

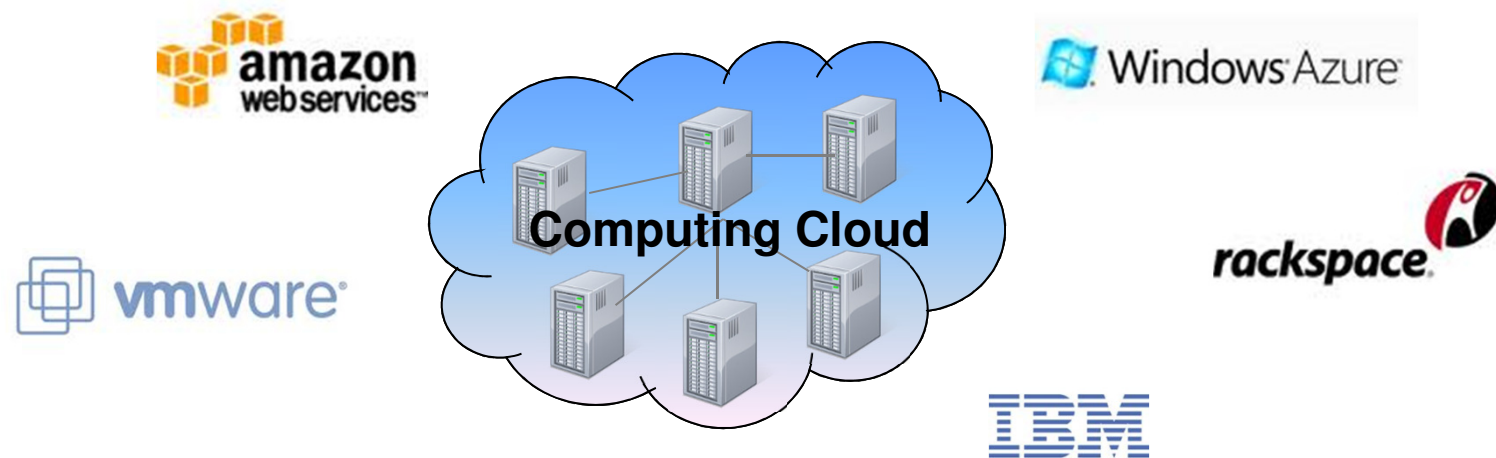
Is Co-scheduling Too Expensive for SMP VMs?

Orathai Sukwong and Hyong S. Kim
{osukwong,kim}@ece.cmu.edu

Carnegie Mellon University, PA, USA

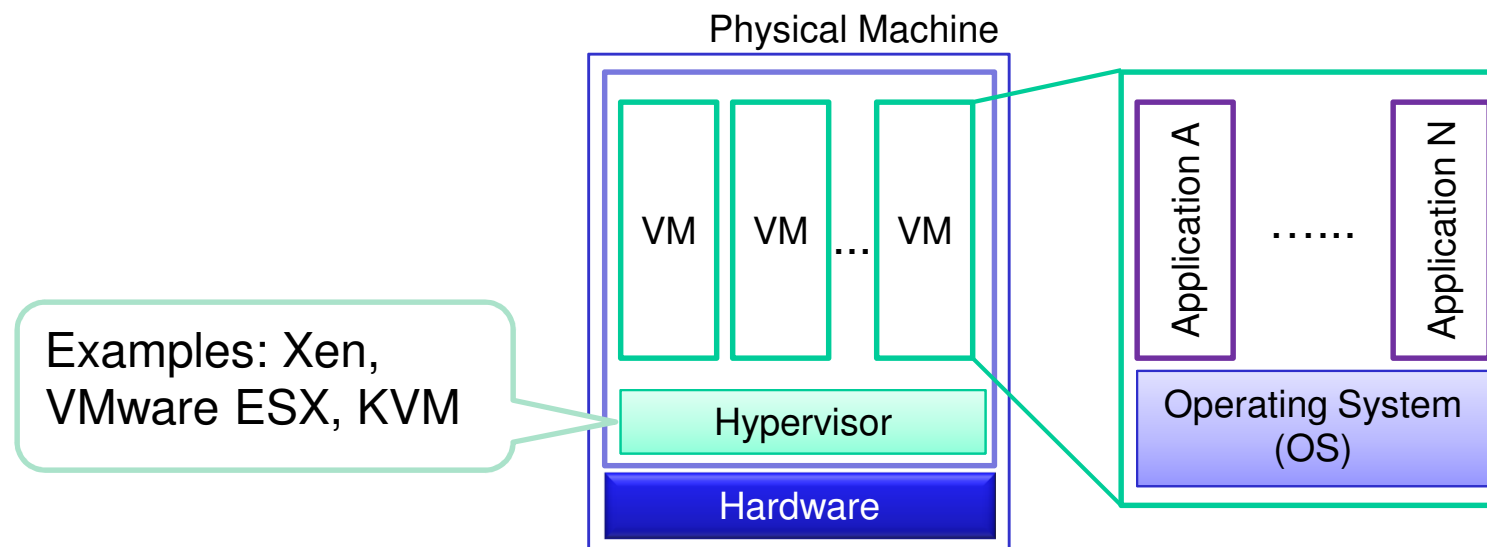
EuroSys 2011 - April 12, 2011

Cloud & Virtualization



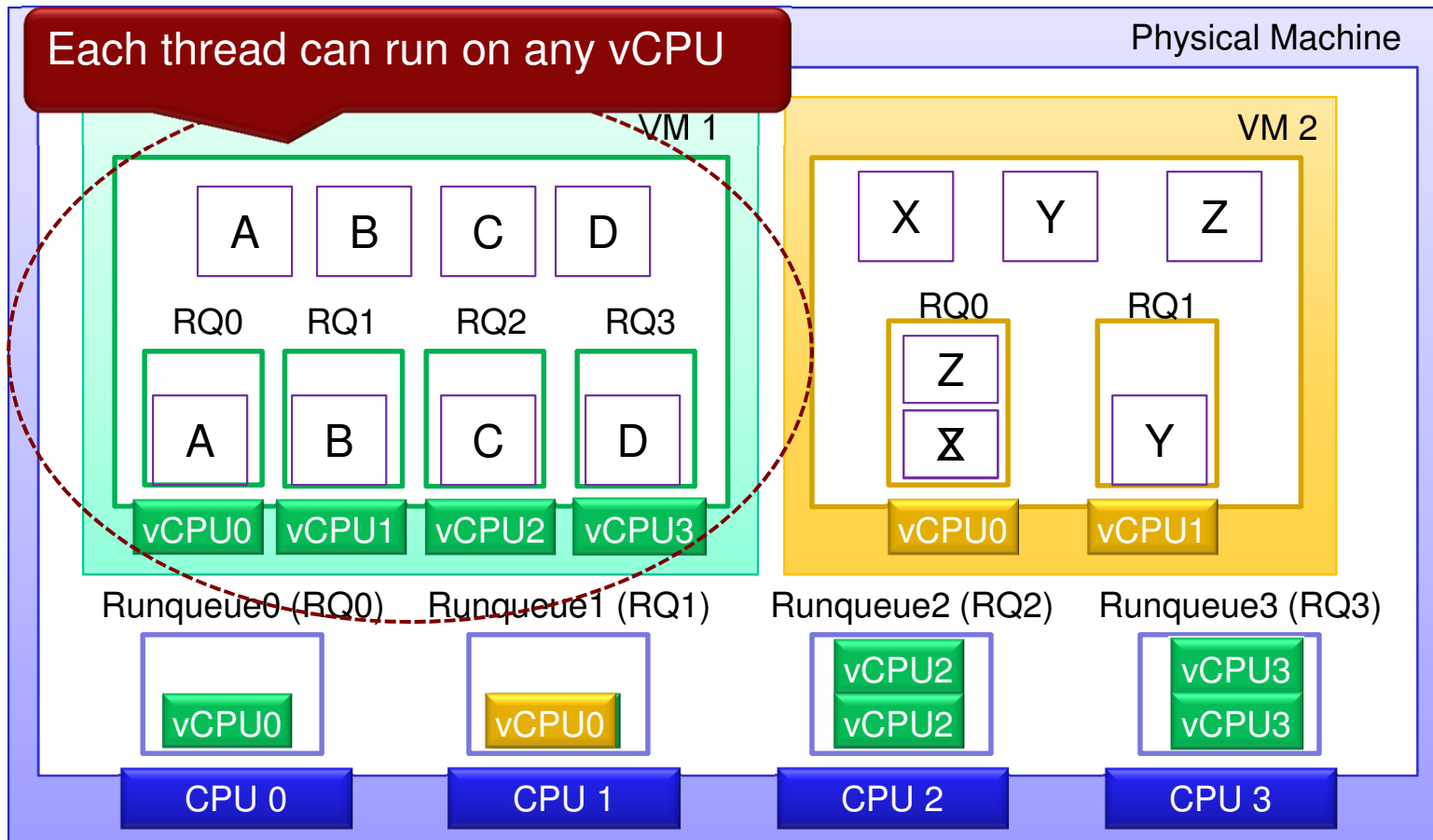
- Virtualization
 - Allow multiple servers to share the same physical machine
 - Achieve higher utilization of physical machines
 - Ease infrastructure management

Virtualization

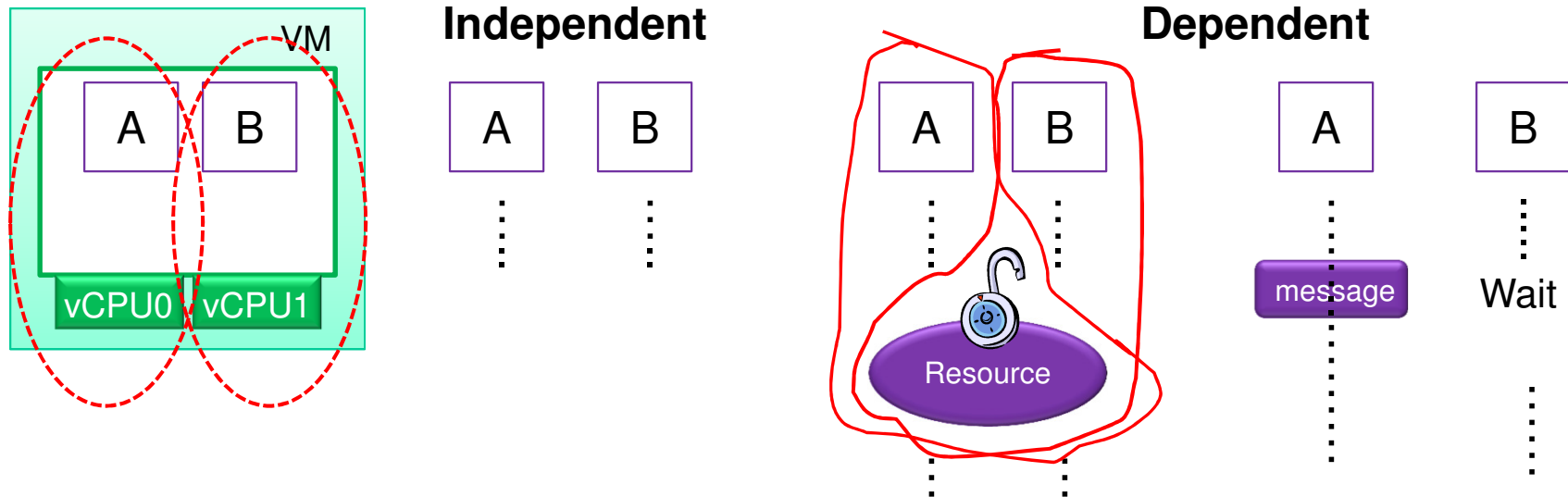


- A symmetric multiprocessing (SMP) VM/guest
 - A VM with > 1 virtual CPU (vCPU)
 - Each vCPU behaves identically
- vCPU siblings = vCPUs belonging to the same VM

VM Scheduling



Synchronization in SMP VMs



- Assuming that *vCPU0* runs *A* and *vCPU1* runs *B*
- If *A* and *B* are dependent, *vCPU0* and *vCPU1* are also dependent

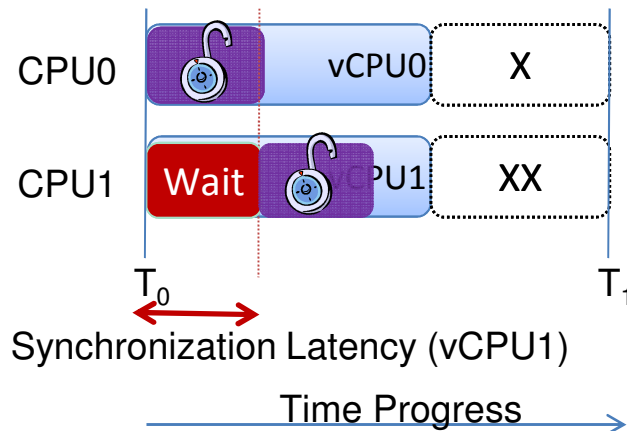
Synchronization Latency Problem

- Recall: each thread can run on any CPU

Assume vCPU0 successfully acquires the lock

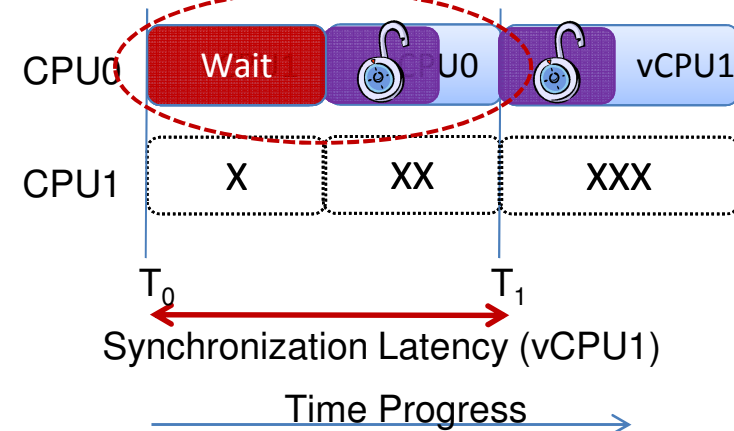
Stacking vCPUs

Schedule vCPU0 & vCPU1 simultaneously



vCPU1 waits $< (T_1 - T_0)$ for the lock

Schedule vCPU1 before vCPU0



vCPU1 waits $(T_1 - T_0)$ for the lock

- Synchronization latency can increase significantly, depending on scheduling order

How Often Does Scheduler Stack vCPUs?

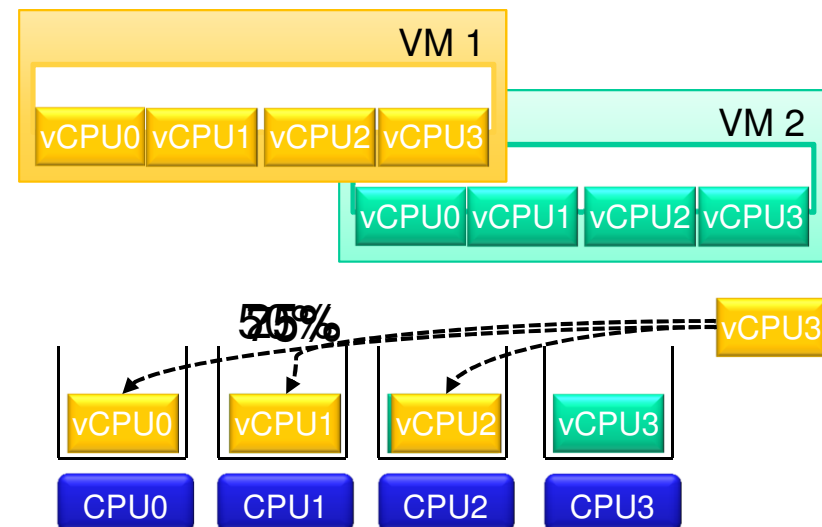
- Run 4-vCPU VMs on a 4-CPU physical host
- Run the CPU-bound workload inside the VMs
 - 100% utilization on each vCPU

# VMs	≥ 2 vCPU siblings stacking on the same CPU
1	5.564%
2	43.127%
3	45.932%

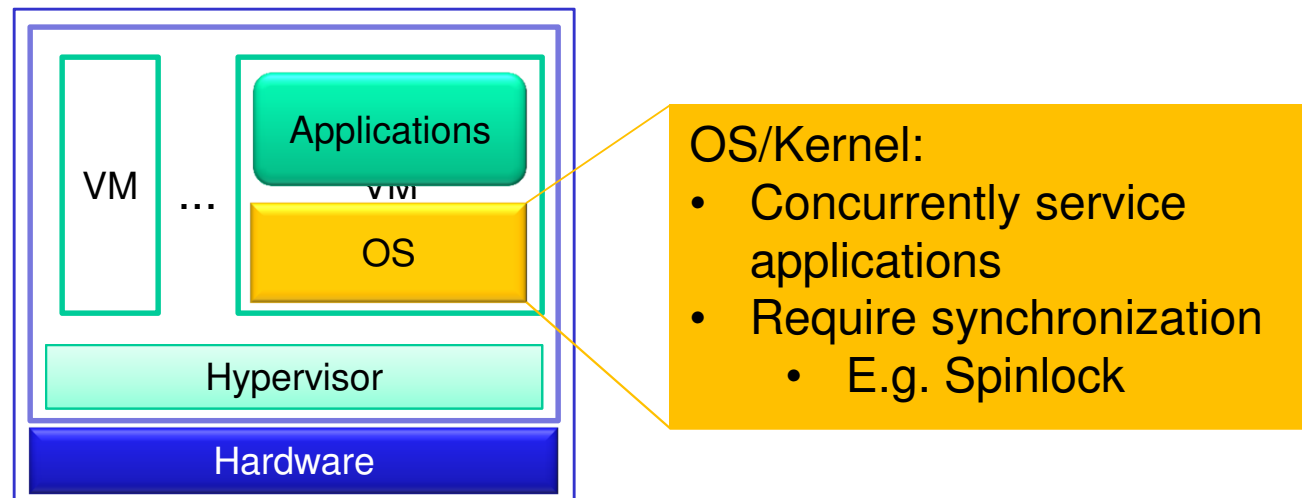
How Often Does Scheduler Stack vCPUs?

- Run 4-vCPU VMs on a 4-CPU physical host
- Run the CPU-bound workload inside the VMs
 - 100% utilization on each vCPU

# VMs	≥ 2 vCPU siblings stacking on the same CPU
1	5.564%
2	43.127%
3	45.932%



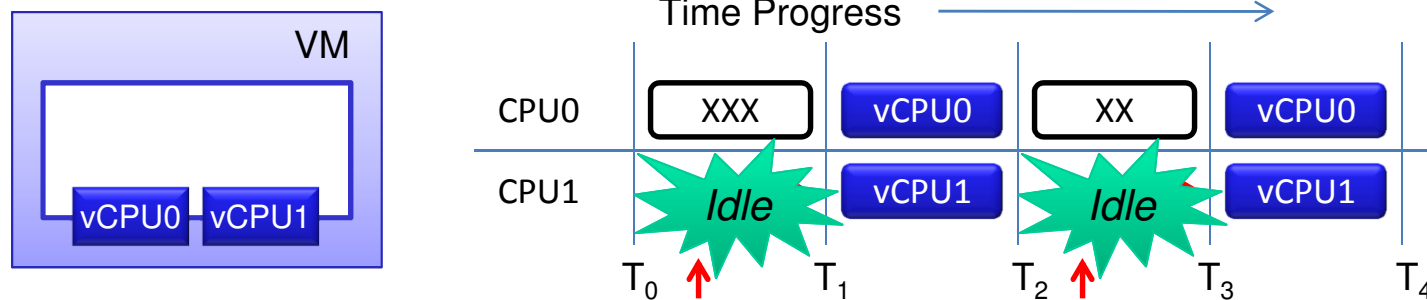
What if running non-concurrent application?



- A VM runs both applications and OS
- Even though running synchronization-free applications inside the VM, the VM may still encounter the synchronization latency problem.

Co-scheduling

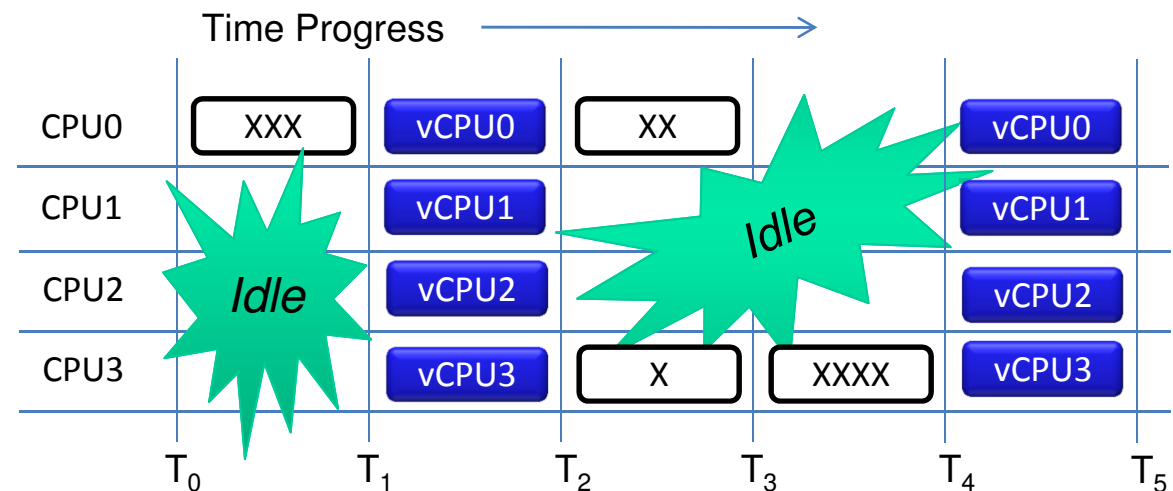
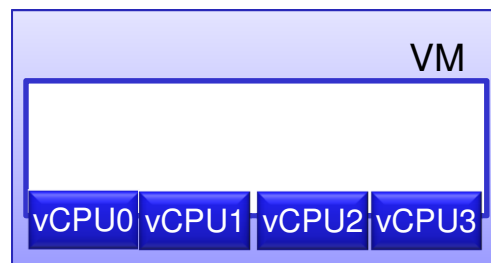
- Schedule vCPU siblings simultaneously



- Drawback
 - CPU fragmentation
 - Lower utilization and delay vCPU execution

Co-scheduling

- Schedule vCPU siblings simultaneously



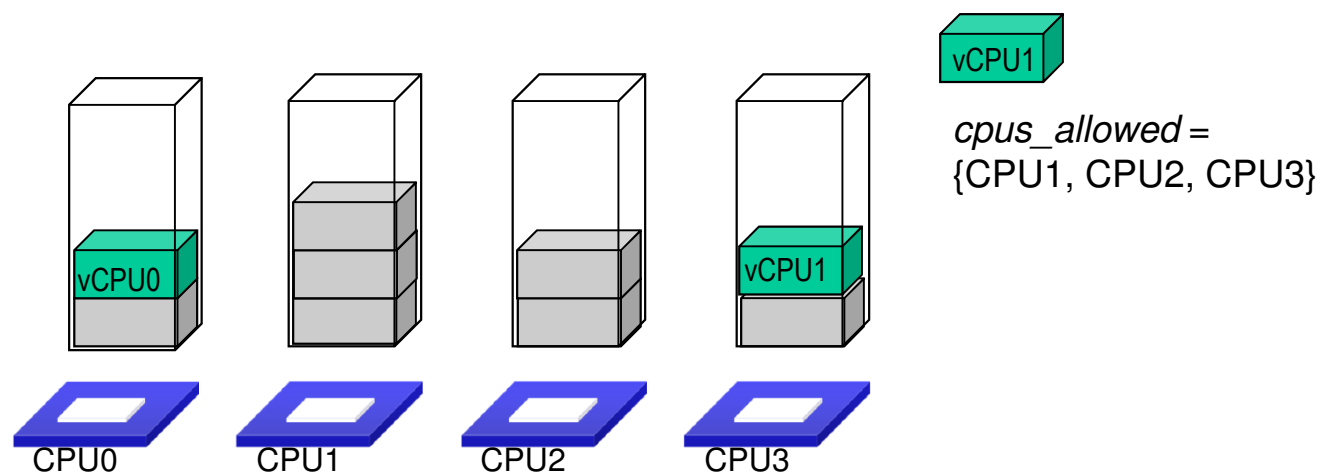
- Drawback
 - CPU fragmentation
 - *Significantly* lower utilization and delay vCPU execution

Related Work

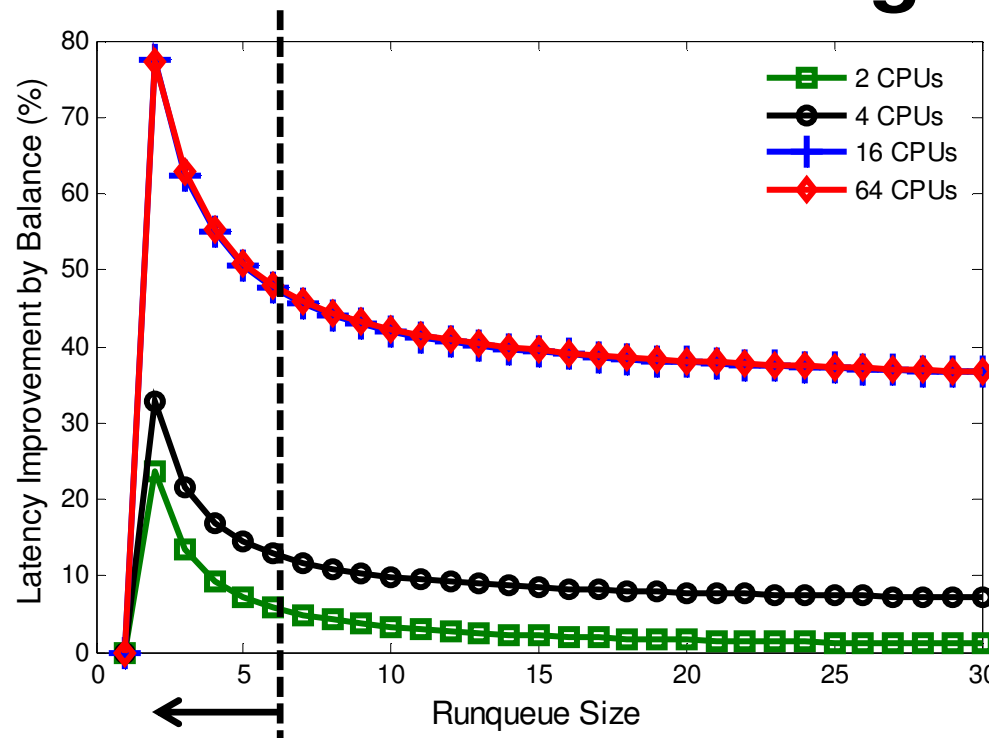
- Co-scheduling
 - VMware => *Relax* co-scheduling to mitigate CPU fragmentation
 - Strict co-scheduling (ESX 2.x)
 - Relaxed co-scheduling (ESX 3.x)
 - Further relaxed co-scheduling (ESX 4.x)
 - Xen => *Selectively* apply co-scheduling to the concurrent VMs
 - Weng2009, Bai2010
- Affinity-based scheduling
 - *Statically* bind a vCPU to a set of CPUs
 - *Carefully* bind vCPUs to avoid overloading particular CPUs

Our Balance Scheduling

- Simple idea: *Balance* vCPU siblings across CPUs
 - Never put any two vCPU siblings into the same RQ
 - No need to force vCPU siblings to be scheduled simultaneously
- Cause no CPU fragmentation and improve the performance of SMP VMs as well as co-scheduling does
- Easy to implement
 - Modify each vCPU's *cpus_allowed* field before selecting a RQ



Synchronization Latency Improvement By Balance Scheduling



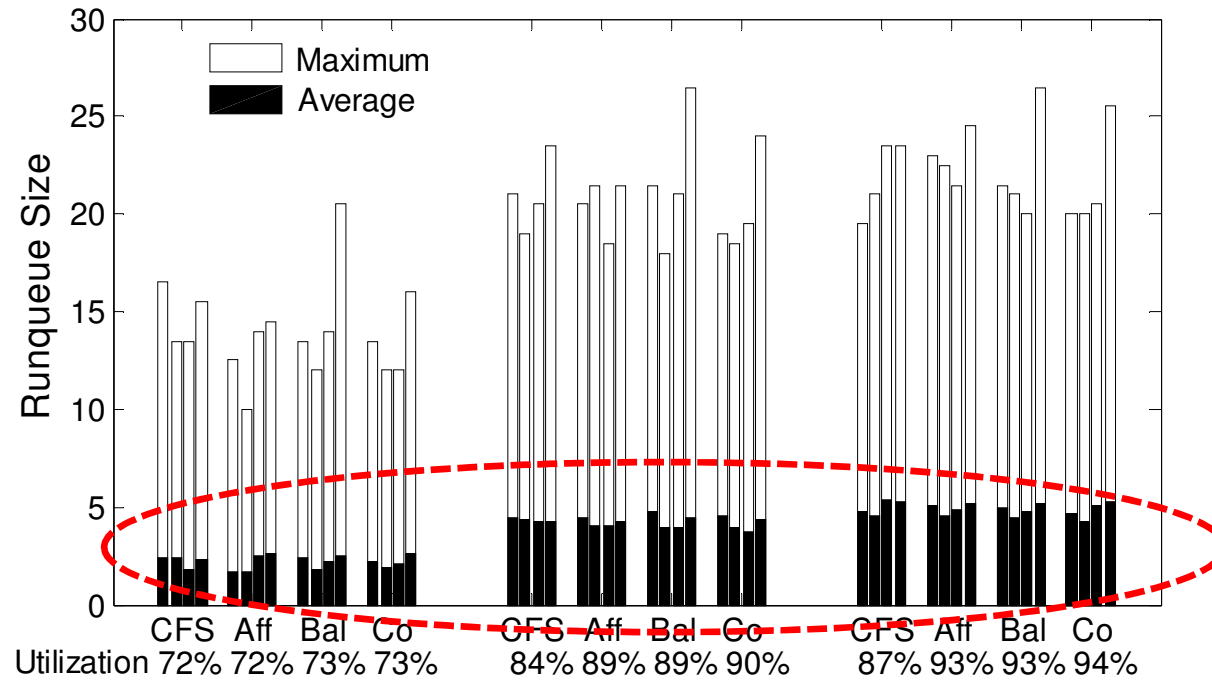
- The improvement decreases as the runqueue size grows
- The empirical results show that the runqueue size is ≤ 6 on average

Evaluate Scheduling Algorithms

- Completely Fair Scheduler (CFS)
 - Default scheduler in KVM
 - Treat each vCPU the same
- Affinity-based algorithm (Aff)
 - Statically bind each vCPU to a CPU before running an experiment
 - # vCPUs per physical CPU is relatively the same
 - Do not assign any two vCPU siblings to the same physical CPU
- Co-scheduling algorithm (Co)
 - Implement on top of CFS
 - No longer have CPU fragmentation problem but may incur additional context switching
- Our balance scheduling algorithm (Bal)

Runqueue Size

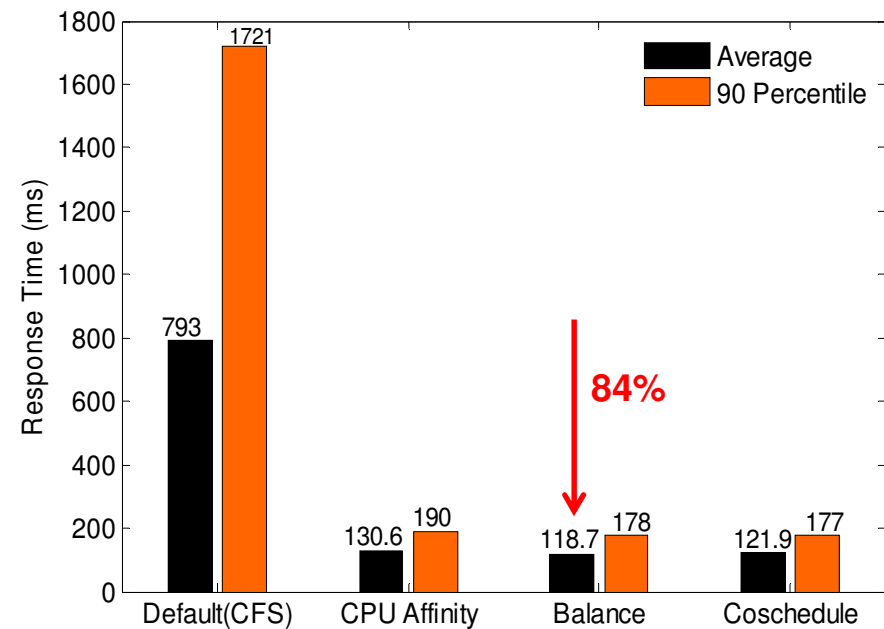
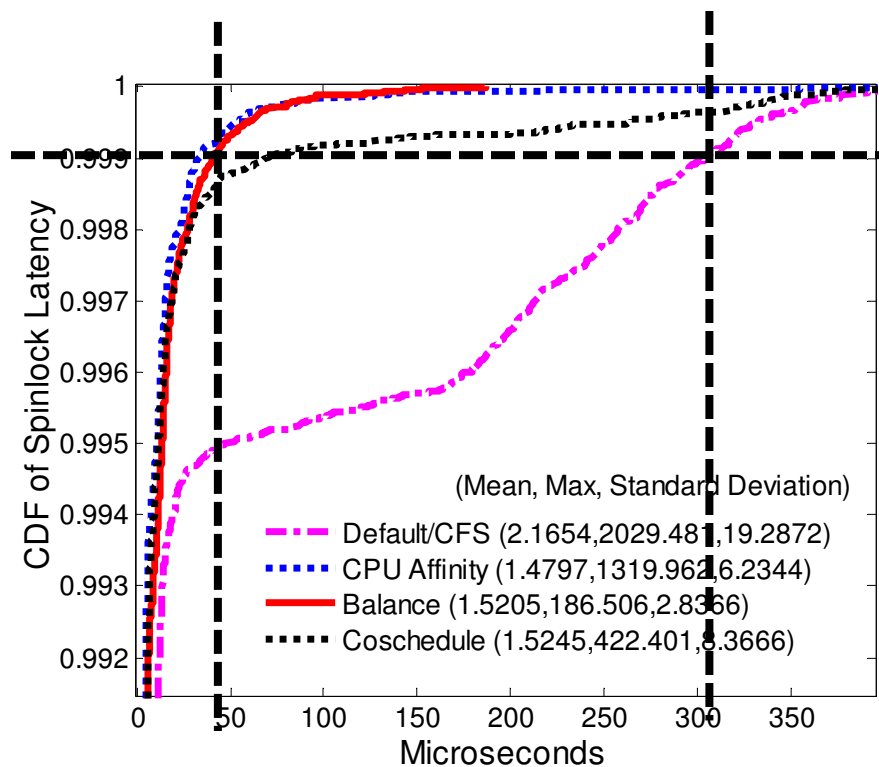
- Run 14 VMs on a 4-CPU physical machine
- Expect 56 vCPU threads + I/O QEMU threads



- Runqueue size is about **4-6** on average

TPC-W

- Run 3 four-vCPU VMs for a proxy server, an application server, and a database server on a 4-CPU host.

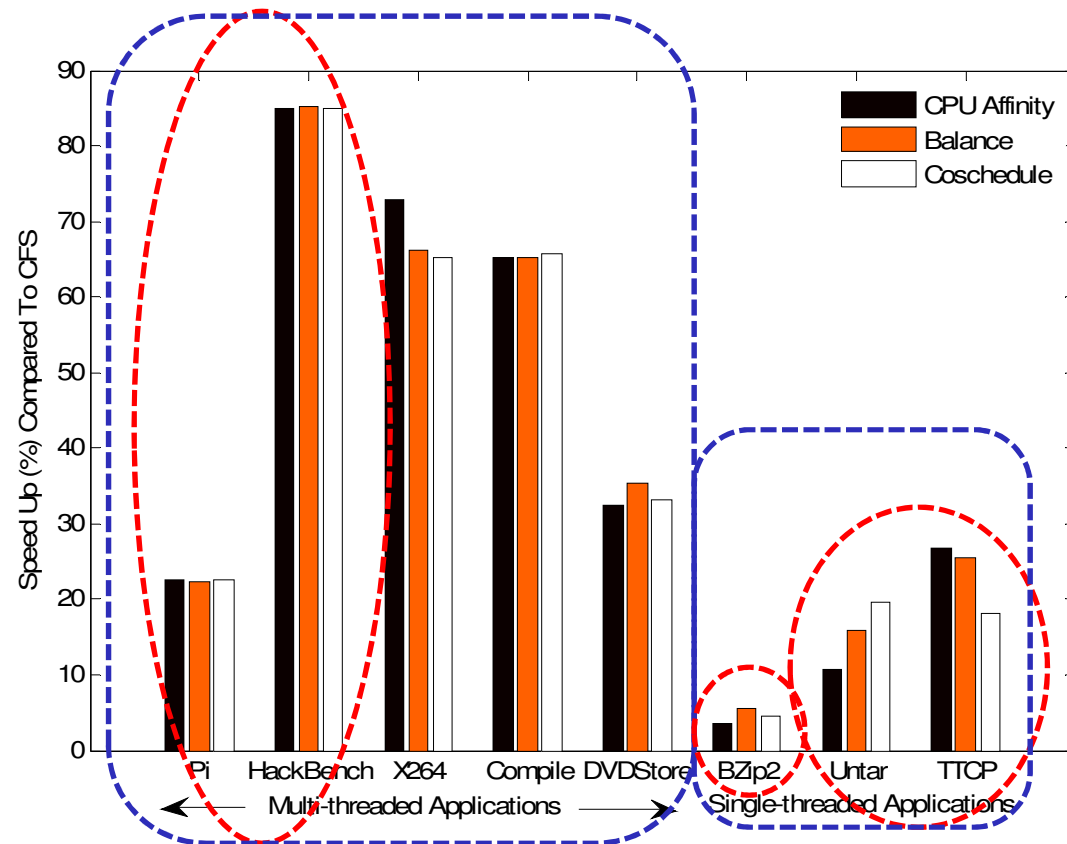


Spinlock Latency \uparrow \rightarrow TCP Retransmission \uparrow \rightarrow Response Time \uparrow

Different Applications

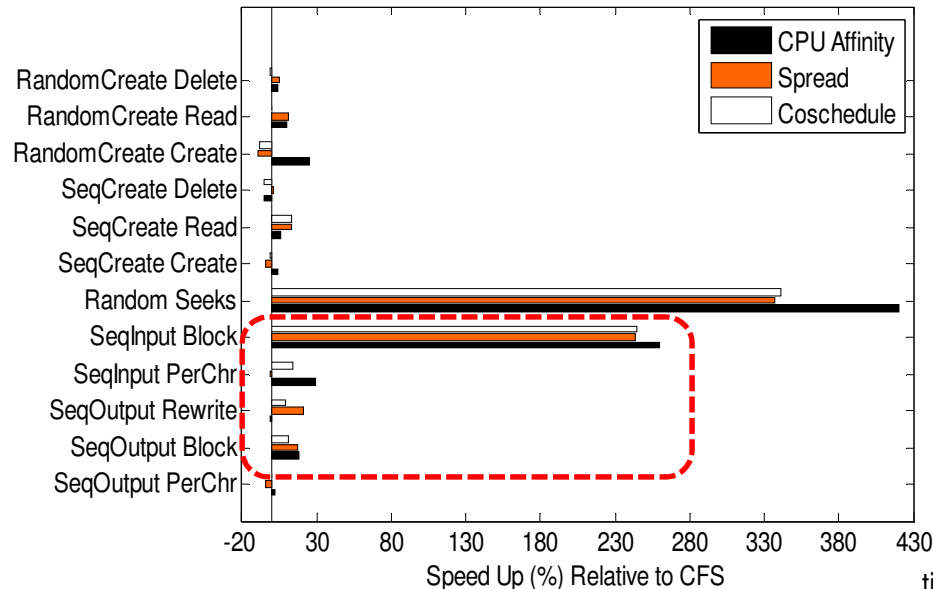
Run 2 VMs in the host

- One 4-vCPU VM
 - Run an application
- One 2-vCPU VM
 - Run the CPU-bound workload



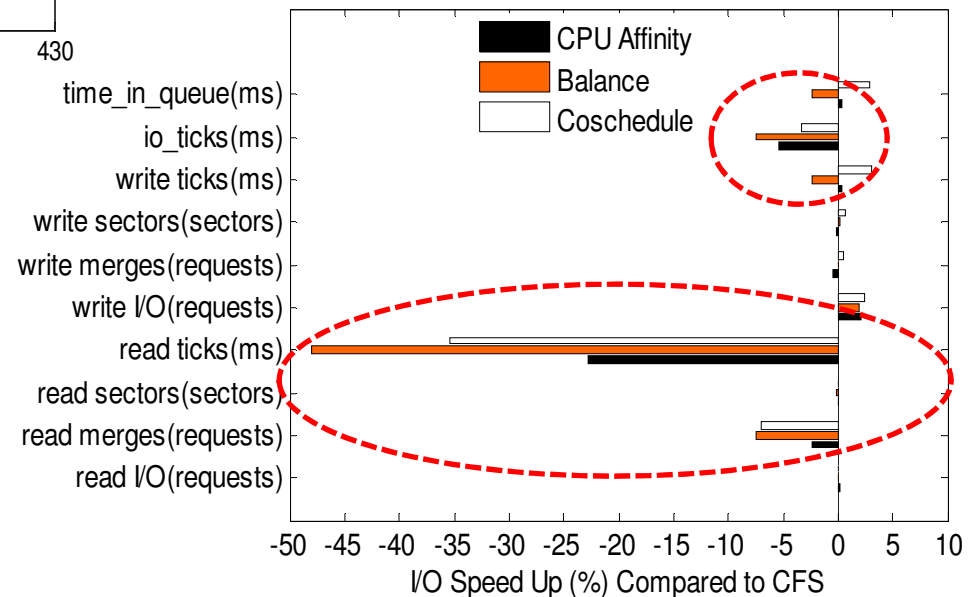
- *Improvement depends on the synchronization degree in VMs*
- *Balance scheduling can improve application performance as much as co-scheduling*

Bonnie++



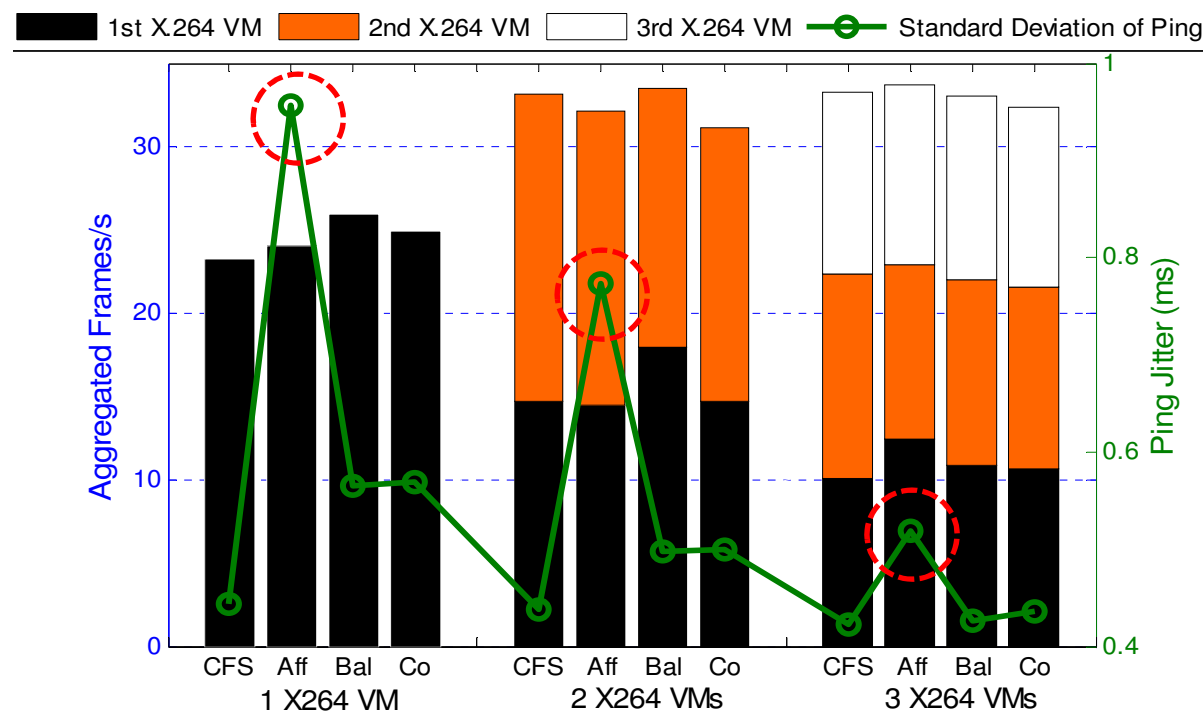
- Running a *single-threaded* application can also benefit from co-scheduling, balance scheduling and affinity-based scheduling

- Run Bonnie++ in a 4-vCPU VM with a 2-vCPU VM running the CPU-bound workload
- Bonnie++ => single-threaded, I/O-intensive



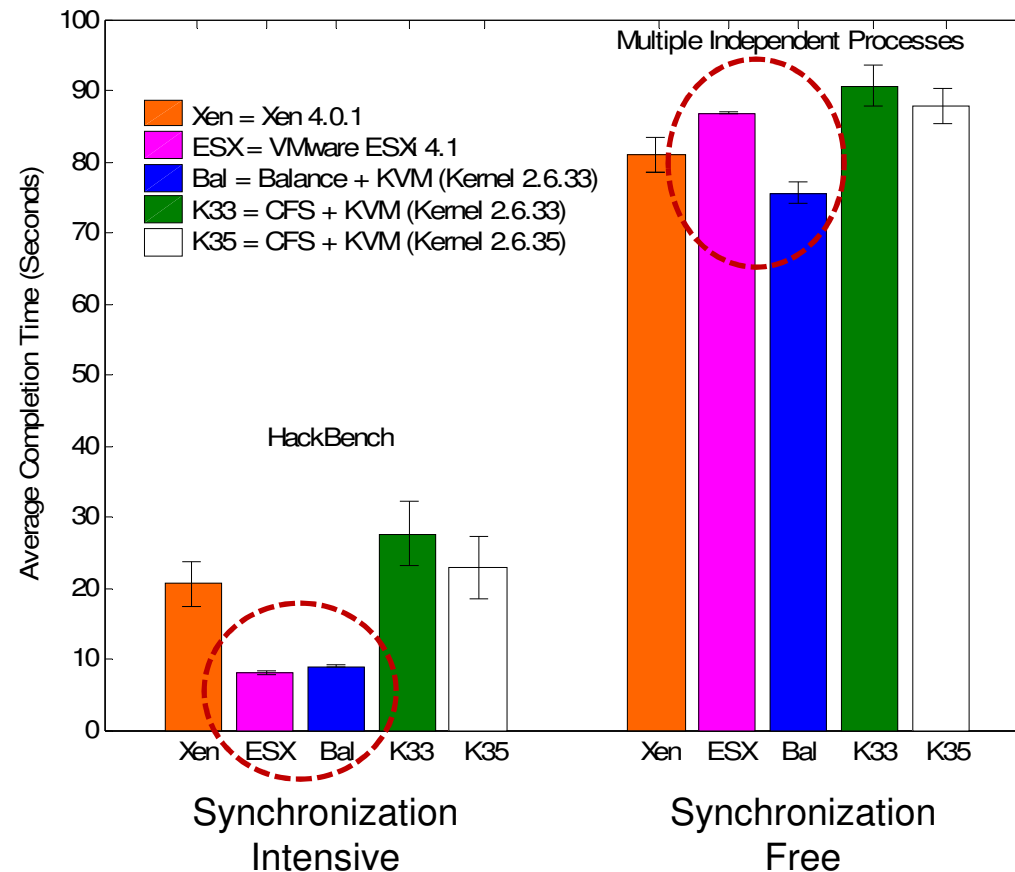
X264 & Ping

- 3 four-vCPU VMs run X.264 and 1 one-vCPU VM runs Ping



- In *affinity-based scheduling*, the Ping vCPU may get stuck in the *busiest* CPU. With *balance scheduling*, the Ping vCPU can choose the *idlest* CPU.

Different Hypervisors



Our *balance scheduling* works well with both *synchronization-intensive* and *synchronization-free* applications

Discussion

- What are most applications?
 - Synchronization-intensive? Synchronization-free?
 - Many legacy applications are still single-threaded.
 - A good number of applications are concurrent programs to take advantage of multi-core architecture
 - A rule of thumb of concurrent programming is using minimal synchronization to promote parallelism
 - We believe that future parallel programs should be leaning toward the minimal usage of synchronization
- => Our balance scheduling should be a way to go!*

Conclusion

- Synchronization latency problem can significantly degrade application performance
 - Even if running synchronization-free applications due to the synchronization in the guest OS
- Co-scheduling can be too expensive for SMP VMs with minimal synchronization
 - CPU fragmentation
 - Reduce the host-CPU utilization and delay vCPU execution
- Our balance scheduling can
 - Perform similarly to co-scheduling given concurrent SMP VMs
 - Work well with minimal-synchronization VMs