VMs, Unikernels and Containers: Experiences on the Performance of Virtualization Technologies

Felipe Huici, Filipe Manco, Jose Mendes, Simon Kuenzer
NEC Europe Ltd. (Heidelberg)
In the Beginning...
In the Beginning...

“Tinyfied VMs”
In the Beginning...

“Tinyfied VMs”

unikernels

VM
In the Beginning...

- "Tinyfied VMs"
- unikernels
- containers

VM
In the Beginning...

“Tinyfied VMs”
unikernels
containers
Virt. Technology Benchmarking

• Metrics:
  – VM Image and memory consumption
  – VM creation time
  – Delay
  – Throughput
Virt. Technology Benchmarking

• Metrics:
  – VM Image and memory consumption
  – VM creation time
  – Delay
  – Throughput
Virt. Technology Benchmarking

• Metrics:
  – VM Image and memory consumption
  – VM creation time
  – Delay
  – Throughput
Virt. Technology Benchmarking

• Metrics:
  – VM Image and memory consumption
  – VM creation time
  – Delay
  – Throughput
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput
Virt. Technology Benchmarking

• Metrics:
  – VM Image and memory consumption
  – VM creation time
  – Delay
  – Throughput
Virt. Technology Benchmarking

• Metrics:
  – VM Image and memory consumption
  – VM creation time
  – Delay
  – Throughput

higher overhead

Standard VM

Tinyfied VM

Unikernel

Containers

lower overhead

?
Virt. Technology Benchmarking

- Metrics:
  - VM Image and memory consumption
  - VM creation time
  - Delay
  - Throughput
Virt. Technology Benchmarking

• Metrics:
  – VM Image and memory consumption
  – VM creation time
  – Delay
  – Throughput
Virtualization Technology Benchmarking

• Metrics:
  – VM image and memory consumption: ls, top, xl
  – VM creation time: SYN flood + RST detection
  – Throughput: iperf, guest to host (TCP traffic)
  – RTT: ping flood

• VM-based tests run on both Xen and KVM

• Hardware: x86_64 server with an Intel Xeon E5-1630 v3 3.7GHz CPU (4 cores), 32GB RAM.
Virtualization Technologies

• “Standard” VM
  – Standard Debian-based Linux VM

• “Tinyfied” VM
  – Tinyx, based on Linux kernel/busybox

• Unikernel
  – On Xen: MiniOS + miniperf
  – On KVM: OSv + iperf

• Containers
  – Docker
Virtualization Technologies

• “Standard” VM
  – Standard Debian-based Linux VM

• “Tinyfied” VM
  – Tinyx, based on Linux kernel/busybox

• Unikernel
  – On Xen: MiniOS + miniperf
  – On KVM: OSv + iperf

• Containers
  – Docker
Standard VM: Application on Top of Distro

- User Application
- 3rd Party Applications
- Libraries
- Services
- Kernel
Most of the VM not Used...

User Application

3rd Party Applications

Libraries

Services

Kernel
Tinyx: Keep Only What’s Needed

Nginx

User Application

3rd Party Applications

Libraries

Services

Kernel

memcached
bash

libssl
libc

ssh

init

ext4
netfront
blkfront
Tinyx: Taylor-made Distro

Nginx

- memcached
- bash
- libssl
- libc
- ssh
- init
- netfront
- blkfront
- ext4

User Application

3rd Party Applications

Libraries

Services

Kernel
Tinyx: Taylor-made Distro

- Nginx
  - memcached
  - bash
  - libssl
  - libc
- ssh
- init
- netfront
- blkfront
- ext4

```
# ps aux

PID USER   TIME  COMMAND
 1 root  0:02  init
 2 root  0:00  [kthreadd]
 3 root  0:00  [ksftirpqd/0]
 4 root  0:00  [kworker/0:0]
 5 root  0:00  [kworker/0:0H]
 6 root  0:00  [kworker/u2:0]
 7 root  0:00  [rcu_sched]
 8 root  0:00  [rcu_bh]
 9 root  0:00  [migration/0]
10 root  0:00  [watchdog/0]
11 root  0:00  [khelper]
12 root  0:00  [kdevtmpfs]
13 root  0:00  [xenwatch]
14 root  0:00  [xenbus]
15 root  0:00  [khungtaskd]
16 root  0:00  [writeback]
17 root  0:00  [crypto]
18 root  0:00  [bioset]
19 root  0:00  [kblockd]
20 root  0:00  [edac-poller]
21 root  0:00  [kworker/0:1]
22 root  0:00  [kswapd0]
23 root  0:00  [fsnotify_mark]
24 root  0:00  [khvcd]
25 root  0:00  [ipv6_addrconf]
26 root  0:00  [deferwq]
27 root  0:00  [kworker/u2:1]
33 root  0:00  nginx: master process /usr/sbin/nginx
34 www-data 0:00  nginx: worker process
35 www-data 0:00  nginx: worker process
36 www-data 0:00  nginx: worker process
37 www-data 0:00  nginx: worker process
38 www-data 0:00  nginx: worker process
43 root  0:00  /usr/sbin/dropbear -R
44 root  0:00  -sh
57 root  0:00  ps aux
```
Tinyx: Taylor-made Distro

Nginx
- memcached
- bash
- libssl
- libc
- ssh
- init

Netfront
- blkfront
- ext4

Kernel Services
- User
- Application
- 3rd Party Applications

Libraries
- memcached
- bash
- 3rd Party Applications
- libc
- libssl
- ssh
- init

Distro
- Nginx
- Kernel
- Services
- Libraries
- User
- Application
- 3rd Party Applications

Example Command:
```
# ps aux
```

Output:
```
  PID USER   TTY      TIME  COMMAND
  1 root  0:02  ?        0:00  init
  2 root  0:00  ?        0:00  [kthreadd]
  3 root  0:00  ?        0:00  [ksoftirqd/0]
  4 root  0:00  ?        0:00  [kworker/0:0]
  5 root  0:00  ?        0:00  [kworker/0:0H]
  6 root  0:00  ?        0:00  [kworker/u2:0]
  7 root  0:00  ?        0:00  [rcu_sched]
  8 root  0:00  ?        0:00  [rcu_bh]
  9 root  0:00  ?        0:00  [migration/0]
 10 root  0:00  ?        0:00  [watchdog/0]
 11 root  0:00  ?        0:00  [khelper]
 12 root  0:00  ?        0:00  [kdevtmpfs]
 13 root  0:00  ?        0:00  [xenwatch]
 14 root  0:00  ?        0:00  [xenbus]
 15 root  0:00  ?        0:00  [khungtaskd]
 16 root  0:00  ?        0:00  [writeback]
 17 root  0:00  ?        0:00  [crypto]
 18 root  0:00  ?        0:00  [bioset]
 19 root  0:00  ?        0:00  [kblockd]
 20 root  0:00  ?        0:00  [edac-poller]
 21 root  0:00  ?        0:00  [kworker/0:1]
 22 root  0:00  ?        0:00  [kswapd0]
 23 root  0:00  ?        0:00  [fsnotify_mark]
 24 root  0:00  ?        0:00  [khvcd]
 25 root  0:00  ?        0:00  [ipv6_addrconf]
 26 root  0:00  ?        0:00  [deferwil]
 27 root  0:00  ?        0:00  [kworker/u2:1]
 43 root  0:00  ?        0:00  nginx: master process /usr/sbin/nginx
 45 www-data 0:00  ?        0:00  nginx: worker process
 46 www-data 0:00  ?        0:00  nginx: worker process
 47 www-data 0:00  ?        0:00  nginx: worker process
 48 www-data 0:00  ?        0:00  nginx: worker process
 53 root  0:00  ?        0:00  /usr/sbin/dropbear -R
 56 root  0:00  ?        0:00  -sh
 57 root  0:00  ?        0:00  ps aux
```
Tinyx: Taylor-made Distro

Nginx

- memcached
- bash

Libraries

- libssl
- libc

3rd Party Applications

- ssh
- init

NetFront

- blkfront
- ext4

Kernel Services

- init
- ext4
- netfront

User Applications

- memcached
- bash
Tinyx: Taylor-made Distro

- Keep only the necessary bits and pieces
  - Specialized kernel build containing only the necessary modules
  - Root filesystem populated with only necessary services, libraries and 3rd party applications
Virtualization Technologies

• “Standard” VM
  – Standard Debian-based Linux VM
• “Tinyfied” VM
  – Tinyx, based on Linux kernel/busybox
• Unikernel
  – On Xen: MiniOS + miniperf
  – On KVM: OSv + iperf
• Containers
  – Docker
Virtualization Technologies

• “Standard” VM
  – Standard Debian-based Linux VM

• “Tinyfied” VM
  – Tinyx, based on Linux kernel/busybox

• Unikernel
  – On Xen: MiniOS + miniperf
  – On KVM: OSv + iperf

• Containers
  – Docker
What’s a Unikernel?

• Specialized VM: single application + minimalistic OS

• Single address space, co-operative scheduler so low overheads
What’s a Unikernel?

• Specialized VM: single application + minimalistic OS

• Single address space, co-operative scheduler so *low* overheads
What’s a Unikernel?

• Specialized VM: single application + minimalistic OS

• Single address space, co-operative scheduler so low overheads
Unikernels for Benchmarking

On Xen

- guest OS
- apps
- Xen
Unikernels for Benchmarking
Unikernels for Benchmarking

On Xen

- apps
- guest OS: Xen

On KVM

- apps
- guest OS: KVM

Iperf

mini OS: Xen
Unikernels for Benchmarking

On Xen

apps

guest OS

Xen

iperf

mini OS

Xen

On KVM

apps

guest OS

KVM

iperf

OSv

KVM
Nota Bene...

• Our unikernel numbers include optimizations to the underlying virtualization platforms (Xen, KVM)
  – Toolstacks
  – Back-end stores
  – Hotplug scripts
  – Network drivers (on Xen Tx)

• No time to go over these...
Image Size, Memory Usage (log scale)
RTT

<table>
<thead>
<tr>
<th>System</th>
<th>RTT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>standardvm.xen</td>
<td>34</td>
</tr>
<tr>
<td>tinyx.kvm</td>
<td>19</td>
</tr>
<tr>
<td>standardvm.kvm</td>
<td>18</td>
</tr>
<tr>
<td>tinyx.xen</td>
<td>15</td>
</tr>
<tr>
<td>unikernel.osv.kvm</td>
<td>9</td>
</tr>
<tr>
<td>unikernel.minios.xen</td>
<td>5</td>
</tr>
<tr>
<td>container</td>
<td>4</td>
</tr>
</tbody>
</table>
Conclusions

• Common lore: VMs provide good isolation but are heavyweight
  – Results with standard VMs confirm this

• Containers provide lighter-weight virtualization
  – But tinyfied VMs and especially unikernels yield comparable performance
Conclusions

• Common lore: VMs provide good isolation but are heavyweight
  – Results with standard VMs confirm this

• Containers provide lighter-weight virtualization
  – But tinyfied VMs and especially unikernels yield comparable performance
Potential Contributions to draft-natarajan-nfvrg-containers-for-nfv-01

2.1.1 Challenges
- VNF provisioning time
- Runtime performance (throughput, scaling up/down)

3. Benefits of Containers
- Service agility vs VMs
- Containers have better runtime performance
- Auto-scaling of VNFs
- Cross-VNF compatibility: container unikernel/minimalistic distro
- Overall performance: VMs -25% throughput vs containers

5. Conclusion
- Containers have significant advantages vs hypervisor-based solutions