ECE 574 – Cluster Computing
Lecture 21

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

11am Barrows 133

9 April 2024

Announcements

- HW#9 was posted (OpenCL)
- Don't forget midterm exam on Tuesday, 16 April
- Don't forget HW#8 was Due
- Project status due 12th (see next slide)

Project Status Update due Friday 12th

- e-mail, one per group
- One sentence summary
- Any hardware/software waiting on
- Related work: 2 if alone, 4 if group
  See pdf for full details Ideally academic
  Don't pay for ACM/IEEE papers, access through UMaine library
- Date you'd like to present 18th/23rd/25th

Can You Use Clusters for non-parallel jobs?

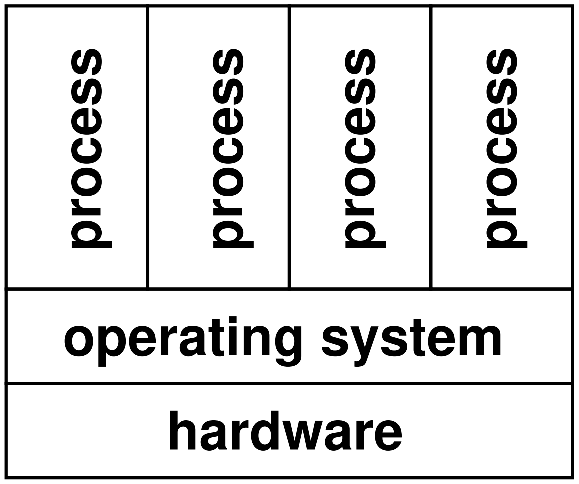
- Yes
- You might just have a lot of copies of single jobs you want to run
- You can use a cluster for this, including things like slurm
- Just don't use MPI
- Story for clusters at Cornell used for comp arch simulations
  Not worth anyone's trouble to parallelize them

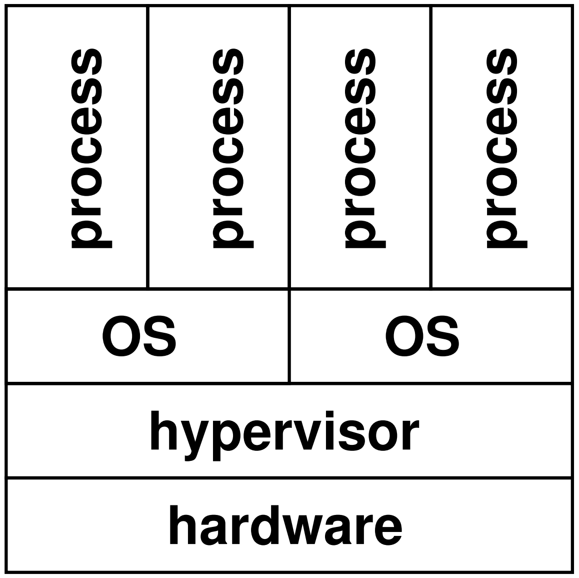
Cloud Computing

- Grid Computing?
- In some ways similar to HPC clusters, others not
- High-performance networking and MPI not as important
- Often things like I/O for storage more important
- Running things in shared environment
- Often virtualized setups

Virtualization

- Lets you run multiple operating system images, giving each the illusion that they are running on distinct hardware.
- The OSes are context-switched between, much as processes are context-switched under an OS
- When running inside a fully virtualized system, code should not be able to tell it’s not running on bare metal
- The OSes are isolated and one crashing should not affect any of others.


process
process
process
process
operating system
hardware


process
process
process
process
OS
OS
hypervisor
hardware

Why virtualize?

- **Server consolidation** – if you have multiple servers, each using 10% of CPU, can put them on one big server
- **Security** – can give each critical task its own full OS instance, so if something goes wrong it won't affect the others (this is harder to do with processes on an OS)
- **Multiple OSes** – can run multiple OSes (Windows, Linux, Etc) on same machine at same time
- **Ease of deployment** – can make OS snapshots/images and can quickly bring up and down on other machine

Downsides of virtualization?

- Like any layer of abstraction: Overhead
- Performance slowdown
- Hardware failure take down multiple OSes
- Security: VM escape
  information leakage/side channel attacks

Terms

- Guest – the operating system running inside a virtual system
- Host – the operating system running on bare metal (may be a hypervisor instead)
- VM (virtual machine) not virtual memory – the software/hardware that provides the virtual hardware interface
- Hypervisor – the software that controls bringing up and controlling the guest operating systems

Are you ever running on real hardware?

- Some modern machines all you ever get to run on is the VM
- VM (some power machines, ps3, never run on raw hardware)
- Nested VM
- x86 SMM mode (system maintenance mode)

Full Simulation

- Emulate the entire CPU and all hardware in software
- Full system simulators, such as Qemu
- What's the downside of this? (slow, slow, slow)

Full Virtualization

- "Virtualize the CPU"
  Run instructions as normal, but anything that gives away it is virtual must trap to the hypervisor.
- Trap on access to hardware and simulate (with Qemu or similar)
- KVM
- VMware

Linux KVM

- Kernel-based virtual machine
- Hardware-assisted virtualization
- Requires CPU with hardware virtualization extensions
- Linux Kernel acts as hypervisor
- Provides /dev/kvm
  - Sets up address space
  - Provide boot firmware
  - Display hardware

Setting up KVM

- Not horrible, but haven't done it for a while
- Need to create disk image
- Getting network going (bridging to outside world) can be challenge
- Tools to automate some of this stuff

Popek and Goldberg virtualization requirements

Formal requirements for virtualizable third generation architectures, Communications of the ACM, 1974.

- equivalence (fidelity): a program running under a VM should behave identical to running on bare metal monitor (VMM) should
- resource control (safety): VM must control all resources
- efficiency (performance): most instructions must execute without intervention

Hardware Virtualization Extensions (CPU)

- IBM System/370 in 1972
- x86 chips by default were not, leak too much info.
- Intel VT-x and AMD-V

x86 virtualization

A Comparison of Software and Hardware Techniques for x86 Virtualization by Adams and Agesen, ASPLOS 2006. VMware managed full virt on 32-bit x86 using dynamic binary instrumentation and segmentation.

- De-privilege: any attempt to read privileged info traps and can be intercepted
- Shadow structures: need copies of things that can't be intercepted at CPU level, like page-tables. Need to trap on access to these. True vs hidden page faults.

- x86 issues (assume protected mode)
  - visible privileged state (see privilege mode when read CS register; CPL (privilege level) lower 2 bits)
  - Lack of traps when privileged instructions run at user-level.
  - popf (pop flags) changes both ALU and system flags (IF, enable interrupts). When run non-privileged ignores this, doesn't trap.
- Intel VT-x and AMD-V
  - 2006
  - Adds virtual machine code block

- ◦ Intel: extended page tables (nested page tables)
- ◦ VMCS shadowing: allow nested VMs

Paravirtualization

- Hypervisor creates a special API that the guest OS uses (operating system must be modified)
- Can be faster (talk directly to hypervisor, no need to emulate hardware)
- Xen – uses stripped down Linux as hypervisor?
- Need specially compiled kernel that knows about hypervisor interfaces

Containers

- Look like you have own copy of OS, but just walled off more thoroughly than normal Unix process. More lightweight than VM
- One of more processes isolated from rest of system
- Self contained, all code needed to run included
  in theory isolated from changes in underlying OS/distribution
- Sort of halfway between running regular process vs running in VM

Linux Containers

- LXD/LXC/LXCFS – Linux container infrastructure
  `linuxcontainers.org`
- LXC – Linux Containers
  - Idea of containers dates back to FreeBSD jails in 1990s
    More of a security idea at the time
  - Linux support implemented out of cgroups (control groups) and systemd
  - Might have some manner of filesystem overlay

Docker

- Software can be installed on Linux to allow running containers
- Lightweight virtualization, runs on top of normal Linux but uses containers to isolate from other instances
- Uses cgroups, namespaces, union filesystems
- Unlike full virtualization, does not require another copy of the OS
- Also a way of packaging
- Written in go

- Difference from virtualization?
  - Doesn't need full disk image (large)
  - Doesn't need large reserved memory range
  - Diagram

    (Host-Hypervisor)-(GuestOS/Libs/App)

    (Host-Docker)-(Libs/apps)

Docker Swarm

- Easily create clusters
- This more like the old definition of clusters, rather than HPC
- More like an automatic failover type situation
- Load balancing

Kubernetes

- How to pronounce? koo-ber-neh-teez (k8s) Word is Greek for captain
- Originally from google? Lighter version of project borg?
- Pods full of containers that can communicate locally, to communicate remotely through an IP?
- Pods work together, use DNS and can share load
- Can run on top of Docker (but doesn't have to)
- Also written in go
- Master node, worker nodes

Kubernetes vs Docker

- Container Orchestration
- `https://containerjournal.com/2019/01/14/kubernetes-vs-docker-a-primer/`

HPC and Containers

- Singularity – openmpi/mpi in containers
- udocker
- socker
- pcocc
- Easier for sysadmin, they don't have to go through lots of effort to get versions/dependencies installed
- Easier for user, don't have to do complex installations from code from scratch and bug sysadmin
- Downsides?

- Swarm – not secure, too simple for HPC workloads
- K8s – hard to setup, poor scheduler / resource management
- Can't use TORQUE or slurm to schedule?
- shifter project to let docker run from slurm

HPC Package Management

- HPC clusters are often limited environments with conservative sysadmins
- What if you need software not installed?
- Virtualization or Containers might not be available
- HPC package managers?

Related: Flatpak vs Snap

- Traditional Linux distributions have package managers to install software for system-wide use
- Flatpak/Snap – more like an app-store model
  - Self contained code that comes with all libraries needed
  - Don't (usually) have to worry about underlying OS changes breaking things with your app
  - Downsides: extra disk space (duplicated files) security: instead of security fixes in one common library, need to update every package

Installing HPC Software

- HPC systems complex, users not always computer experts
- Can be really hard getting hastily written code to compile and run on new machine even if it's the same operating system
- For example, genomics people might standardize on ubuntu/perl/python with specific versions but other groups maybe centos with other versions

HPC Package Managers

- Easybuild – setup for downloading and building HPC software automatically, with reproducible builds
- GUIX HPC – per-user package management
- NIX – package manager and build system
- SPACK – HPC package manager for R, C, C++ and Fortran

Python in HPC

- C/C++/Fortran have somewhat steep learning curves
- Is it possible to use a higher level language?
- Python is popular
- Downside is Python can be 100x slower

Python Tradeoffs

- Pros
  - East to use/program
  - Lots of libraries/tools
  - External C libraries can provide fast code
- Cons
  - 100x slower
  - GIL (global interpreter lock) makes parallel code hard
  - Complex package dependencies on python version and libraries

◦ High disk usage – Often end up install multiple copies of large python libraries

Python related Tools

- Jupyter Notebooks – create documents containing code, data, visualizations
- Ipython (?)
- Spyder – Python IDE for scientific software

Python Package Managers

- Conda – python package manager (anaconda is the full distribution, miniconda minimal version)
- virtualenv

Ways to Improve Python Speed / External Libraries

- Numpy – fast matrix/array manipulation
- Scipy – more HPC code, DSP, image manipulation
- Pytorch – fast AI routines, including CUDA support

Ways to Improve Python Speed / JIT

- Compile interpreted python to machine code / executables
- Cython
- pypy – fast replacement for cython
- Numba – add jit note in code after profiling

Python, More traditional HPC

- Can profile python (snakeviz)
- Regular parallel programming limited by GIL global lock (they are working on removing it)
- MPI – can write MPI code using python

Performance Measurement in the Cloud

This is based on research I did at UTK back in the early 2010s

Traditional HPC

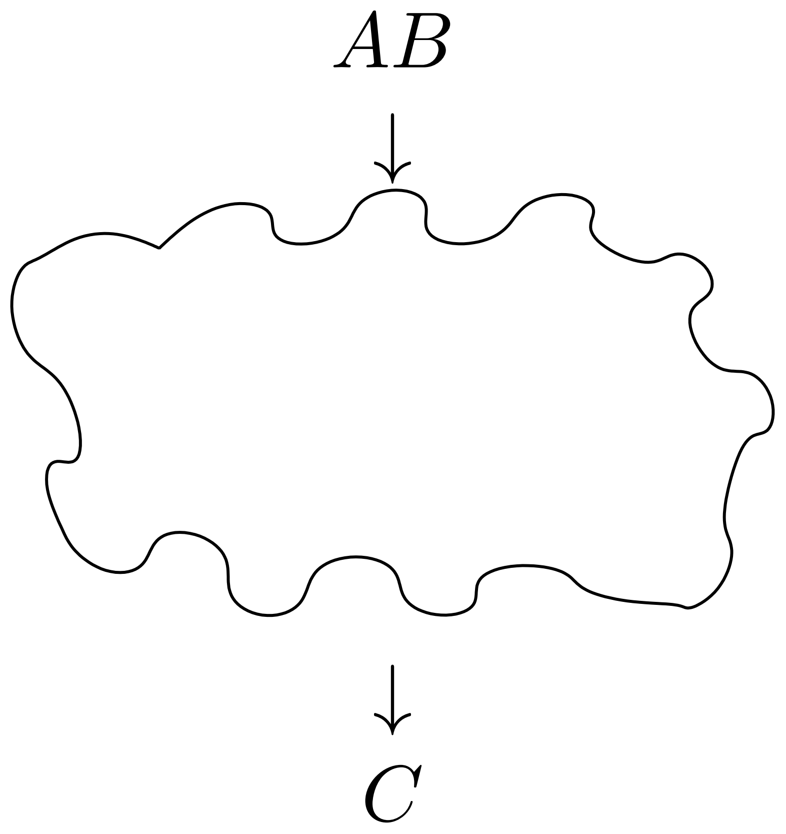
$$AB$$

$$\downarrow$$

$$\downarrow$$

$$C$$

Cloud-based HPC



Cloud Tradeoffs

Pros

- No AC bill
- No electricity bill
- No need to spend $\$\$\$$ on infrastructure

Cons

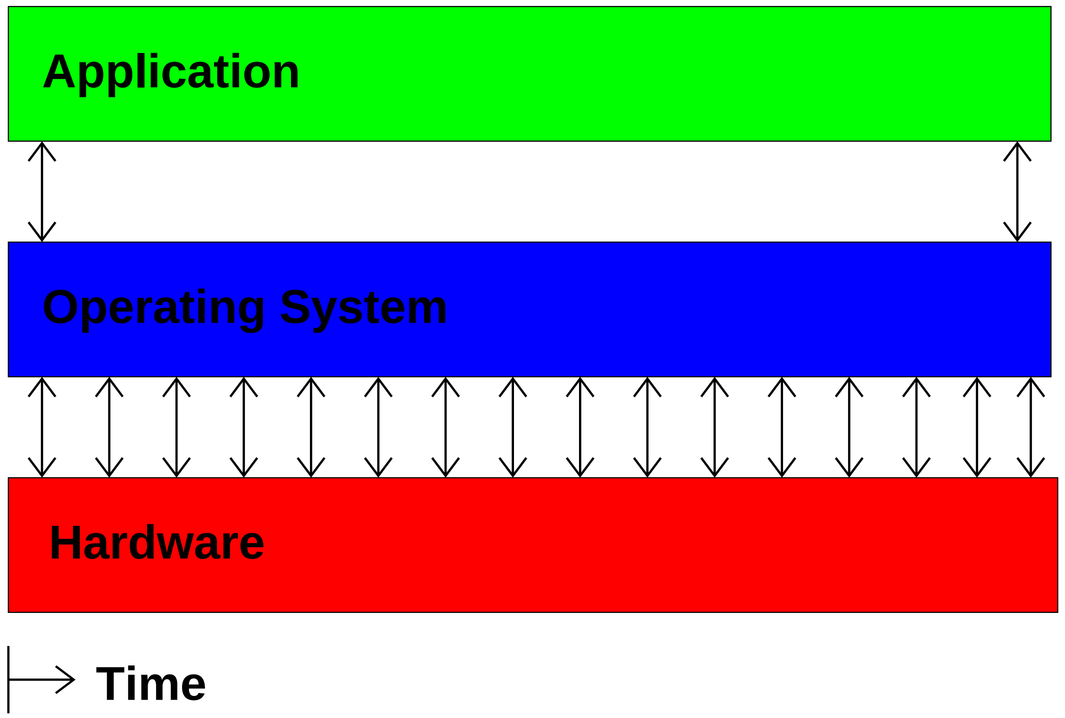
- Unexpected outages
- Data held hostage
- Infrastructure not designed for HPC

Measuring Performance in the Cloud

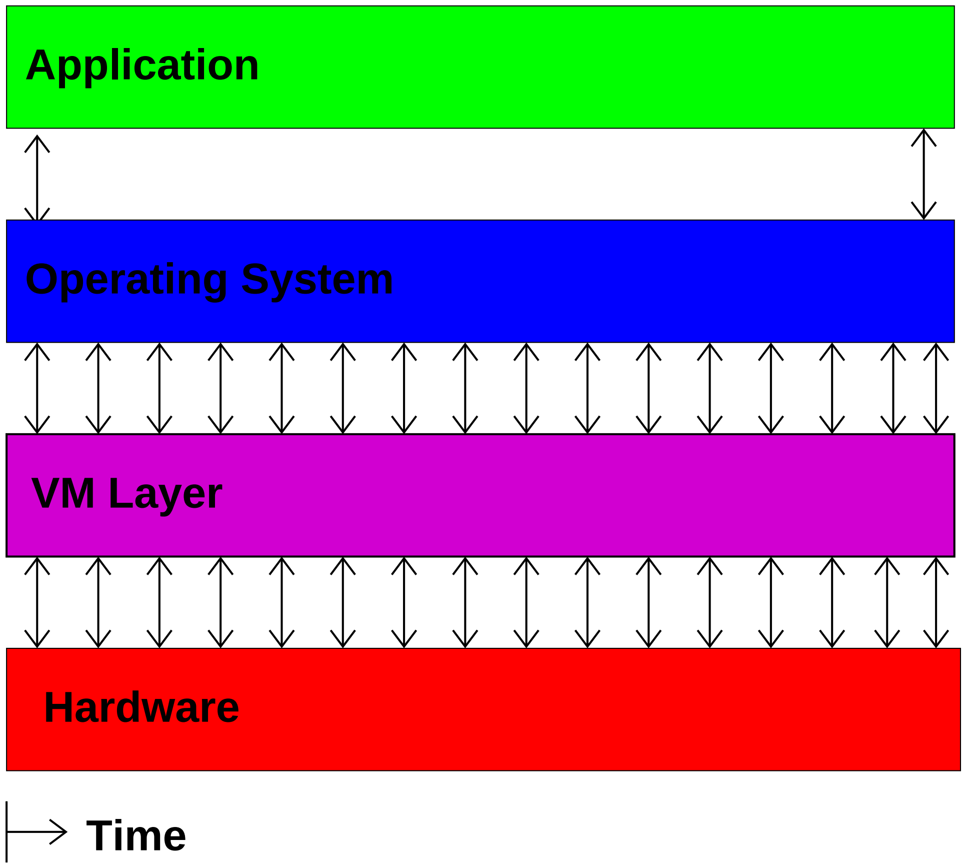
First let's just measure runtime

This is difficult because in virtualized environments

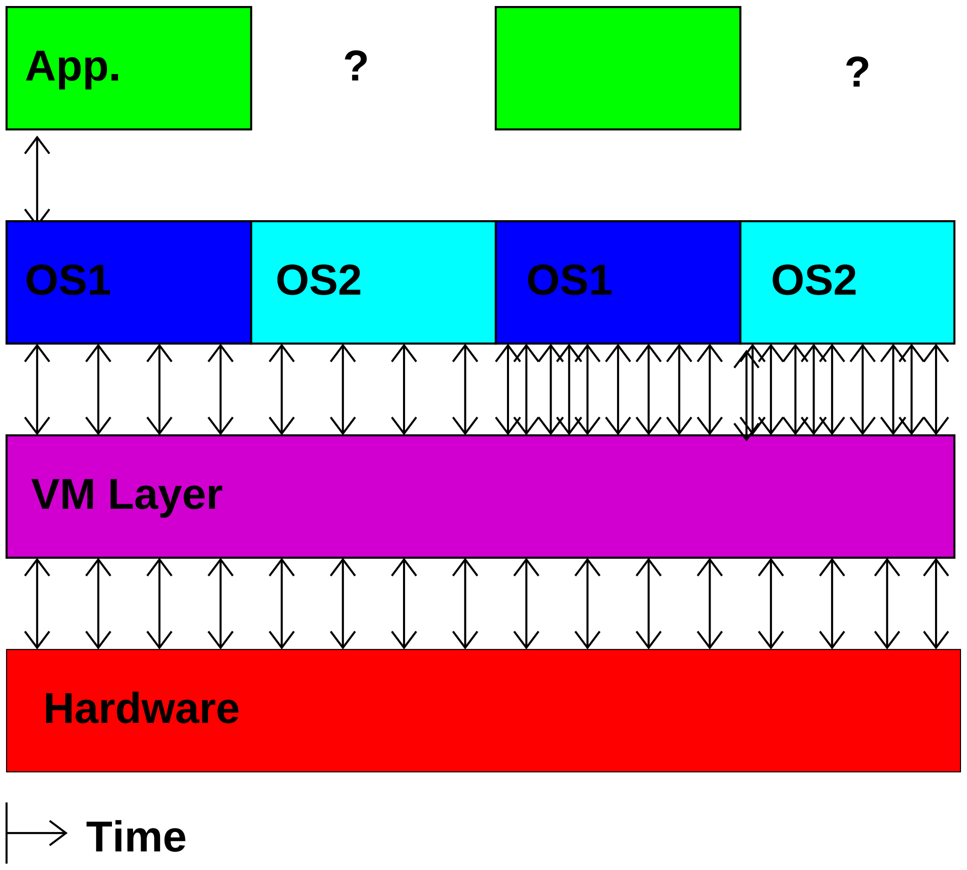
Simplified Model of Time Measurement



Then the VM gets involved



Then you have multiple VMs



So What Can We Do?

Hope we have exclusive access and measure wall-clock time.

Measuring Time Externally

- Ideally have local hardware access, root, and hooks into the VM system
- Otherwise, you can sit there with a watch
- Danciu et al. send UDP packet to remote server
- Most of these are not possible in a true “cloud” setup

Measuring Time From Within Guest

- Use `gettimeofday()` or `clock_gettime()`
- This might be the only interface we have
- How bad can it be?

Cloud Performance Measurement

With High Performance Computing moving to the cloud, virtualization-aware performance measurement tools are a necessity.

Performance API (PAPI)

- Widely-used, Cross-platform, Open-Source Performance Measurement Library
  - ⇒ Linux, AIX, FreeBSD, Solaris
  - ⇒ x86, Power, ARM, MIPS
  - ⇒ BlueGene P/Q, Cray
- Use directly or via high-level tools (TAU, Perfsuite, Vampir, Scalasca, HPCToolkit)

PAPI-V

Virtualization-aware PAPI, or “PAPI-V” extends PAPI to be useful in cloud environments.

- Report virtual system info
- Provide enhanced timing info
- Virtualization-related components
- Virtualized Counters

Virtual System Info

- Virtualization vendor obtained via CPUID, reported in `hw_info.virtual_vendor_string`
- Supported by KVM, Xen, VMware, etc.
- Info for user, helps with bug reports

The Timing Problem

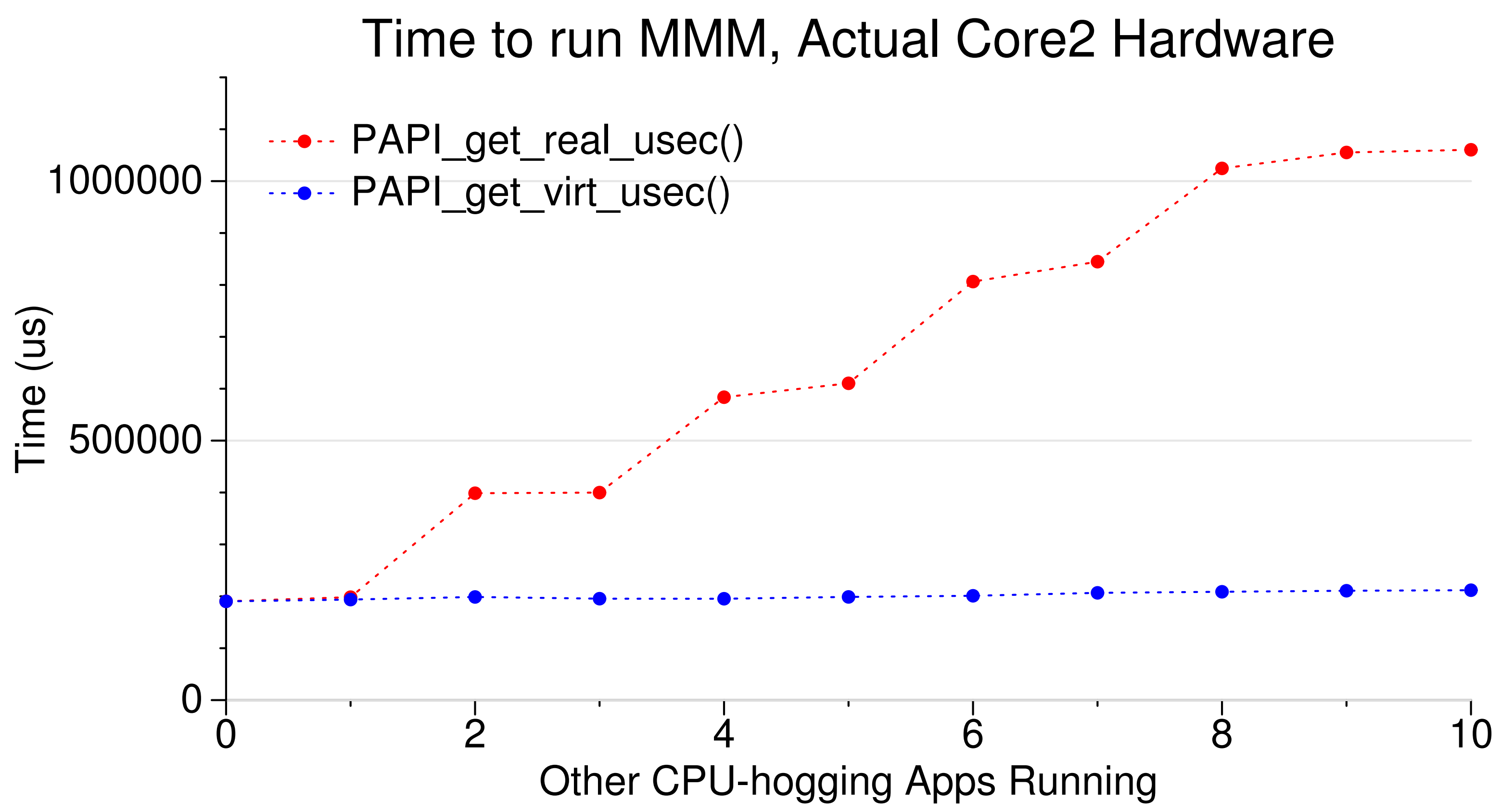
- Time is an important component of most performance measurements
- The concept of “time” gets fluid once virtualization is involved
- Ideally you want wallclock time; this is hard to get within a VM guest

PAPI Timing Interface

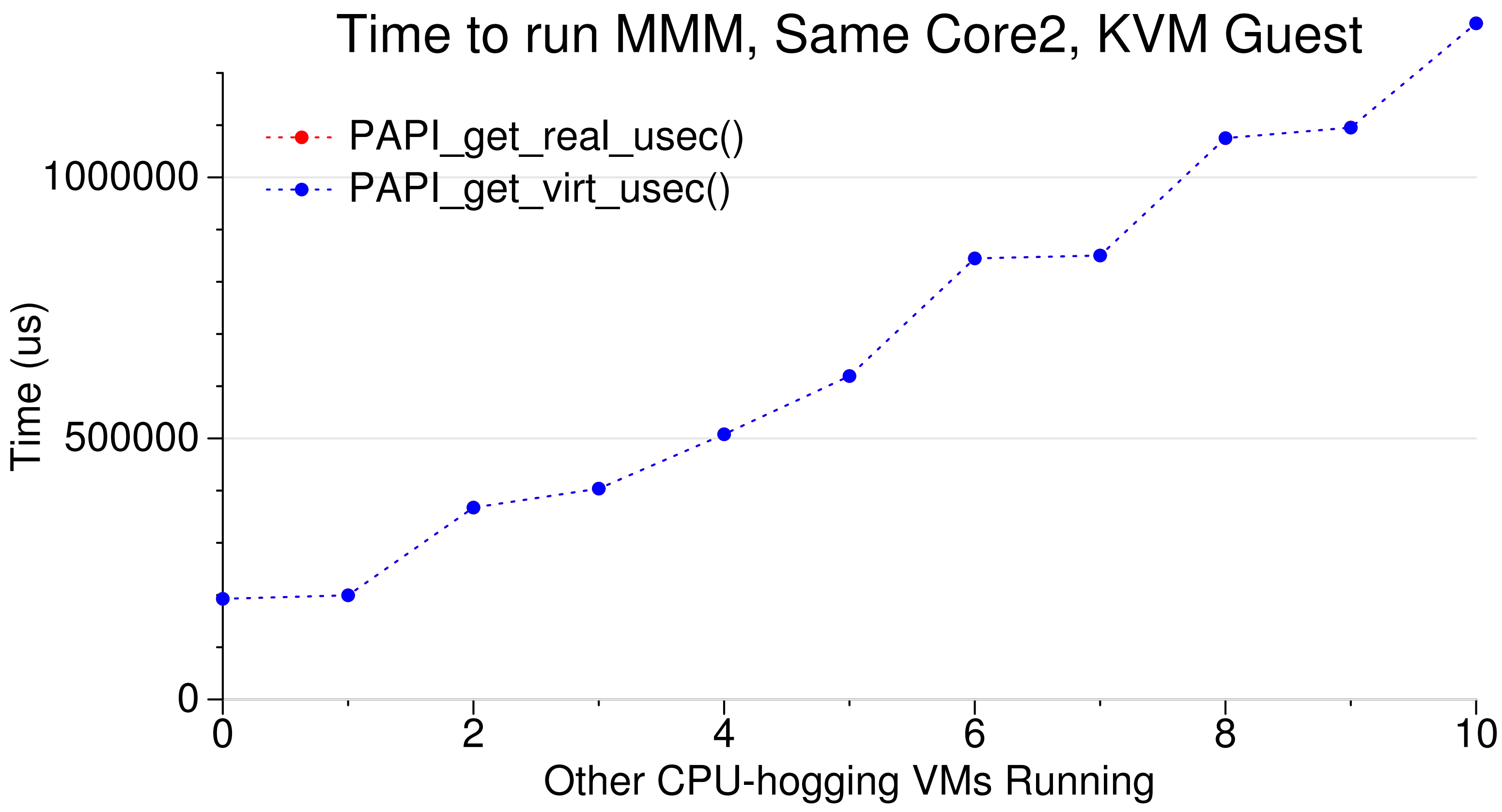
On Linux the timing functions use the POSIX timer interface

- `PAPI_get_real_usec();`
  ⇒`clock_gettime(CLOCK_REALTIME);`
- `PAPI_get_virtual_usec();`
  ⇒`clock_gettime(CLOCK_THREAD_CPUTIME_ID);`

Timing Behavior on Bare Metal



Timing Behavior on Virtualized System

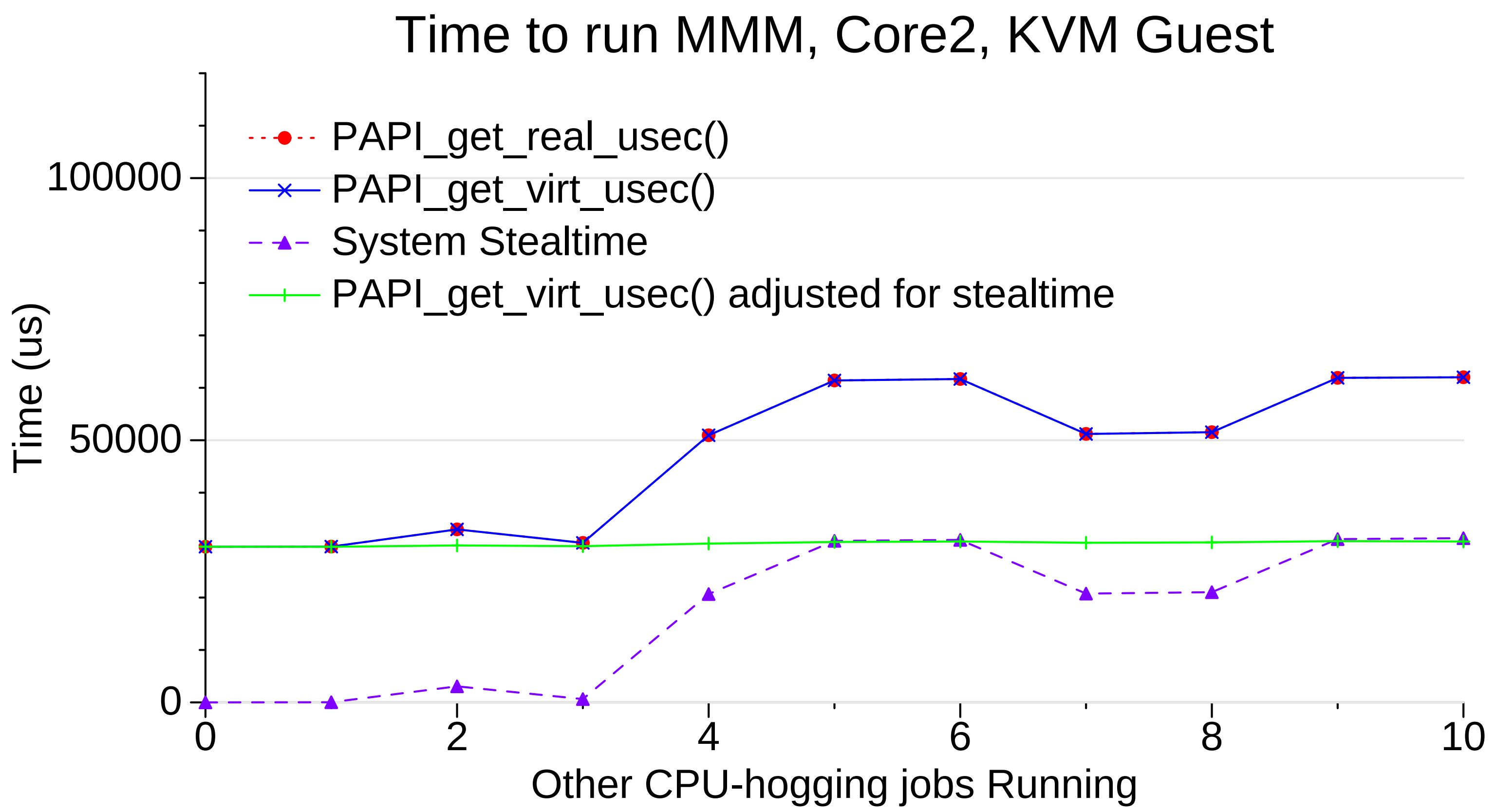


Stealtime

What is needed is a way for accounting for time the VM is scheduled out.

- Since 2.6.11 Linux can provide this *stealtime* information
- It is system wide, not per-process, which makes auto-adjusting PAPI timing measurements problematic
- PAPI 5.0 provides a stealtime component

Timing Adjusted with Stealtime

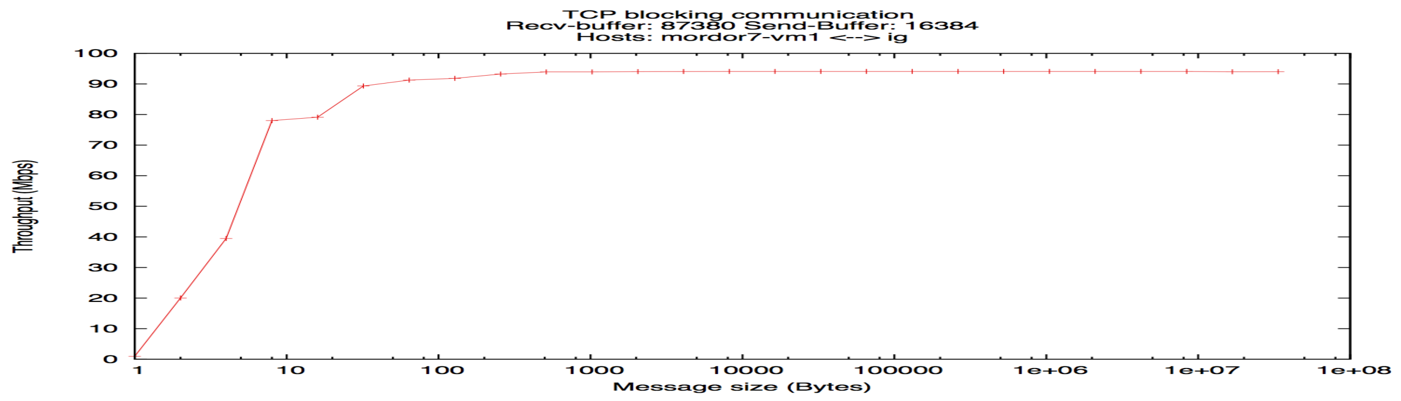


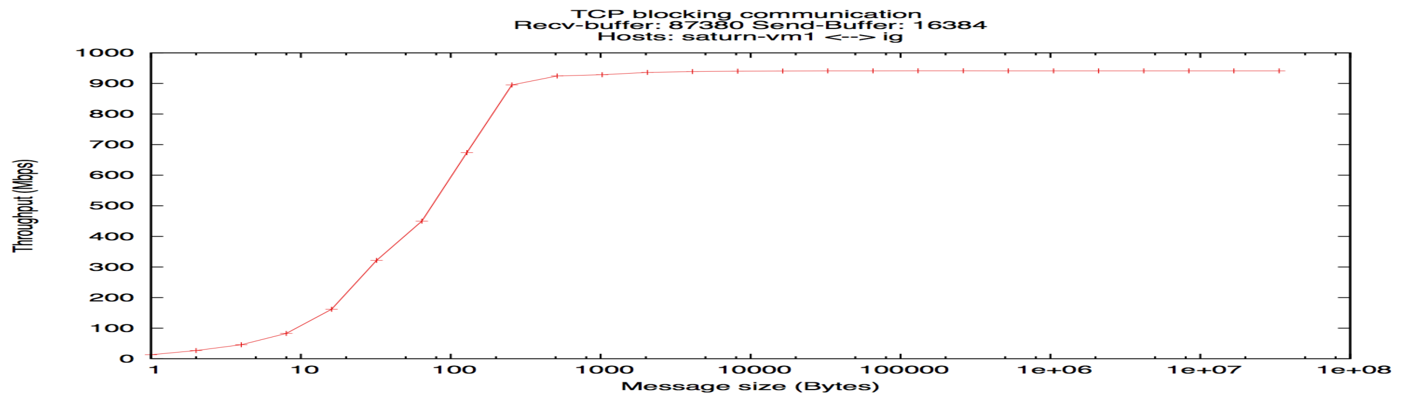
Network Components

PAPI also has components for measuring Network I/O.

- Generic network component
- Infiniband component
- Myrinet component

Infiniband DirectPath Comparison





VMware Component

PAPI supports a component that provides access to VMware-specific interfaces

- pseudo-performance counters – extra timing info via `rdpmc`
- VMware guest SDK (ESX only) – provides various other performance related measurements, including stealtime

Virtualized Performance Counters

The VM host can virtualize performance counter access by trapping access to the MSRs, and saving/restoring values when suspending/resuming VMs.

- KVM supports this as of Linux 3.2 with a sufficiently recent version of the QEMU/KVM tool (with some limitations)
- Xen supports this as of Linux 3.5
- VMware support is underway