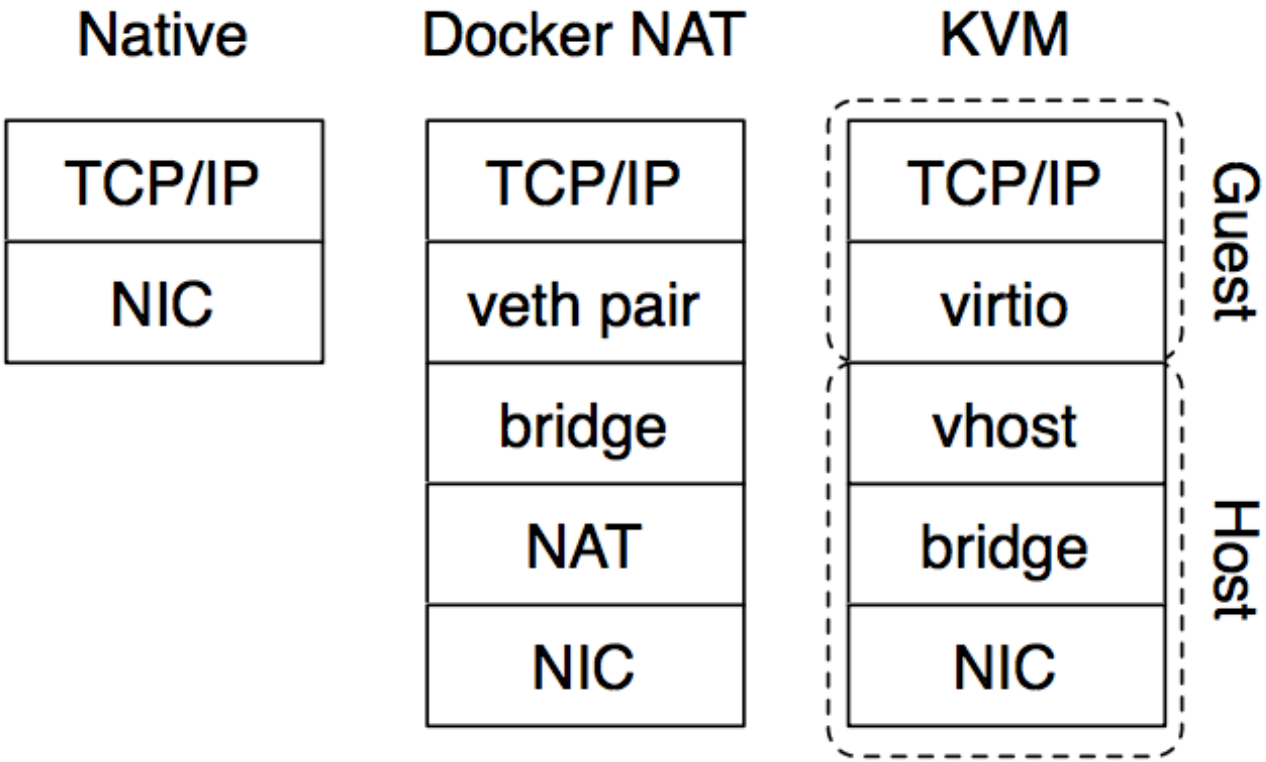
Kernel-based VM

Hypervisor embedded in the Linux kernel itself

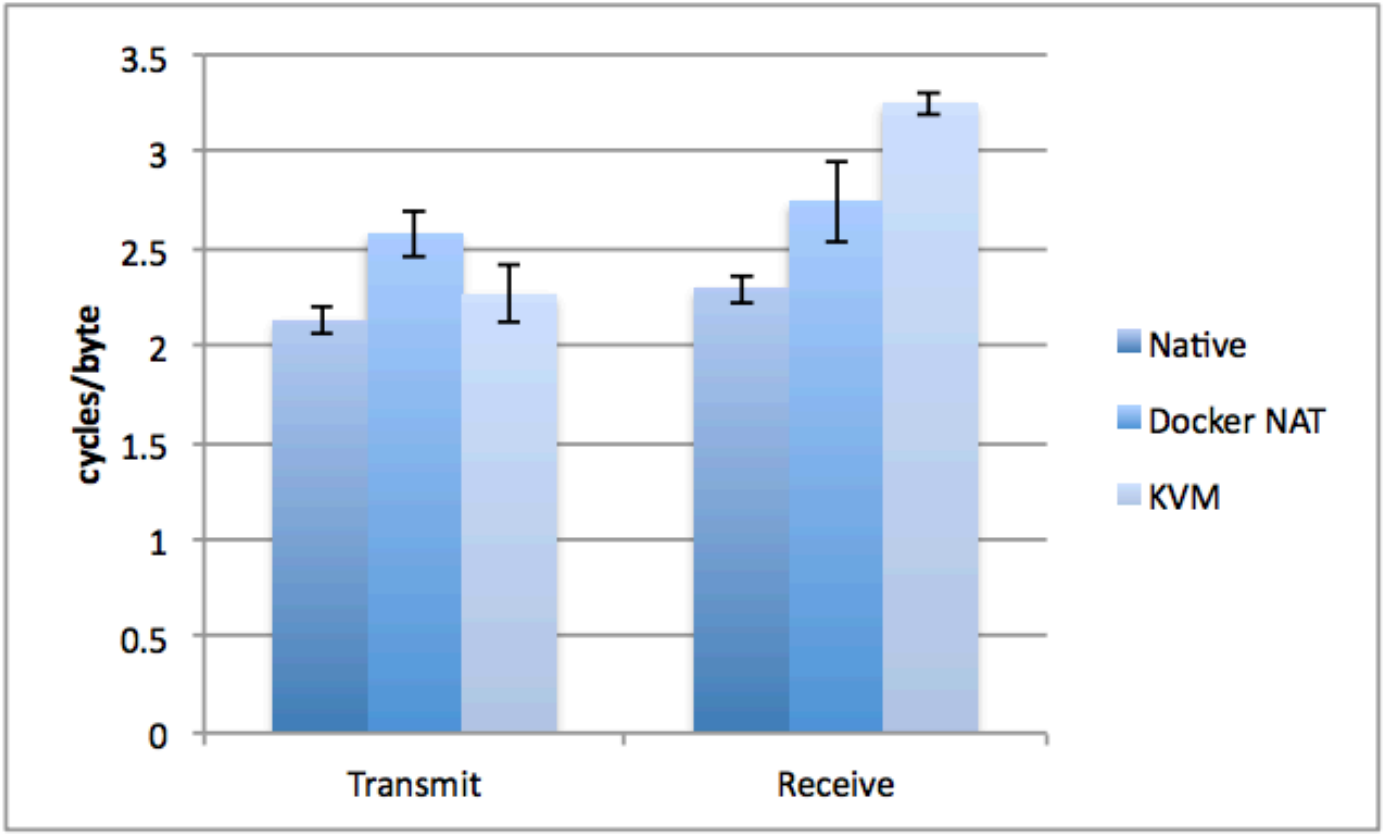
- Every VM is a regular Linux process, scheduled by Linux scheduler
- KVM makes use of hardware virtualization to virtualize processor states

KVM supports live migration

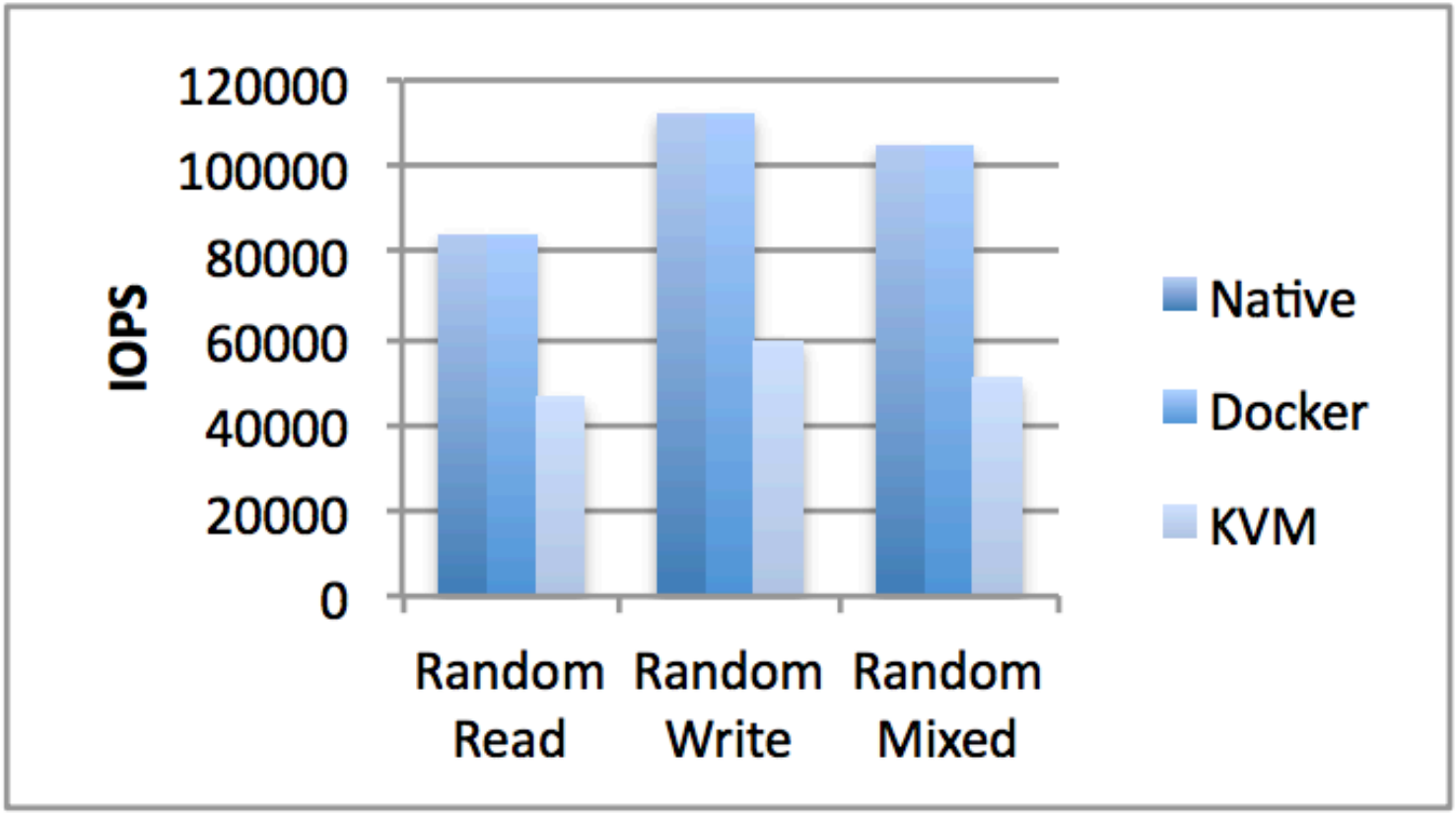
Network Configuration



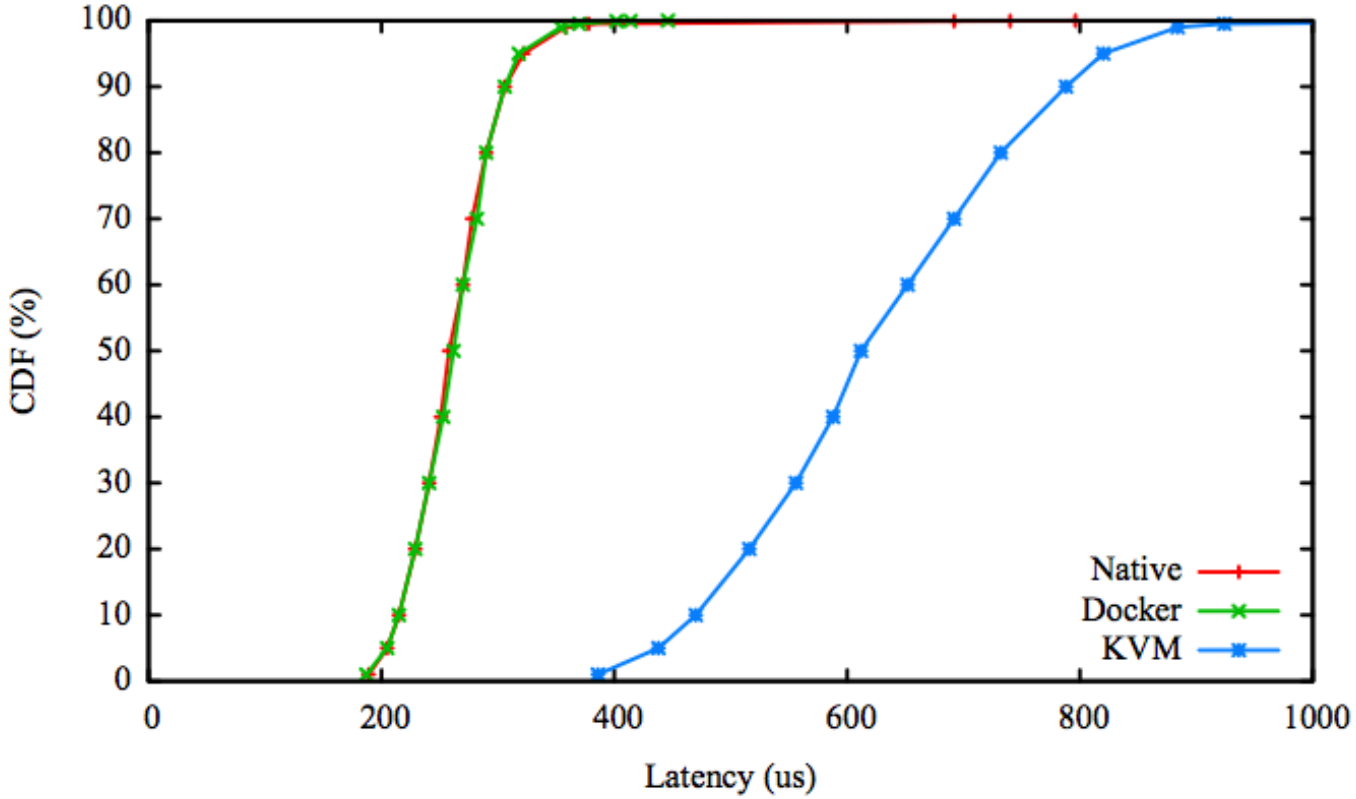
TCP bulk transfer efficiency



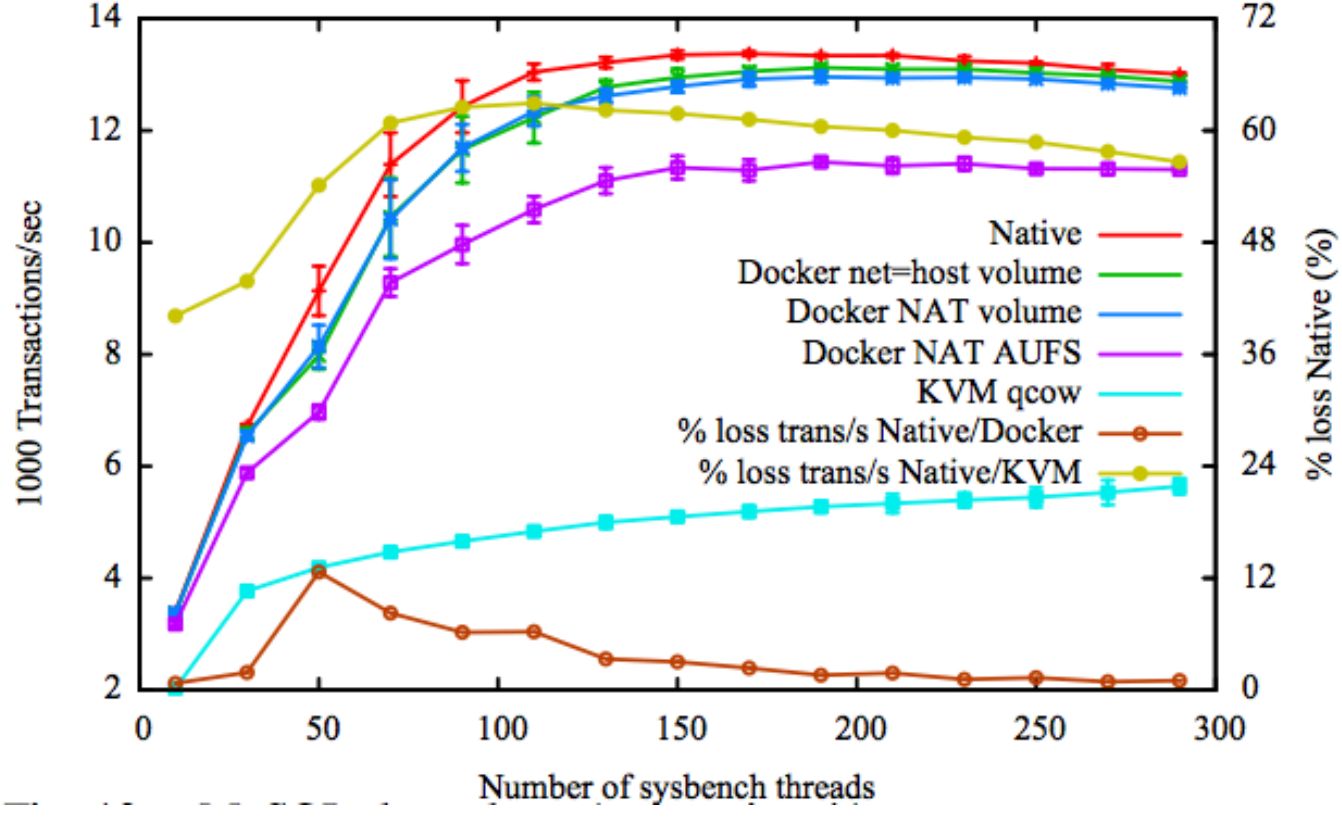
Random I/O Throughput



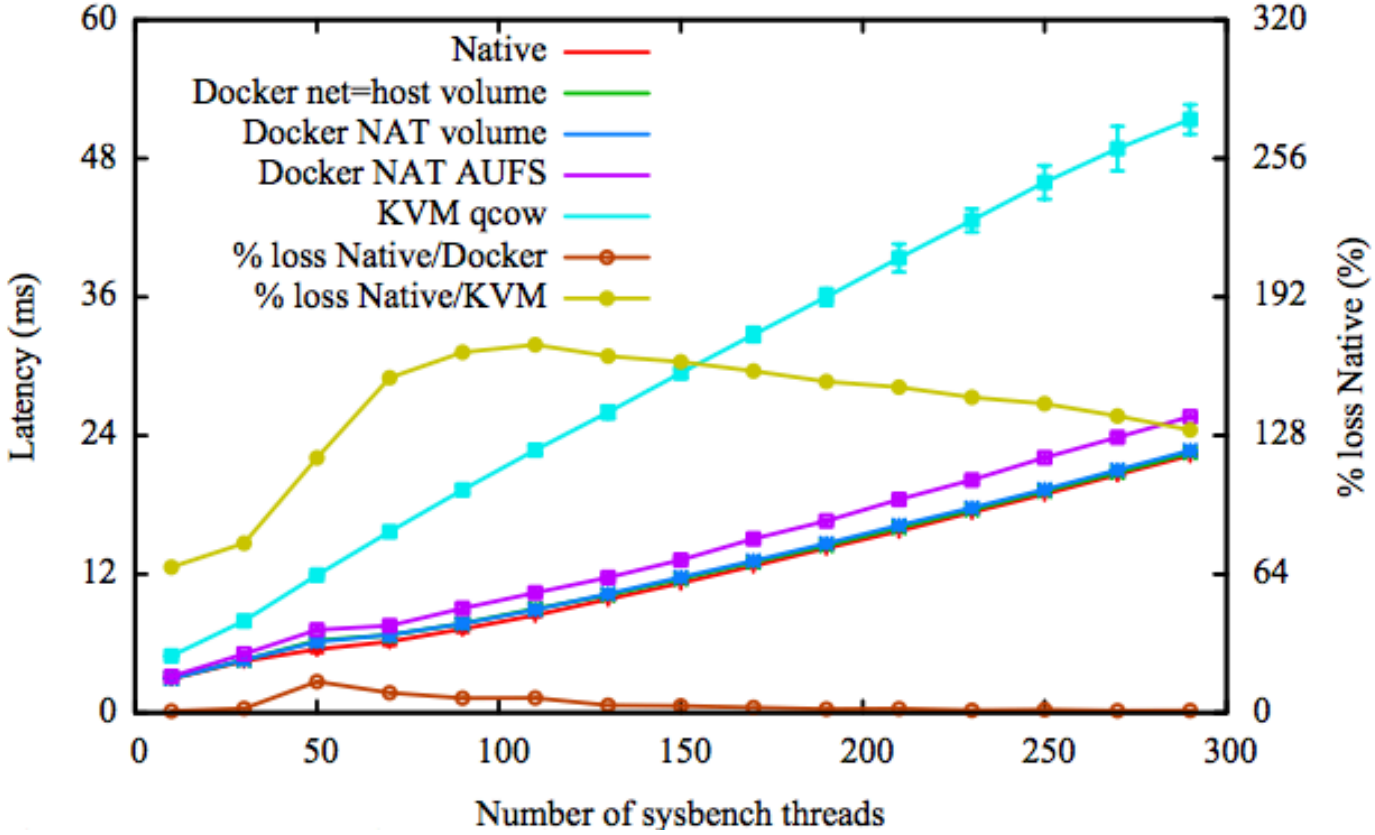
Random read latency CDF



MySQL Throughput vs. Concurrency



MySQL Throughput vs. Concurrency



Summary

KVM performance has improved considerably since its creation

- Leverage virtualization support in modern processors

Docker not without overhead

- E.g., NAT introduces overhead for workloads with high packet rates

“Bad” news for containers?

- Containers started with near-zero overhead so no room to improve!