Reducing Kernel Surface Areas for Isolation and Scalability

Daniel Zahka
Brian Kocoloski

Kate Keahey

Washington University in St. Louis
James McKelvey School of Engineering

Argonne National Laboratory

48th International Conference on Parallel Processing, August 2019
OVERVIEW

1. There is a growing role for cloud computing in executing HPC workloads

2. In cloud environments, virtual machines provide better isolation than containers, and this can lead to significant performance advantages
CLOUD’S BENEFITS TO HPC

• Rapid adoption of new hardware: as cutting edge as all but the newest of HPC machines
CLOUD’S BENEFITS TO HPC

• Rapid adoption of new hardware: as cutting edge as all but the newest of HPC machines

Innovate with high-performance, on-demand hardware and software

Accelerate insights with the power of the Google Cloud compute, networking and storage infrastructure. Build your own supercomputer using the latest Intel processors, NVIDIA GPUs, and Cloud TPUs with high-throughput, low-latency object and file storage.
CLOUD’S BENEFITS TO HPC

• Rapid adoption of new hardware: as cutting edge as all but the newest of HPC machines

True HPC capabilities in the cloud, on demand

The performance and scalability of a world-class supercomputing center is now available to everyone, on demand in the cloud. Run your Windows and Linux HPC applications using high performance A8 and A9 compute instances on Azure, and take advantage of a backend network with MPI latency under 3 microseconds and non-blocking 32 Gbps throughput. This backend network includes remote direct memory access (RDMA) technology on Windows and Linux that enables parallel applications to scale to thousands of cores. Azure provides you with high memory and HPC-class CPUs to help you get results fast. Scale up and down based upon what you need and pay only for what you use to reduce costs.

Learn more about high performance A8 and A9 compute instances

Read about the performance improvements ANEO achieved with Azure
ADDITIONAL BENEFITS

• On-demand service models – e.g., auto-scaling and pay-as-you-go
  • Appropriate for dynamic workloads, which we’re seeing more of in HPC
• Virtually infinite on-demand resources

• Some HPC sites really need huge amounts of on-demand resources:
  • HEPCloud (Fermilab)
  • Advanced Photon Source (Argonne Natl. Lab)
  • Common issue: workload bursts and timeliness requirements
  • Batch scheduled HPC systems are not a good fit for this
Evolving the Computing Facility: Elasticity

- Usage is not steady-state
- Computing schedules driven by real-world considerations (detector, accelerator, …) but also ingenuity – this is research and development of cutting-edge science
WHAT DO WE NEED / WANT?

- Flexibility:
  - (nearly) every beamline instrument, and every science application is different, but can’t reinvent the wheel every time
  - => a modular data analysis pipeline that an BL or experience domain scientist can assemble?

- Ease of use: Computer literacy varies significantly, as does available institutional support (light sources are user facilities).
  - Automation !!!
  - Can beamline and domain scientists contribute code effectively?

- Visualization: how do you show highly multimodal, multiscale data?

- Scalability: depending on specific problem, may need to scale up to super computers, eg, to follow in real time processes.

- Accessibility, reliability, and on demand:
  - Most users probably cannot analyse their data at ‘home’. Need analysis tools somewhere (‘in the cloud’ ?), that can access the data somewhere else, and process

- But, Lightsources are ~24/7 facilities. Users get experimental time for 8-96 hours, typically once or twice a year. You need to be able to look at the data in order to determine next steps, ie, need (at least preliminary) analysis done within a few minutes. CanNOT (?) rely on an outside entity to keep beamlines running.

- Need HPC resources ‘on demand’, but can estimate when resources will be needed.
SUMMARY: MOVING TO CLOUDS?

• While hardware support is generally there, that’s only part of the problem
  • Cost
  • Licensing
  • Performance variation + multi tenancy

• Multi-tenancy is challenging for HPC applications to deal with

• The focus of this talk: if multi-tenancy comes to HPC, what type of system software infrastructure is best?
CONTAINERS OR VMS?

• HPC
  • Bare-metal Linux, probably with Linux containers

• Commodity Cloud
  • Container ecosystems …
  • … but also VMs
    • VMs running Linux
    • VMs running micro-kernels
    • VMs running functions (e.g., FaaS or “serverless” computing)

• The cloud still relies on VMs to provide isolation, primarily for security

• Current trends in HPC system software are actually towards solutions with strictly less isolation
  • Evidenced by the success of “bare-metal” containers
    • e.g., Singularity, Shifter
  • Focus is on proximity to hardware rather than isolation from multi-tenancy
ISOLATION FOR MULTI-TENANCY

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Performance</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Software</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ISOLATION FOR MULTI-TENANCY

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Performance</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ generally well studied</td>
<td>✓ accepted philosophy</td>
</tr>
<tr>
<td>System Software</td>
<td></td>
<td>Lots of industry buy-in with “lightweight virtualization”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-- Kata containers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-- AWS Firecracker</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-- IBM’s Nabla</td>
</tr>
</tbody>
</table>
ISOLATION FOR MULTI-TENANCY

Hardware
✓ generally well studied

System Software
Where we are focusing

Performance

Security
✓ accepted philosophy

Lots of industry buy-in with “lightweight virtualization”
-- Kata containers
-- AWS Firecracker
-- IBM’s Nabla
KERNEL VARIABILITY ANALYSIS

• Goal: determine whether choice of virtualization/containerization impacts parallel application performance due to lack of isolation
  • How to measure this?
  • What is the layer of the stack where multi-tenant performance characteristics are much different?
**Methodology**

- Each thread issues system calls to the kernel
- Workload is not hardware intensive – it relies almost exclusively on software efficiency
  - Locks on data structures
  - Software caches (e.g. page cache, SLAB allocator)
EXPERIMENTAL SETUP

Configuration 1
Linux only

Linux kernel

Physical Cores
EXPERIMENTAL SETUP

Configuration 2
KVM virtualization

Physical Cores
**EXPERIMENTAL SETUP**

Configuration 3

Docker containerization

<table>
<thead>
<tr>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₆₃</th>
</tr>
</thead>
</table>

- Docker container
- Docker container
- Docker container
- Docker container

Linux kernel

Physical Cores
**System Call Performance**

<table>
<thead>
<tr>
<th></th>
<th>% of system calls with median below</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1µs</td>
</tr>
<tr>
<td>Linux</td>
<td>11.12</td>
</tr>
<tr>
<td>KVM</td>
<td>8.34</td>
</tr>
<tr>
<td>Docker</td>
<td>7.35</td>
</tr>
</tbody>
</table>

**Table 1.** Breakdown of median system call performance in Linux, KVM, and Docker

<table>
<thead>
<tr>
<th></th>
<th>% of system calls with 99th percentile below</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1µs</td>
</tr>
<tr>
<td>Linux</td>
<td>0.01</td>
</tr>
<tr>
<td>KVM</td>
<td>0.02</td>
</tr>
<tr>
<td>Docker</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 2.** Breakdown of 99th percentile system call performance in Linux, KVM, and Docker
**Lack of isolation -> Poor tail performance**

Tail performance in Linux/Docker is **really bad**

KVM tail performance limited to ~10 ms
KERNEL SURFACE AREA

• Basic hypothesis: performance variability is a function of how “large” the kernel is
  • How many cores its managing
  • How much memory its managing
  • How many applications/processes its managing

• Why would this be the case?
  • The Linux kernel is heavily “fast path” optimized
  • E.g.,
    • Fast-path memory allocation (slab cache)
    • Caching in front of the slab cache! (qmempool)
    • Caching of file-I/O (page cache)
  • Locking and synchronization in general

• Idea: replicated OS kernels via virtualization, each managing a smaller subset of resources and tenants
  • Allows us to trade-off space efficiency for isolation and consistency
KERNEL SURFACE AREA

• We characterized every Linux system call into a set of six categories:
  1. Process management & scheduling
  2. Memory management
  3. File I/O
  4. Filesystem management
  5. Inter-process communication
  6. Permissions/capabilities management

• Which are most positively influenced by reductions in kernel surface area?
KERNEL SURFACE AREA

(a) Process management/scheduling
(b) Memory management
(c) File I/O
(d) Filesystem operations
(e) IPC
(f) Permissions/capabilities
REDUCTION IN EXTREME OUTLIERS

(a) Process management/scheduling

(d) Filesystem operations
CONSISTENT LATENCY REDUCTION

(b) Memory management

(f) Permissions/capabilities
**Parallel Implications?**

- Low level performance shows benefit from smaller isolated kernels

- Does this matter for large scale parallel applications?

- We ran a collection of kernel-intensive parallel applications on 64 Chameleon nodes using virtual machines and containers
(a) Isolated performance
With multi-tenancy, virtualized environments outperform native environments due to better isolation
• xapian (search engine)
• img-dnn (image recognition)
• sphinx (speech recognition)
• moses (machine translation)
CONCLUSION

• System software isolation provides performance benefits due to performance isolation
• Future: hybrid containers within lightweight VMs may provide the best of both worlds
THANK YOU

BRIAN KOCOLOSKI

http://www.cse.wustl.edu/~brian.kocoloski
brian.kocoloski@wustl.edu