Achieving Performance Isolation with Lightweight Co-Kernels

Jiannan Ouyang, Brian Kocoloski, John Lange

The Prognostic Lab @ University of Pittsburgh

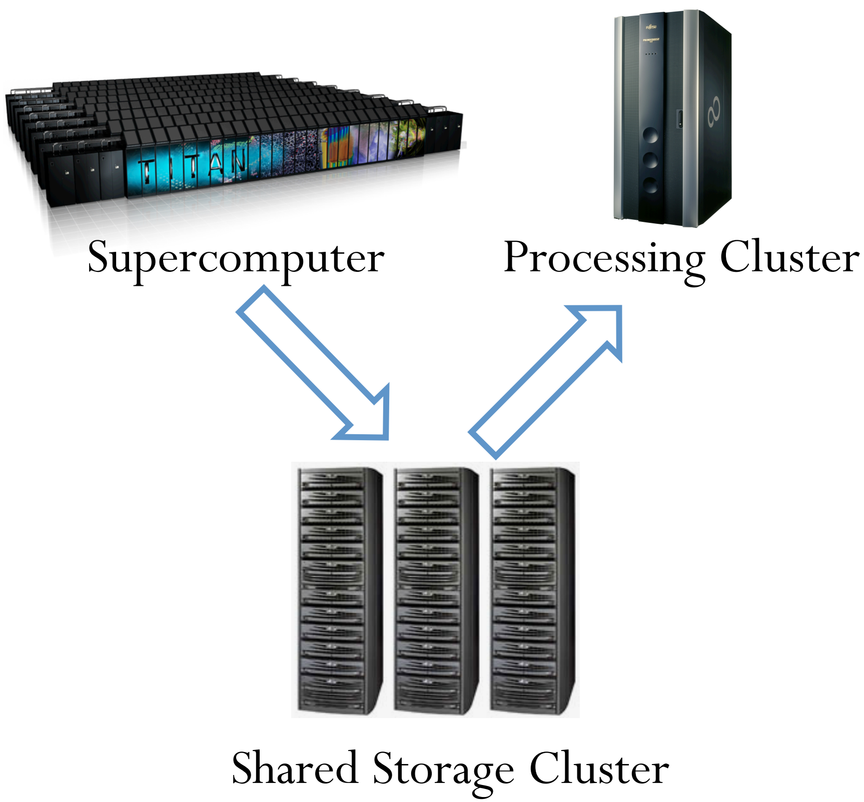
Kevin Pedretti

Sandia National Laboratories

HPDC 2015

HPC Architecture

Traditional



Problem: massive data movement over interconnects

In Situ Data Processing

Simulation	Analytic / Visualization
Operating System and Runtimes (OS/R)	
Compute Node	

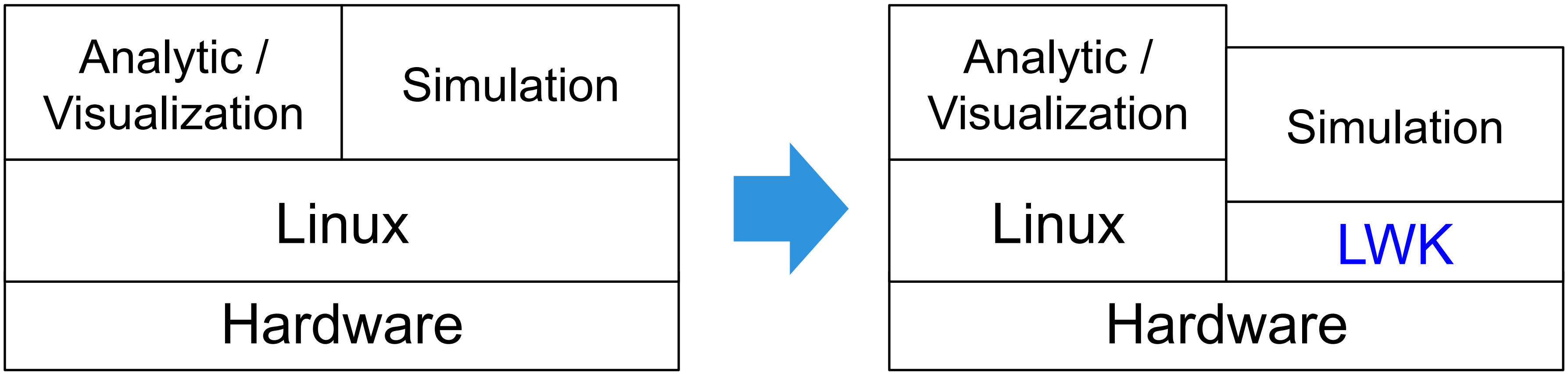
- Move computation to data
- Improved data locality
- Reduced power consumption
- Reduced network traffic

Challenge: Predictable High Performance

- **Tightly coupled HPC workloads are sensitive to OS noise and overhead** [Petrini SC'03, Ferreira SC'08, Hoefler SC'10]
  - Specialized kernels for predictable performance
    - Tailored from Linux: CNL for Cray supercomputers
    - Lightweight kernels (LWK) developed from scratch: IBM CNK, Kitten
- **Data processing workloads favor Linux environments**
- **Cross workload interference**
  - Shared hardware (CPU time, cache, memory bandwidth)
  - Shared system software

How to provide both Linux and specialized kernels on the same node, while ensuring performance isolation??

Approach: Lightweight Co-Kernels



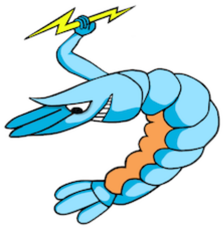
- Hardware resources on one node are dynamically composed into multiple partitions or enclaves
- Independent software stacks are deployed on each enclave
  - Optimized for certain applications and hardware
- Performance isolation at both the software and hardware level

Agenda

- Introduction
- **The Pisces Lightweight Co-Kernel Architecture**
- Implementation
- Evaluation
- Related Work
- Conclusion

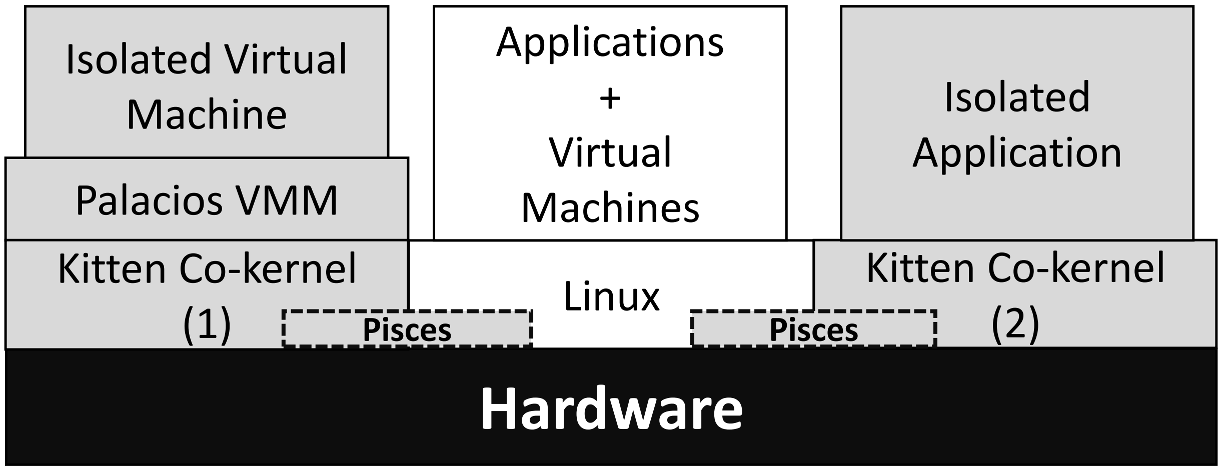
Building Blocks: Kitten and Palacios

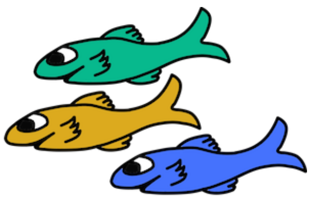
- **the Kitten Lightweight Kernel (LWK)**
  - Goal: provide predictable performance for massively parallel HPC applications
  - Simple resource management policies
  - Limited kernel I/O support + direct user-level network access
- **the Palacios Lightweight Virtual Machine Monitor (VMM)**
  - Goal: predictable performance
  - Lightweight resource management policies
  - Established history of providing virtualized environments for HPC [Lange et al. VEE '11, Kocoloski and Lange ROSS '12]



Kitten: https://software.sandia.gov/trac/kitten
Palacios: http://www.prognosticlab.org/palacios http://www.v3vee.org/

The Pisces Lightweight Co-Kernel Architecture





Pisces Design Goals

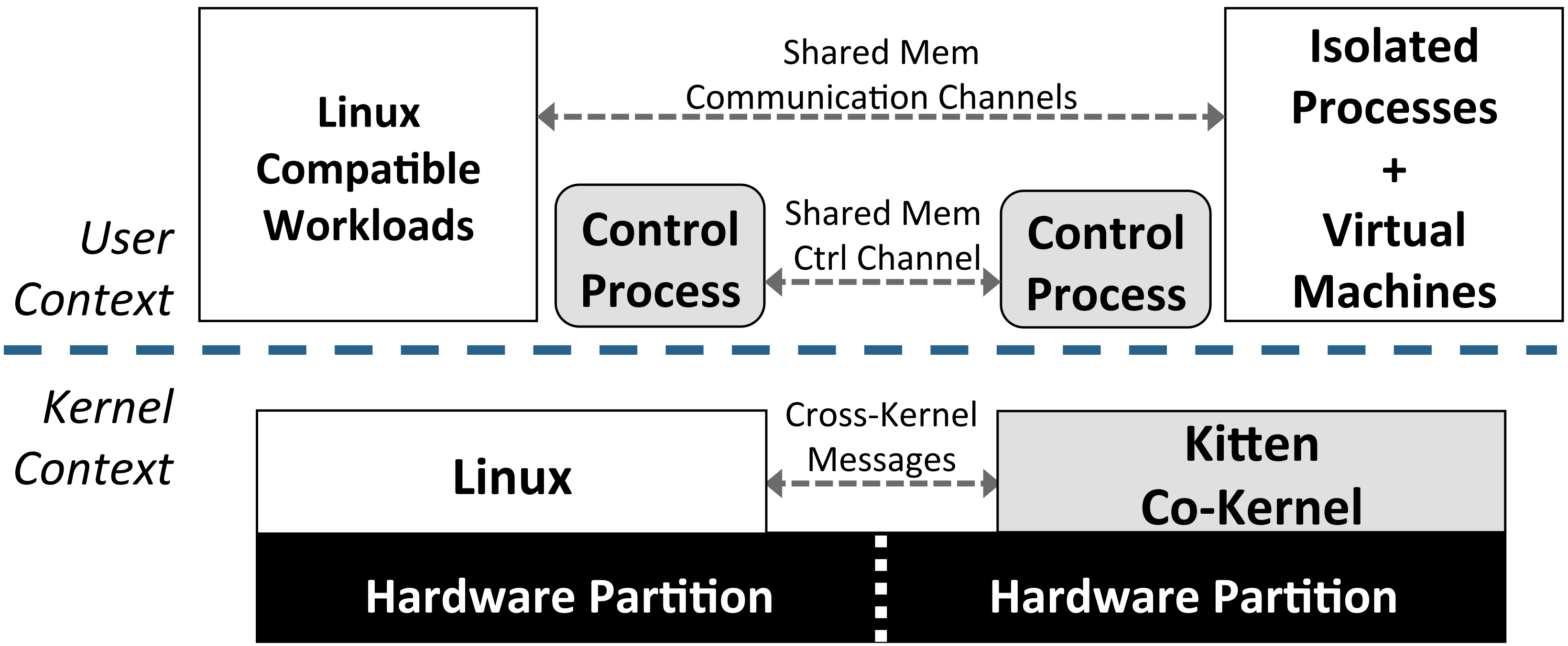
- Performance isolation at both software and hardware level
- Dynamic creation of resizable enclaves
- Isolated virtual environments

http://www.prognosticlab.org/pisces/

Design Decisions

- **Elimination of cross OS dependencies**
  - Each enclave must implement its own complete set of supported system calls
  - No system call forwarding is allowed
- **Internalized management of I/O**
  - Each enclave must provide its own I/O device drivers and manage its hardware resources directly
- **Userspace cross enclave communication**
  - Cross enclave communication is not a kernel provided feature
  - Explicitly setup cross enclave shared memory at runtime (XEMEM)
- **Using virtualization to provide missing OS features**

Cross Kernel Communication



XEMEM: Efficient Shared Memory for Composed Applications on Multi-OS/R Exascale Systems

[Kocoloski and Lange, HPDC '15]

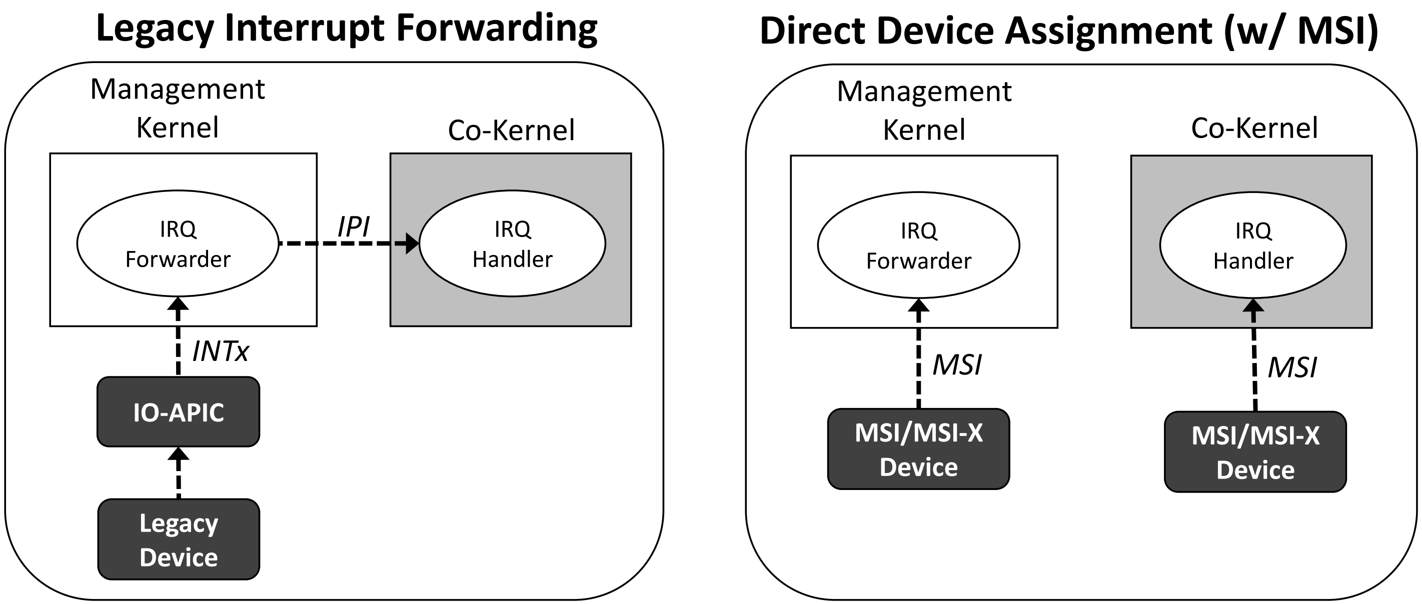
Agenda

- Introduction
- The Pisces Lightweight Co-Kernel Architecture
- **Implementation**
- Evaluation
- Related Work
- Conclusion

Challenges & Approaches

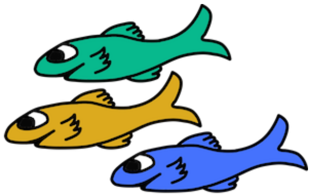
- **How to boot a co-kernel?**
  - Hot-remove resources from Linux, and load co-kernel
  - Reuse Linux boot code with modified target kernel address
  - Restrict the Kitten co-kernel to access assigned resources only
- **How to share hardware resources among kernels?**
  - Hot-remove from Linux + direct assignment and adjustment (e.g. CPU cores, memory blocks, PCI devices)
  - Managed by Linux and Pisces (e.g. IOMMU)
- **How to communicate with a co-kernel?**
  - Kernel level: IPI + shared memory, primarily for Pisces commands
  - Application level: XEMEM [Kocoloski HPDC'15]
- **How to route device interrupts?**

I/O Interrupt Routing



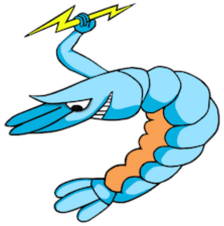
- Legacy interrupt vectors are potentially shared among multiple devices
  - Pisces provides IRQ forwarding service
  - IRQ forwarding is only used during initialization for PCI devices
- Modern PCI devices support dedicated interrupt vectors (MSI/MSI-X)
  - Directly route to the corresponding enclave

Implementation



- Pisces
  - Linux kernel module supports unmodified Linux kernels (2.6.3x – 3.x.y)
  - Co-kernel initialization and management

- Kitten (~9000 LOC changes)
  - Manage assigned hardware resources
  - Dynamic resource assignment
  - Kernel level communication channel



- Palacios (~5000 LOC changes)
  - Dynamic resource assignment
  - Command forwarding channel

Pisces: http://www.prognosticlab.org/pisces/
Kitten: https://software.sandia.gov/trac/kitten
Palacios: http://www.prognosticlab.org/palacios http://www.v3vee.org/

Agenda

- Introduction
- The Pisces Lightweight Co-Kernel Architecture
- Implementation
- **Evaluation**
- Related Work
- Conclusion

Evaluation

- 8 node Dell R450 cluster
  - Two six-core Intel "Ivy-Bridge" Xeon processors
  - 24GB RAM split across two NUMA domains
  - QDR Infiniband
  - CentOS 7, Linux kernel 3.16
- For performance isolation experiments, the hardware is partitioned by NUMA domains.
  - i.e. Linux on one NUMA domain, co-kernel on the other

Fast Pisces Management Operations

Operations	Latency (ms)
Booting a co-kernel	265.98
Adding a single CPU core	33.74
Adding a 128MB memory block	82.66
Adding an Ethernet NIC	118.98

Eliminating Cross Kernel Dependencies

	solitary workloads (us)	w/ other workloads (us)
Linux	3.05	3.48
co-kernel fwd	6.12	14.00
co-kernel	0.39	0.36

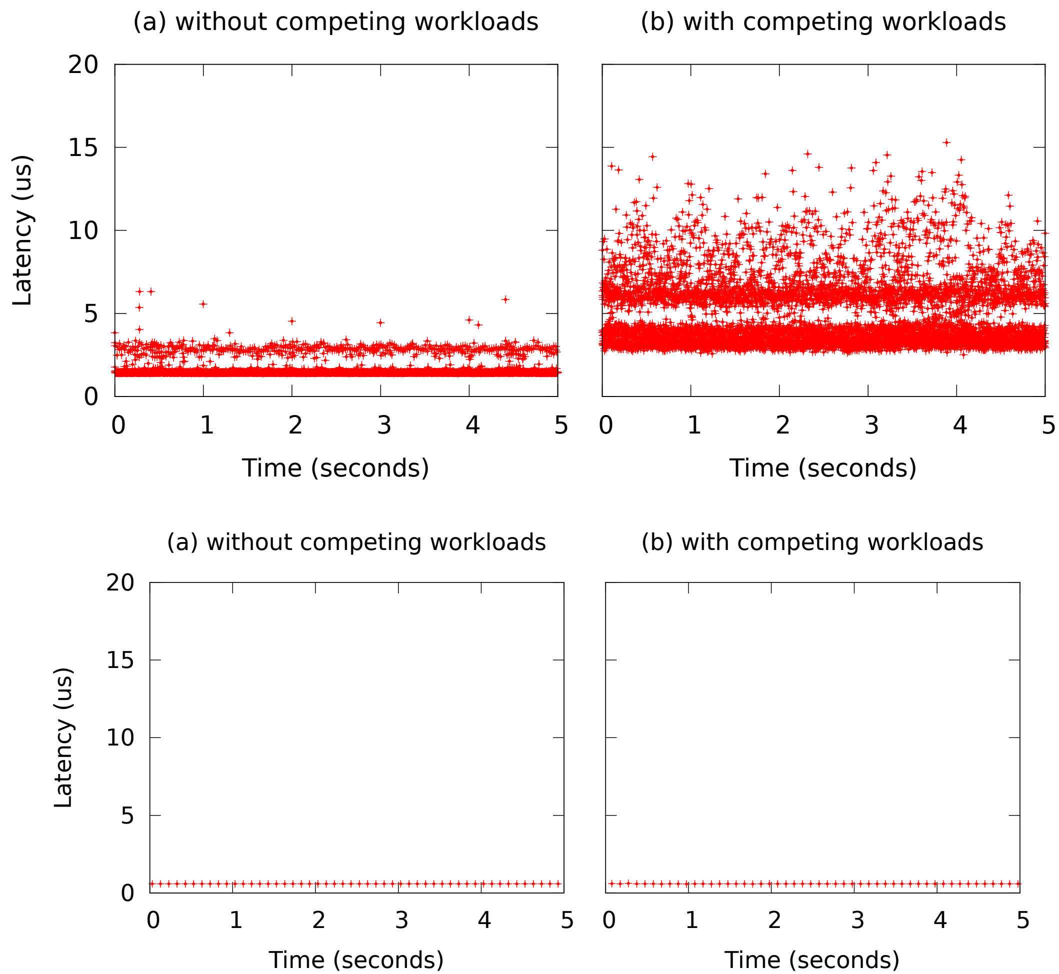
Execution Time of getpid()

- Co-kernel has the **best average performance**
- Co-kernel has the **most consistent performance**
- System call forwarding has longer latency and suffers from cross stack performance interference

Noise Analysis

Linux

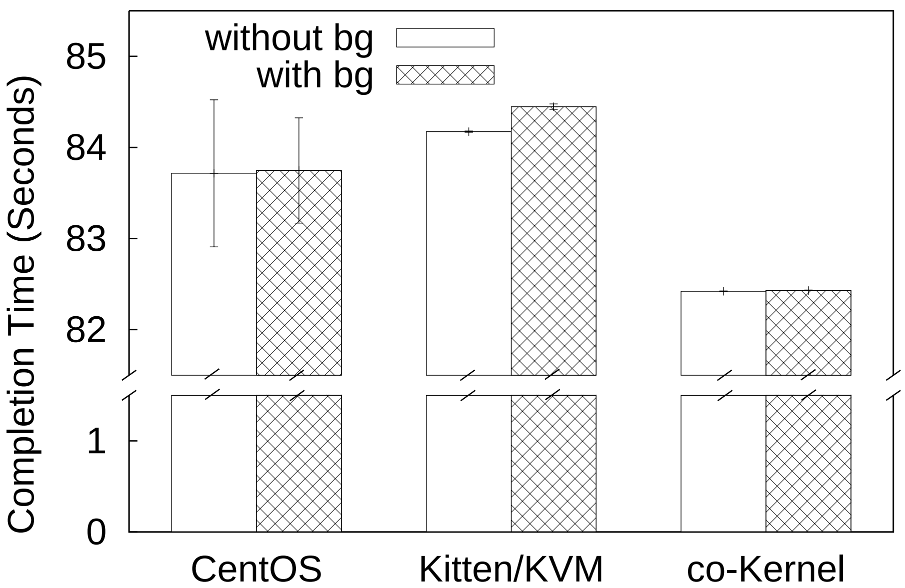
Kitten co-kernel



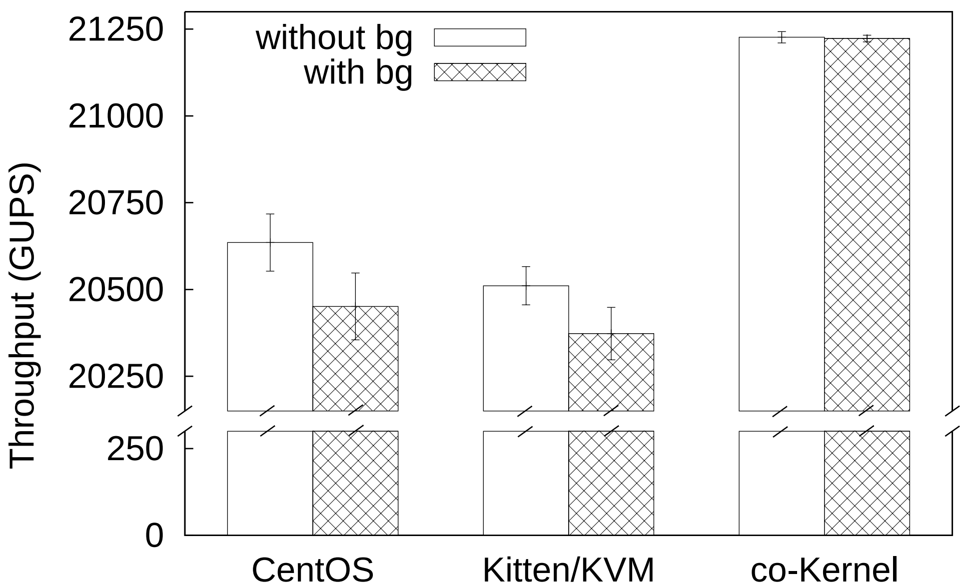
Co-Kernel： less noise + better isolation

* Each point represents the latency of an OS interruption

Single Node Performance



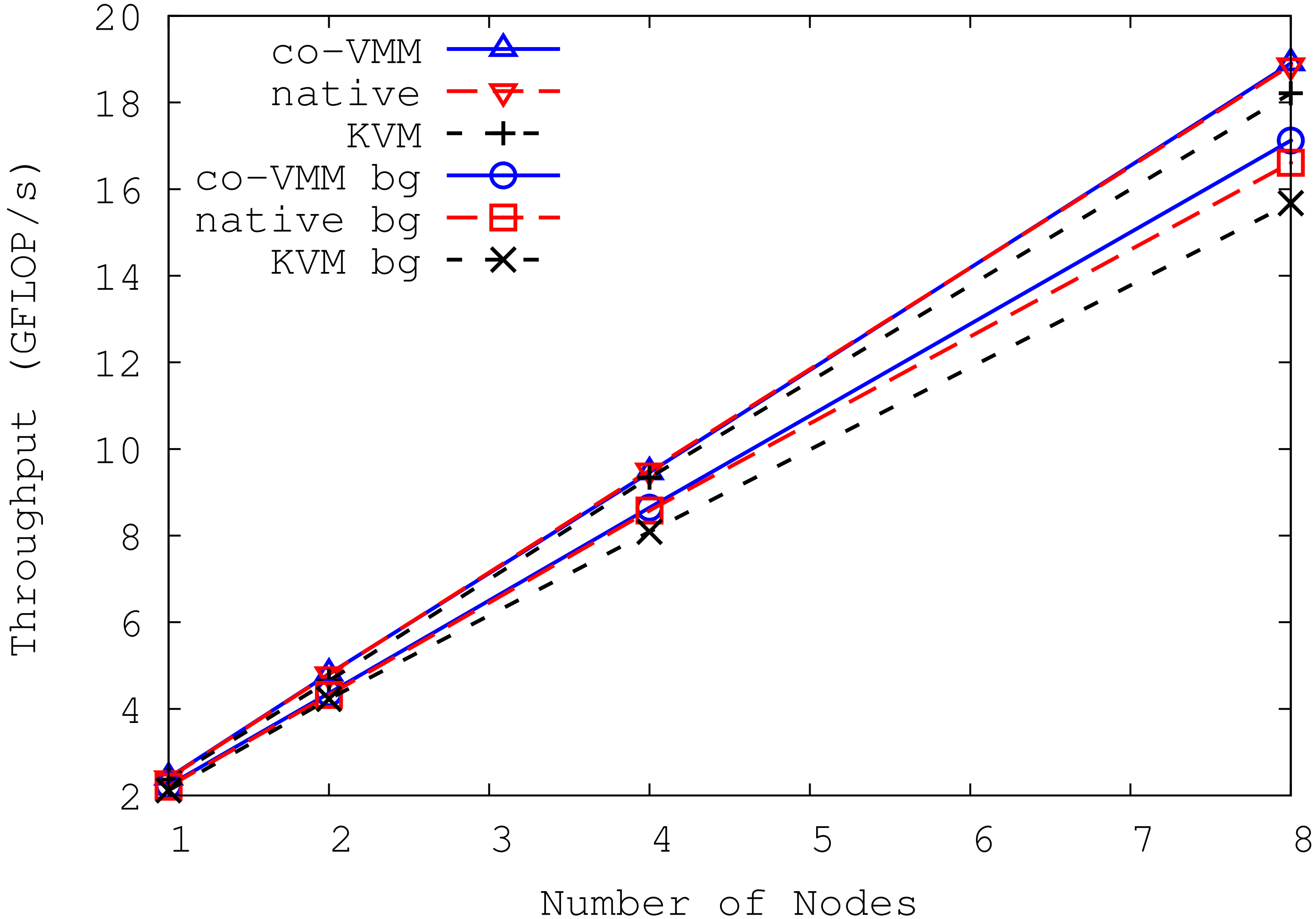
CoMD Performance



Stream Performance

Co-Kernel: consist performance + performance isolation

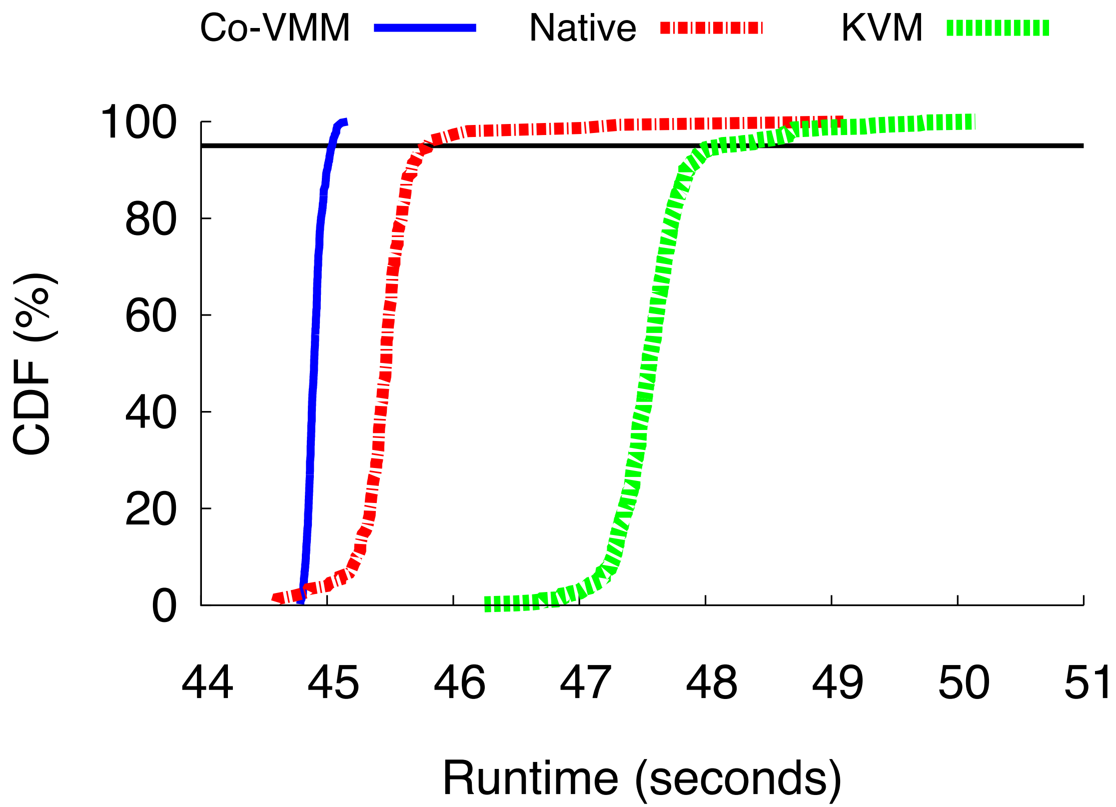
8 Node Performance



w/o bg: co-VMM achieves native Linux performance

w/ bg: co-VMM outperforms native Linux

Co-VMM for HPC in the Cloud



CDF of HPCCG Performance (running with Hadoop, 8 nodes)

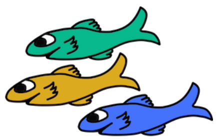
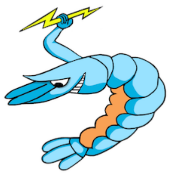
co-VMM: consistent performance + performance isolation

Related Work

- Exascale operating systems and runtimes (OS/Rs)
  - **Hobbes (SNL, LBNL, LANL, ORNL, U. Pitt, various universities)**
  - Argo (ANL, LLNL, PNNL, various universities)
  - FusedOS (Intel / IBM)
  - mOS (Intel)
  - McKernel (RIKEN AICS, University of Tokyo)

Our uniqueness: performance isolation, dynamic resource composition, lightweight virtualization

Conclusion

- Design and implementation of the Pisces co-kernel architecture
  - Pisces framework  http://www.prognosticlab.org/pisces/
  - Kitten co-kernel  https://software.sandia.gov/trac/kitten
  - Palacios VMM for Kitten co-kernel  http://www.prognosticlab.org/palacios

- Demonstrated that the co-kernel architecture provides
  - Optimized execution environments for in situ processing
  - Performance isolation

Thank You

Jiannan Ouyang

- Ph.D. Candidate @ University of Pittsburgh
- ouyang@cs.pitt.edu
- http://people.cs.pitt.edu/~ouyang/

- The Prognostic Lab @ U. Pittsburgh
  - http://www.prognosticlab.org