Preserving I/O Prioritization in Virtualized OSes

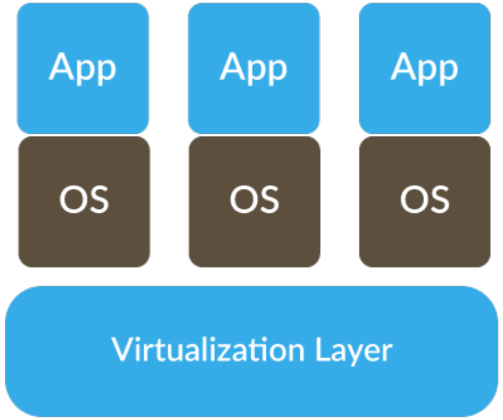
Kun Suo[1], Yong Zhao[1], Jia Rao[1], Luwei Cheng[2], Xiaobo Zhou[3], Francis C. M. Lau[4]

The University of Texas at Arlington[1], Facebook[2],

University of Colorado, Colorado Springs[3], The University of Hong Kong[4]

Virtualization

- A powerful abstraction that transforms a physical resource into a more general, easy-to-use virtual form
  - ✓ Workload consolidation
  - ✓ Fault isolation
  - ✓ Service migration

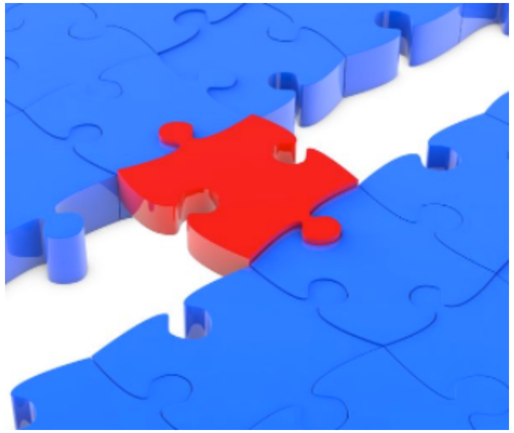


- Ubiquitous in modern IT management
  - ✓ Java virtual machine (JVM), container, and full-fledged virtual machine (VM)

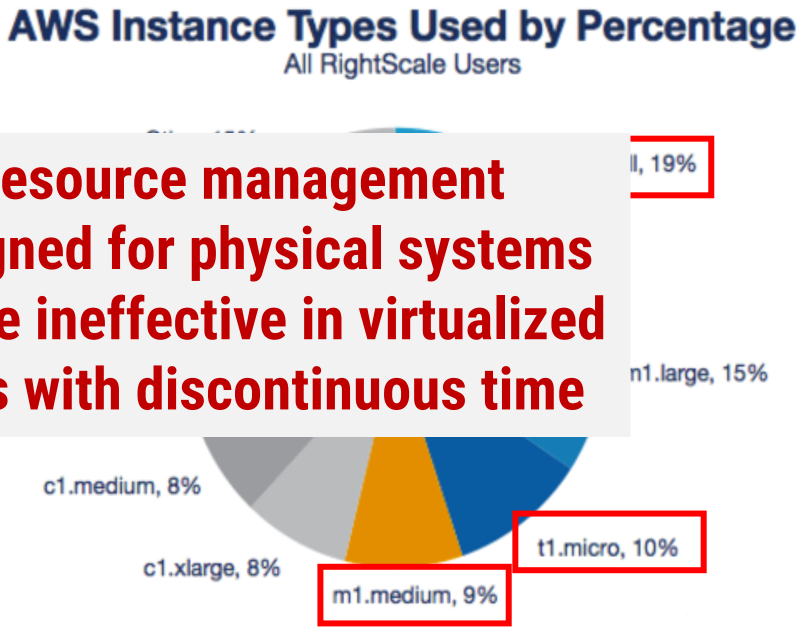
"A program running in virtualized environments should exhibit a behavior essentially identical to that in physical environments."

-- [Popek and Goldberg'74]

An Inherent Semantic Gap



- Virtualization presents the illusion of **dedicated**, **continuous** hardware, but operates on **shared**, **discontinuous** resources
- Time is discontinuous in multi-tenant systems
  - ✓ VMs with capped CPU capacity account for **40%** Amazon EC2 usage
  - ✓ CPU multiplexing in public and private clouds



Resource management designed for physical systems can be ineffective in virtualized OSes with discontinuous time

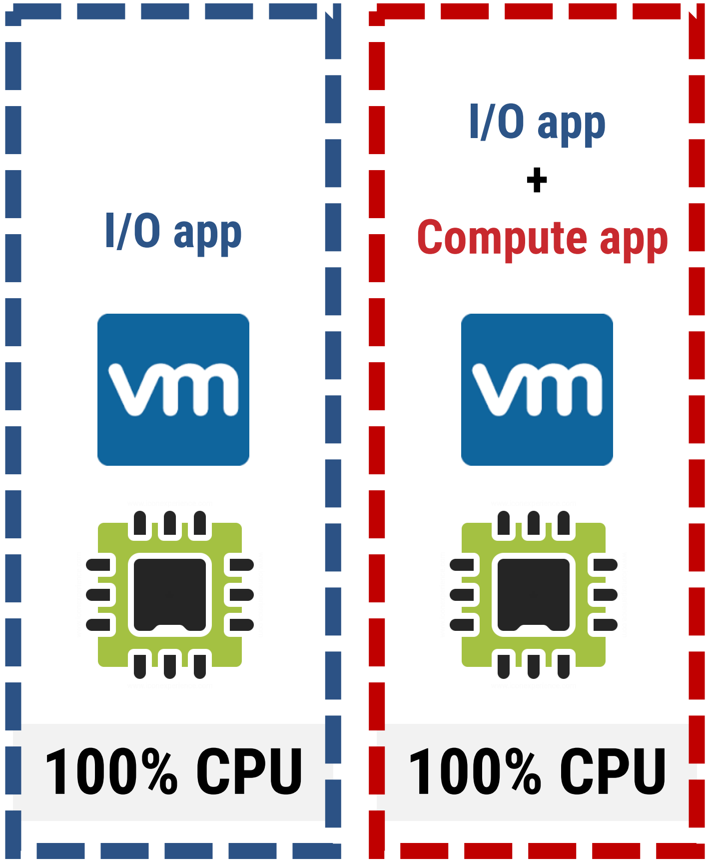
VM running	VM descheduled	VM running
Clock ticking	VM clock stops	Sync with host clock and restart ticking

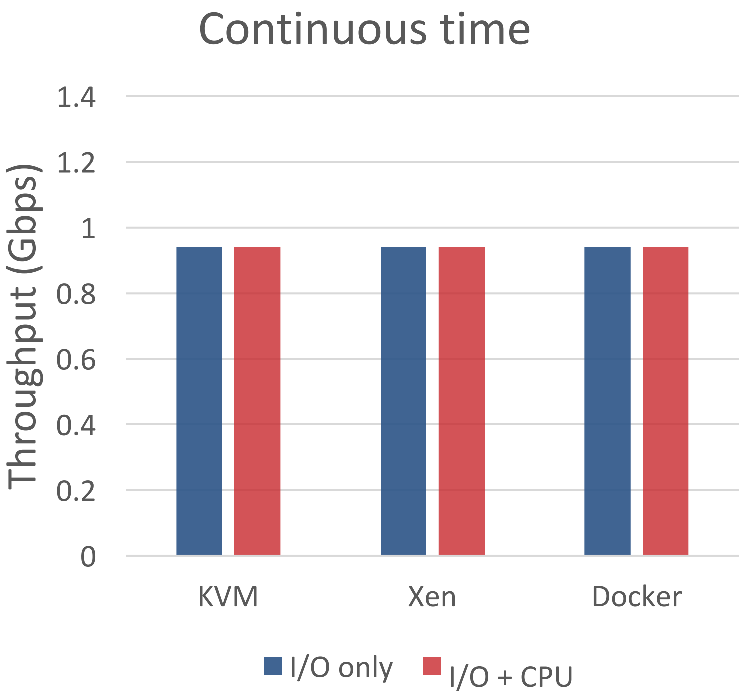
VM's perception of time passage:
discontinuous; clock reading jumps with intermittent gaps between each run

I/O prioritization

- An important OS design to improve system responsiveness without compromising throughput.
- I/O prioritization relies on two mechanisms
  - ✓ The identification of I/O-bound tasks
  - ✓ The preemption of CPU-bound tasks
- User-configured priority -- **static priority**
  - ✓ Real-time vs. normal priorities, e.g., SCHED_RR vs. SCHED_OTHER
- Linux completely fair scheduler (CFS) -- **dynamic priority**
  - ✓ uses **virtual runtime** (vruntime) to track how much time a task has run on CPU
  - ✓ prioritizes tasks with smaller vruntimes ("**I/O-bound**") over those with larger vruntimes ("**CPU-bound**")

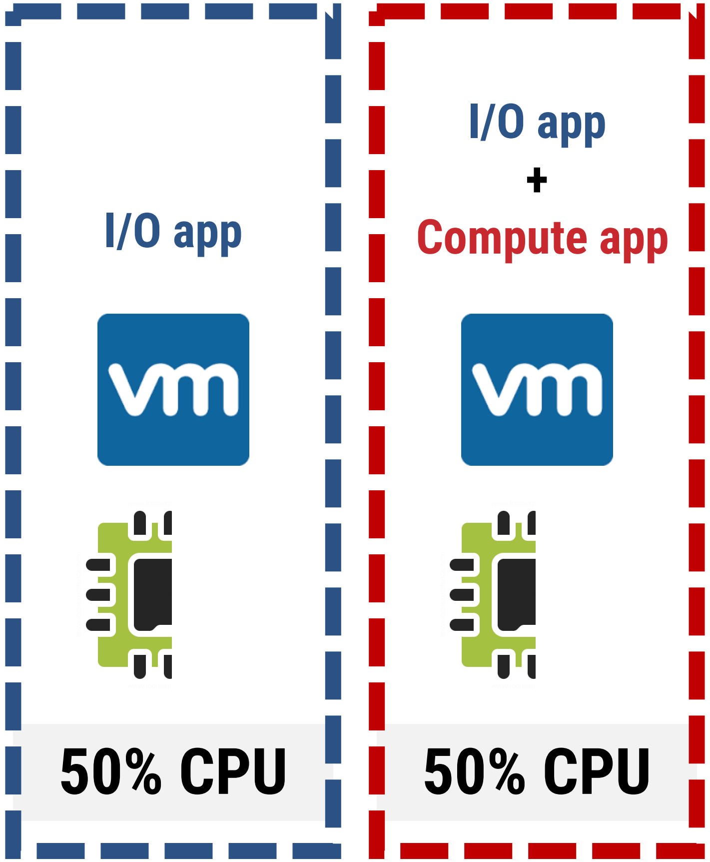
Violation of I/O Prioritization

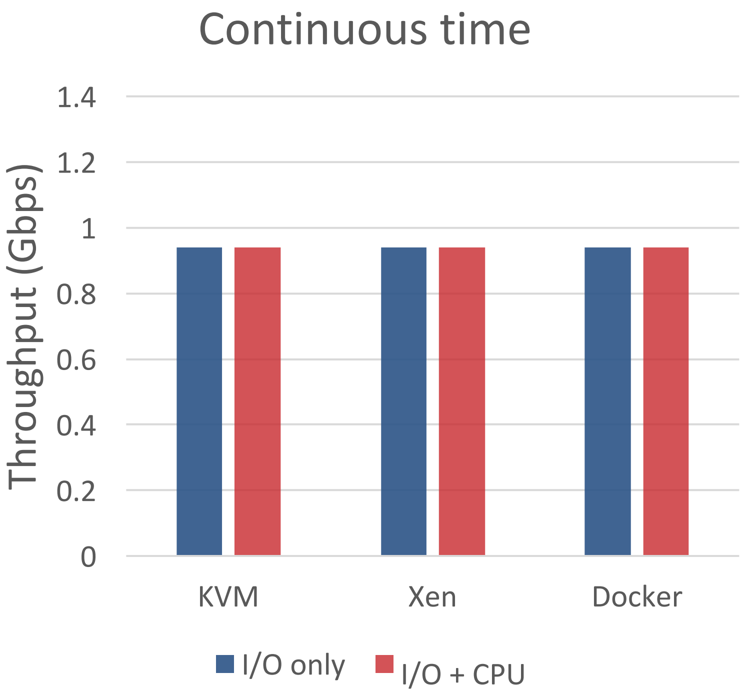


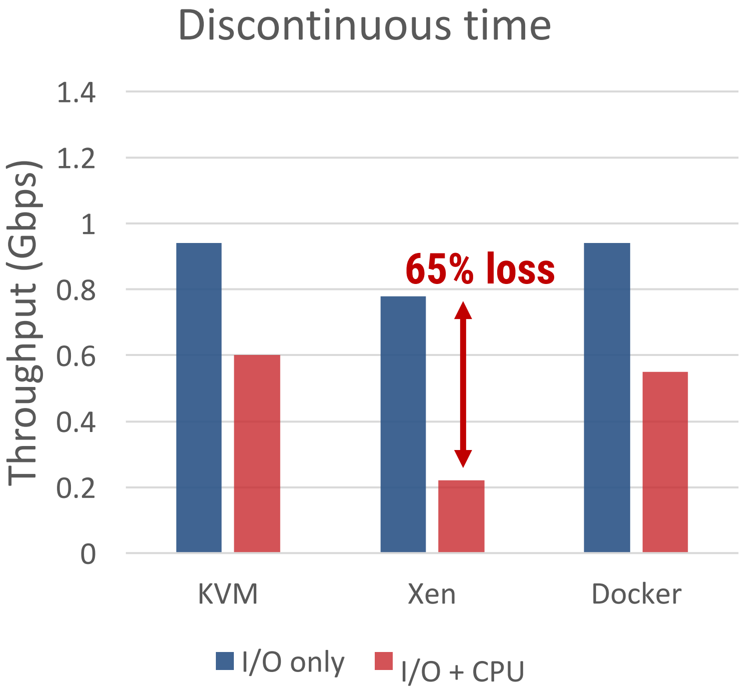


I/O task always
has a higher priority

Violation of I/O Prioritization

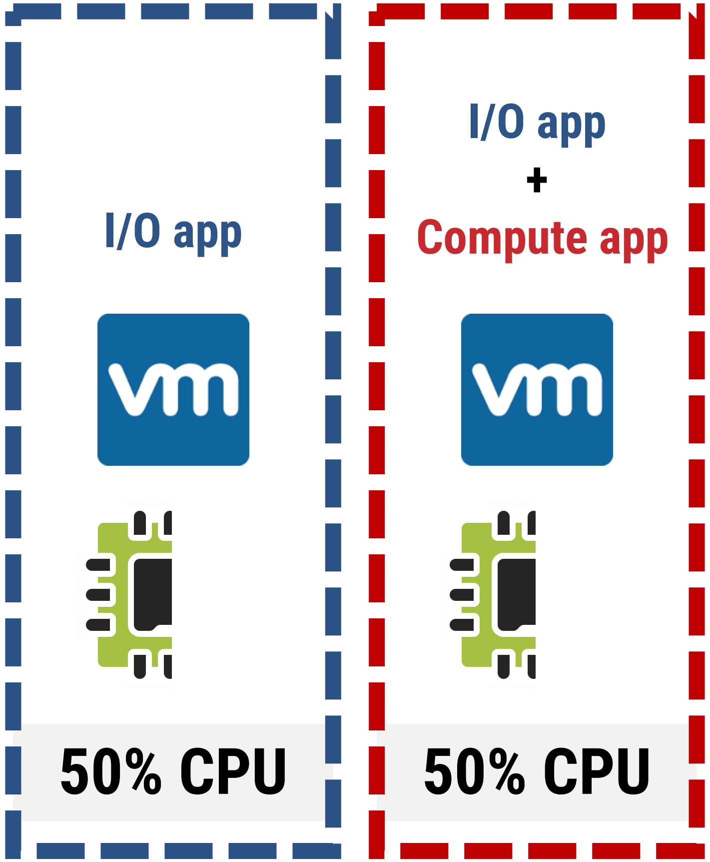


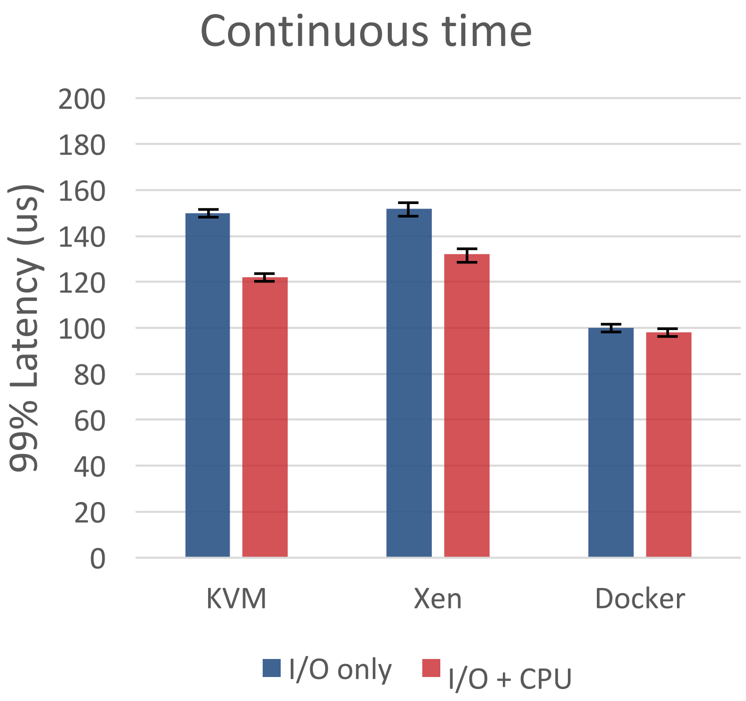


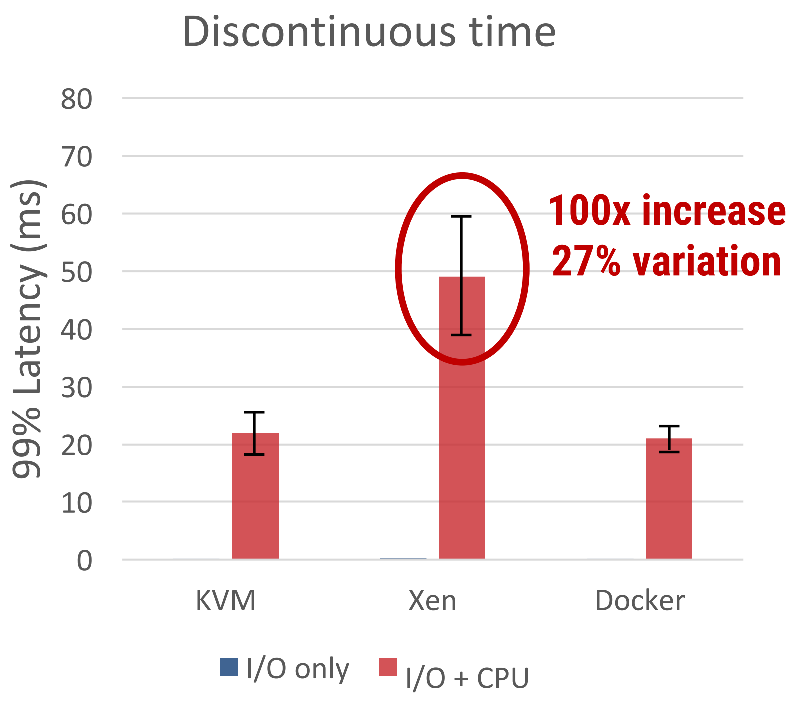


I/O task always has a higher priority

Violation of I/O Prioritization



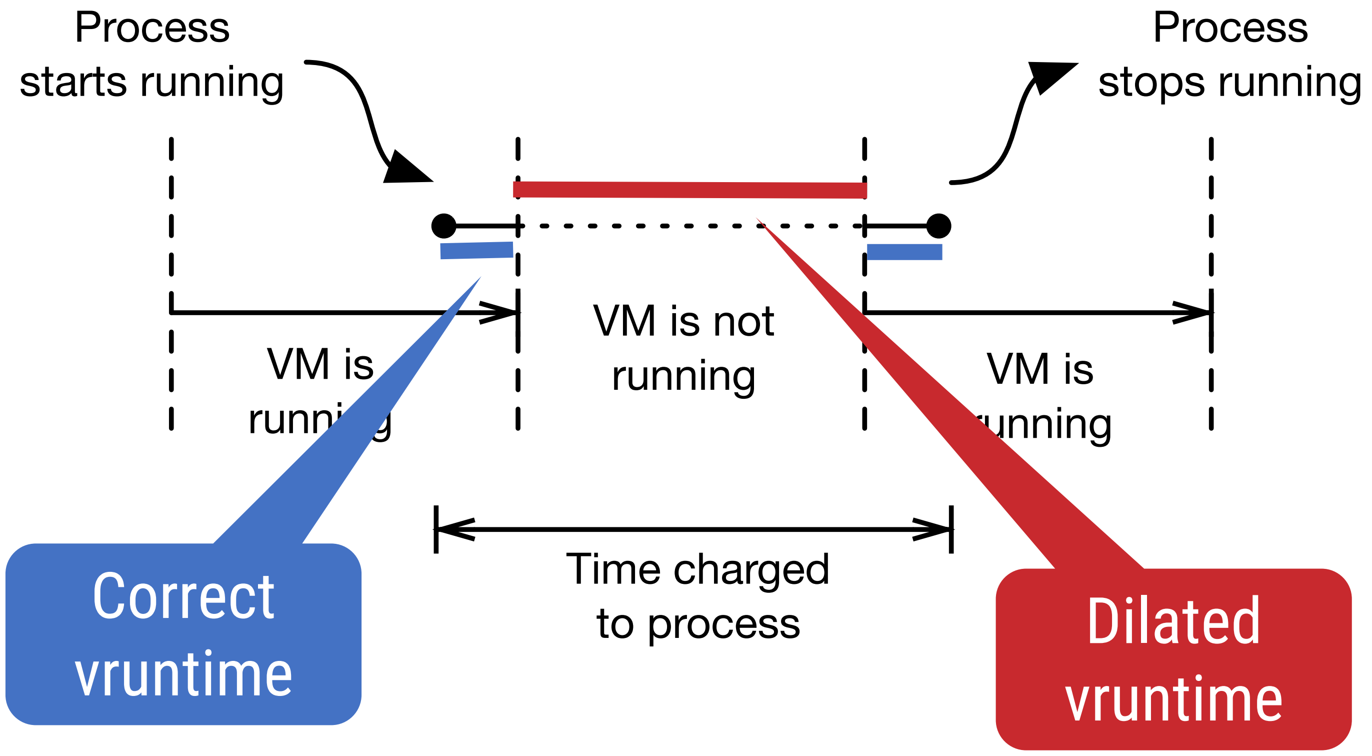




Why I/O prioritization is not effective?

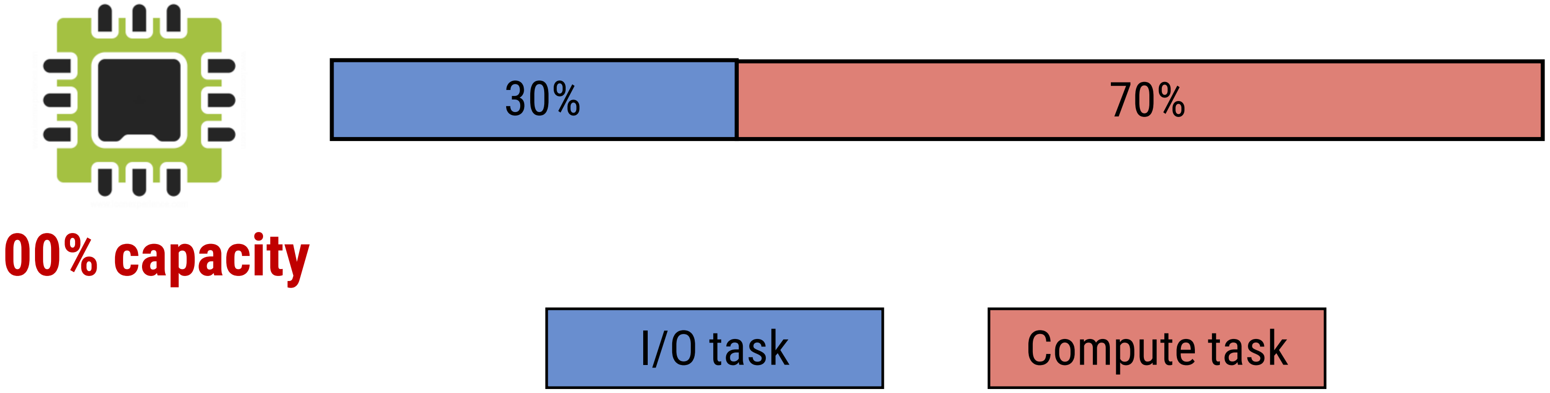
Short-term Priority Inversion

- Time discontinuity may cause inaccurate CPU accounting in the guest OS



vruntime dilation only affects I/O-bound tasks

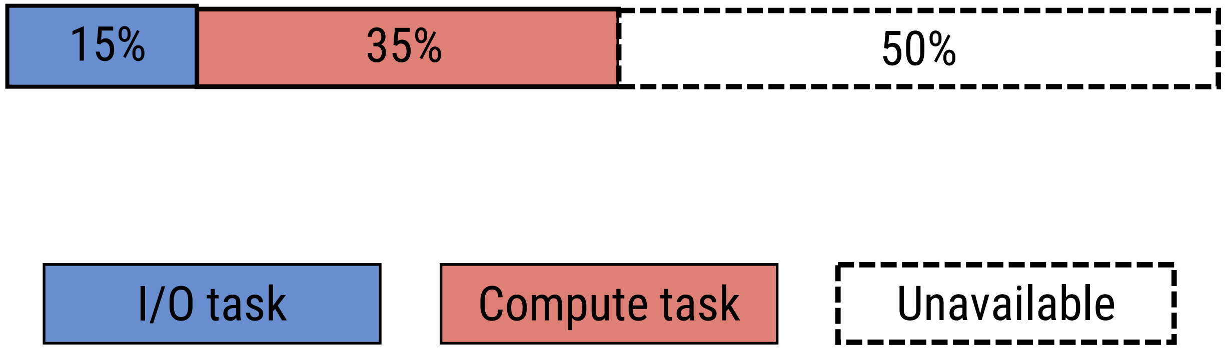
Long-term Priority Inversion



I/O task’s 30% CPU demand is fully satisfied.
Thus, I/O performance is not degraded

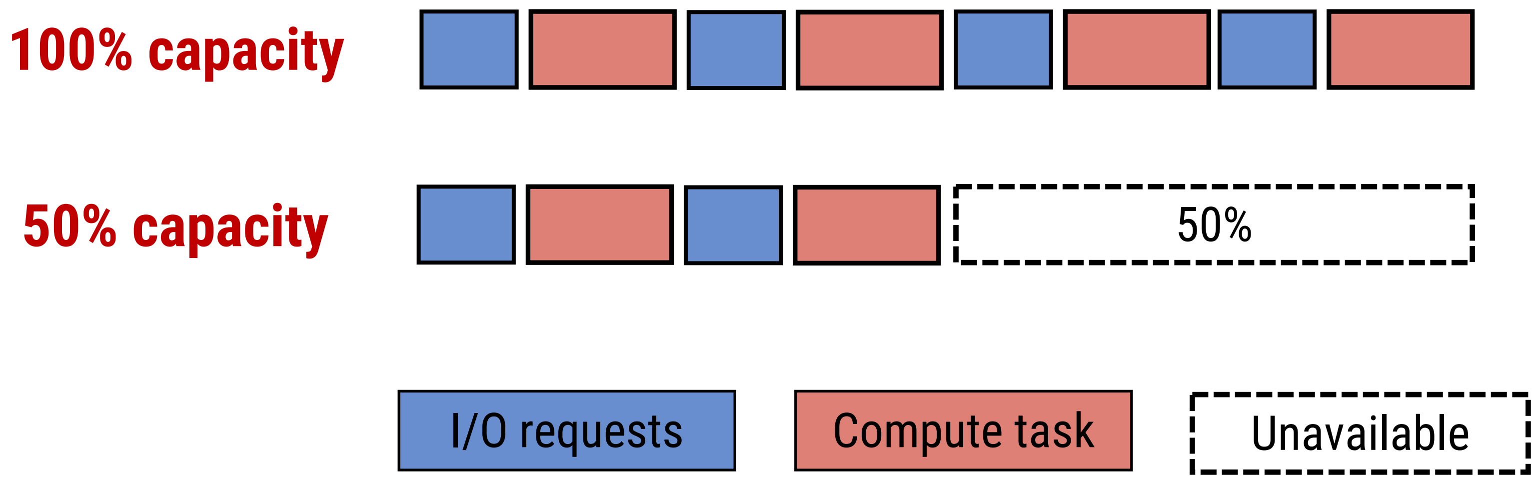
Long-term Priority Inversion

50% capacity



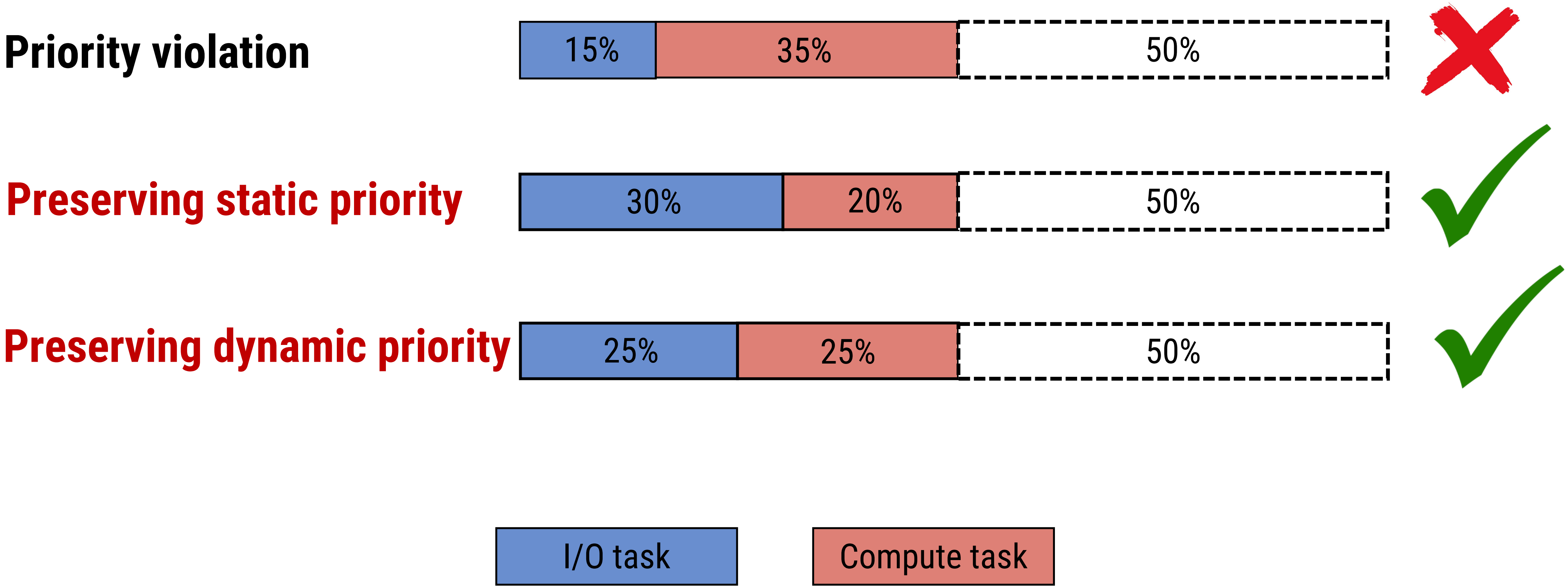
The CPU allocation to the I/O task is significantly reduced

Long-term Priority Inversion



Work-conserving scheduling in the guest OS allows compute-bound tasks to use the CPU that can be used by I/O-bound tasks in the future under constrained resource allocation

Preserving Long-term Priority

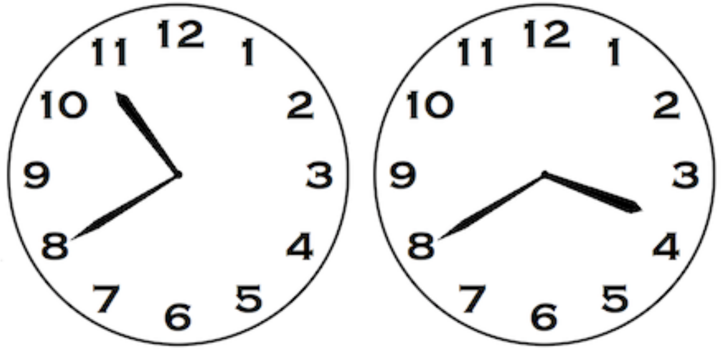


xBalloon

- Two clocks
- A CPU balloon process
- A semi-work-conserving (SWC) scheduling algorithm

Two Clocks

- Maintain two clocks in the guest OS



✓**Global clock** - synchronized with wall-clock time

✓**Virtual clock** - only ticks when the VM is running. In-guest scheduling uses this clock [KVM]

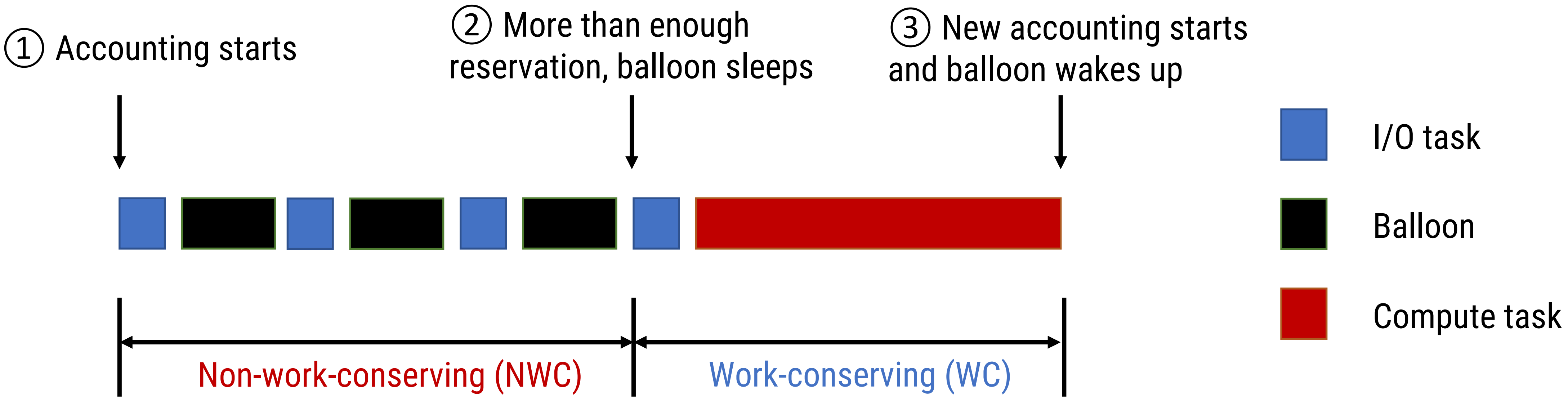
Guest OS is made aware of time discontinuity to address short-term priority inversion

The CPU Balloon Process

- Inspired by memory ballooning
  - ✓ The size of the memory balloon represents the memory taken away from the VM
- The runtime of the CPU balloon process represents the amount of time guest OS reserves for the future

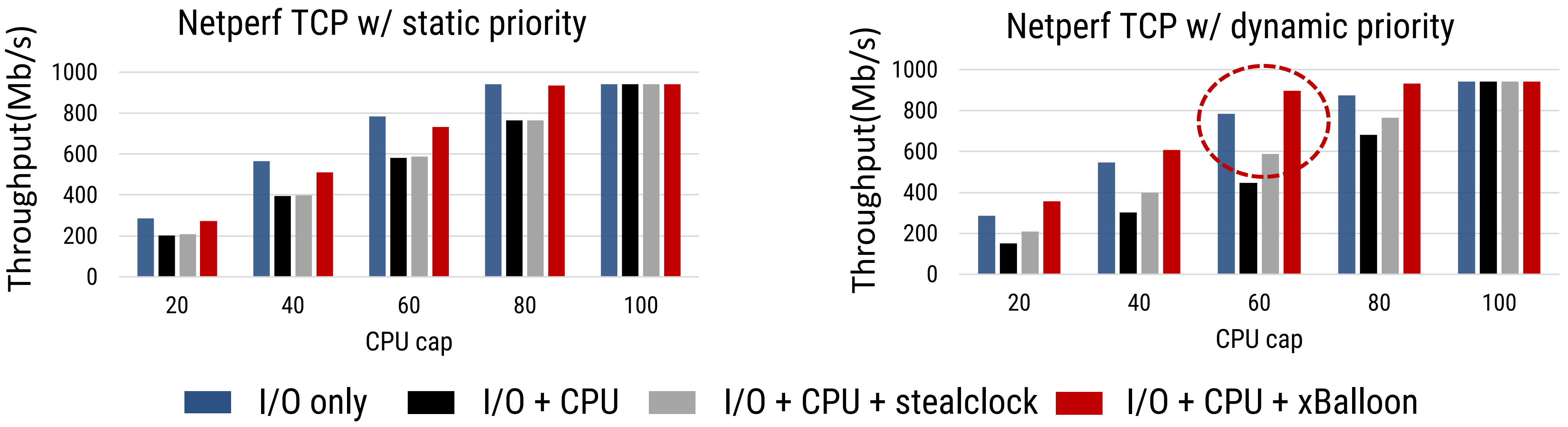
Whenever the balloon process runs, it suspends the VM.
The VM is woken up upon receiving an I/O request

Semi-Work-Conserving Scheduling



NWC: balloon is running, reserving CPU for future
WC: balloon is suspended, all tasks are free to run

Improving I/O performance

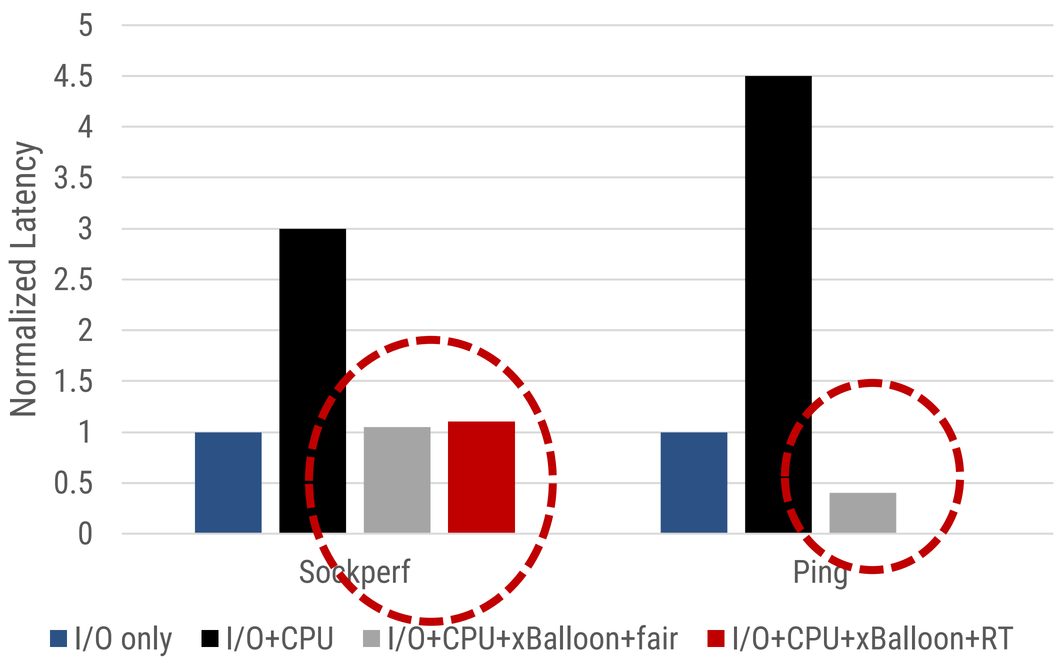


xBaloon achieves performance close to the I/O only case

Most performance gain is due to the prevention of long-term priority inversion

Results on Amazon EC2

- A brute-force implementation of xBalloon on an m3.medium instance
  - ✓ SCHEDOP_block to suspend VM
  - ✓ No semi-work-conserving mode due to no access to the hypervisor



Wiser scheduling decisions in the guest OS can greatly improve I/O latency in public clouds

Conclusion

Motivation

Time discontinuity due to CPU multiplexing or capping can render I/O prioritization in the guest ineffective, leading to I/O performance loss and unpredictability.

xBalloon

A lightweight approach to preserving static and dynamic priorities between I/O- and compute-bound tasks under discontinuity.

Evaluation

Results in a local environment and Amazon EC2 show that xBalloon improves I/O performance in both network throughput and tail latency.

Thank you !

Questions?

Backup Slides …

Q: What if workload is CPU-intensive and I/O-intensive at the same time?

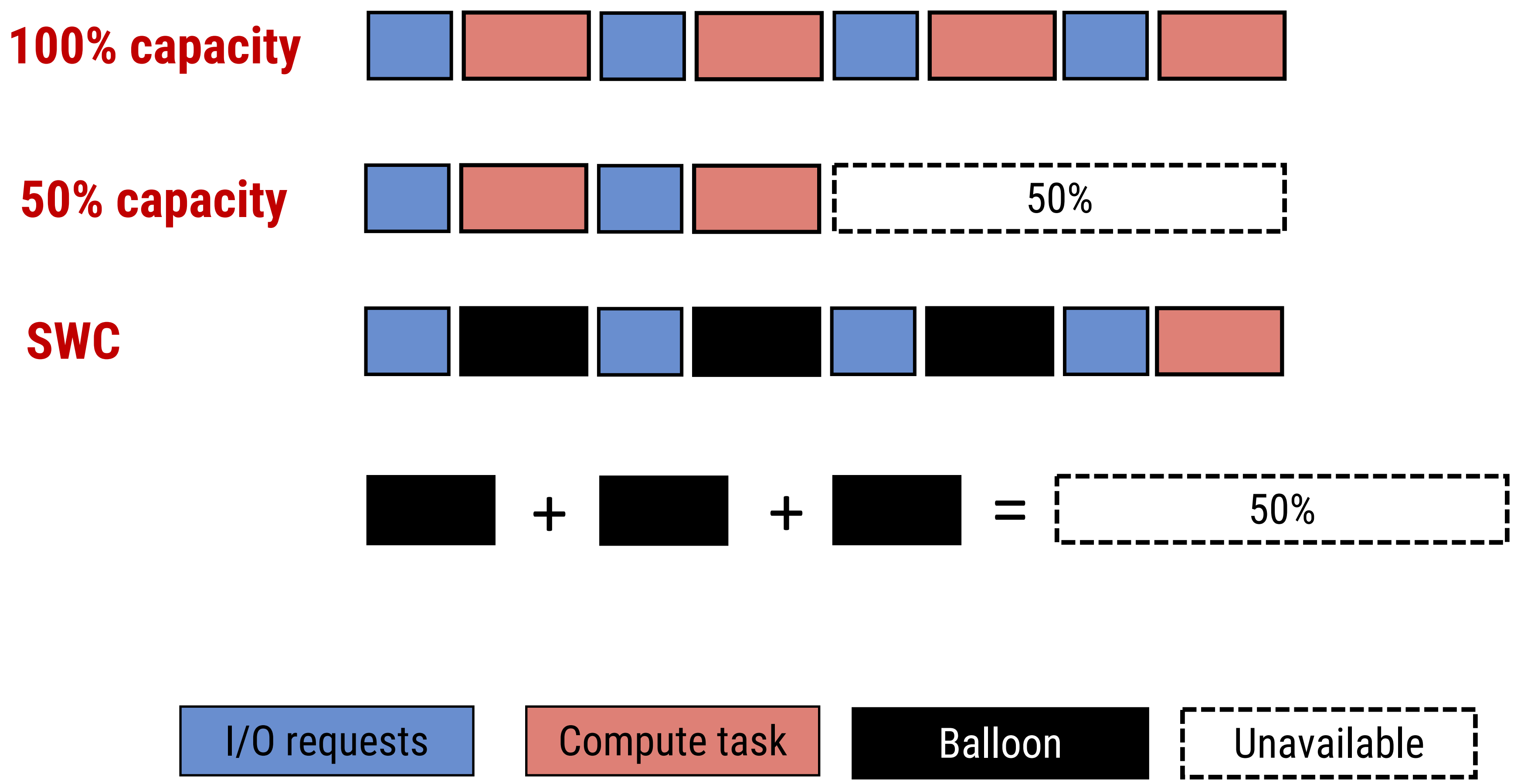
A: If the I/O workload contains non-negligible CPU processing, it may not be wise to place it with another CPU-bound workload as the aggregate CPU demand of the two workloads could be well above the CPU capacity.

The CPU utilization of some workloads, such as memcached, will increase as more user requests sent to the server side. However, they can still benefit from xBalloon if user activities drop and become less CPU-bound and more I/O-bound.

Q: The overhead of xBalloon?

A: xBalloon’s overhead includes the time spent in user space, in the guest kernel, and inside Xen. As the balloon is woken up by I/O events, the frequency of xBalloon invocation depends on the intensity of the I/O workload. Our results show that in most cases xBalloon’s overhead is negligible.

How SWC Scheduling Works



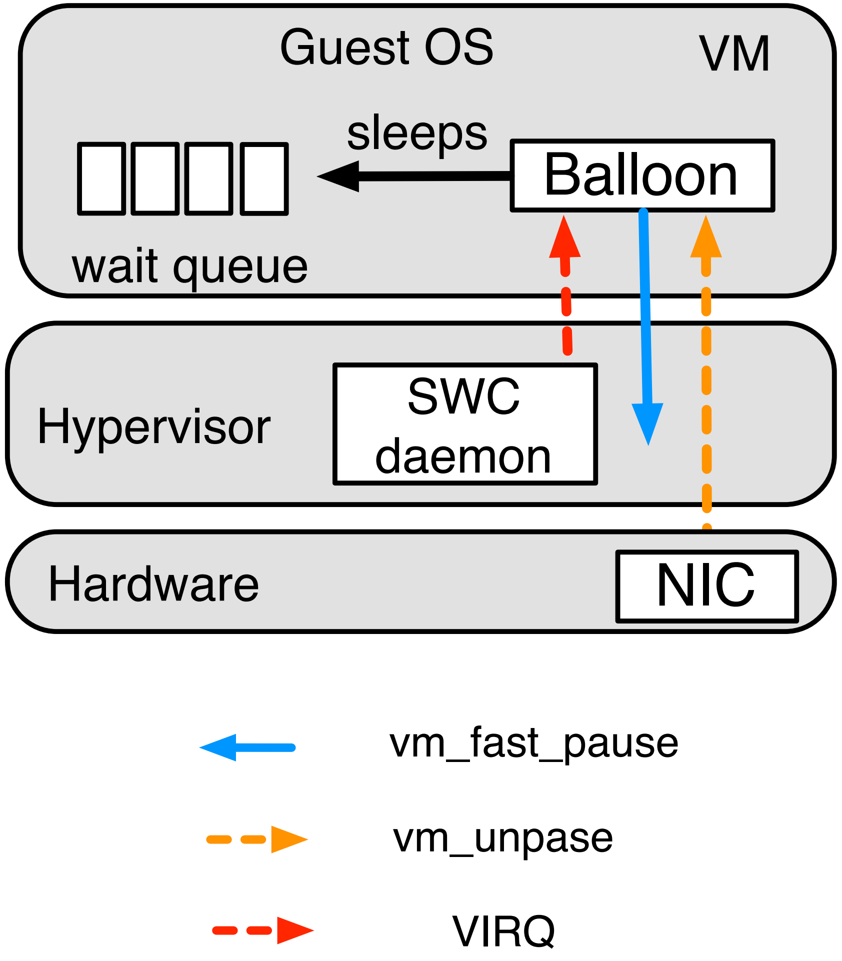
The Balloon Process

```
void balloon (void)
{
        int ret;
        while(1)
        {ret = syscall(__NR_sched_balloon);
        if (ret)sched_yield();
}
}
```

Priority Inversions in Containers

- As long as the I/O task and the compute task share limited resource allocations, e.g., belonging to the same `cgroup`, long-term priority inversion could happen

Putting It Together



Simplicity:
balloon easy to disable

Minimal intrusiveness:
one tweak in CFS and one new mechanism in Xen

Autonomy:
Guest OS has total control, no optimization at Xen

Evaluation

Hardware

DELL PowerEdge T420 servers, two six-core Intel Xeon E5-2410 1.9GHz processors, 32GB memory, one Gigabit Network card and a 1TB 7200RPM SATA hard disk.

Software

Linux 3.18.21 as the guest and dom0 OS, and Xen 4.5.0 as the hypervisor.

Benchmark: netperf, sockperf, memcached, Nginx, MySQL, etc.

Configuration

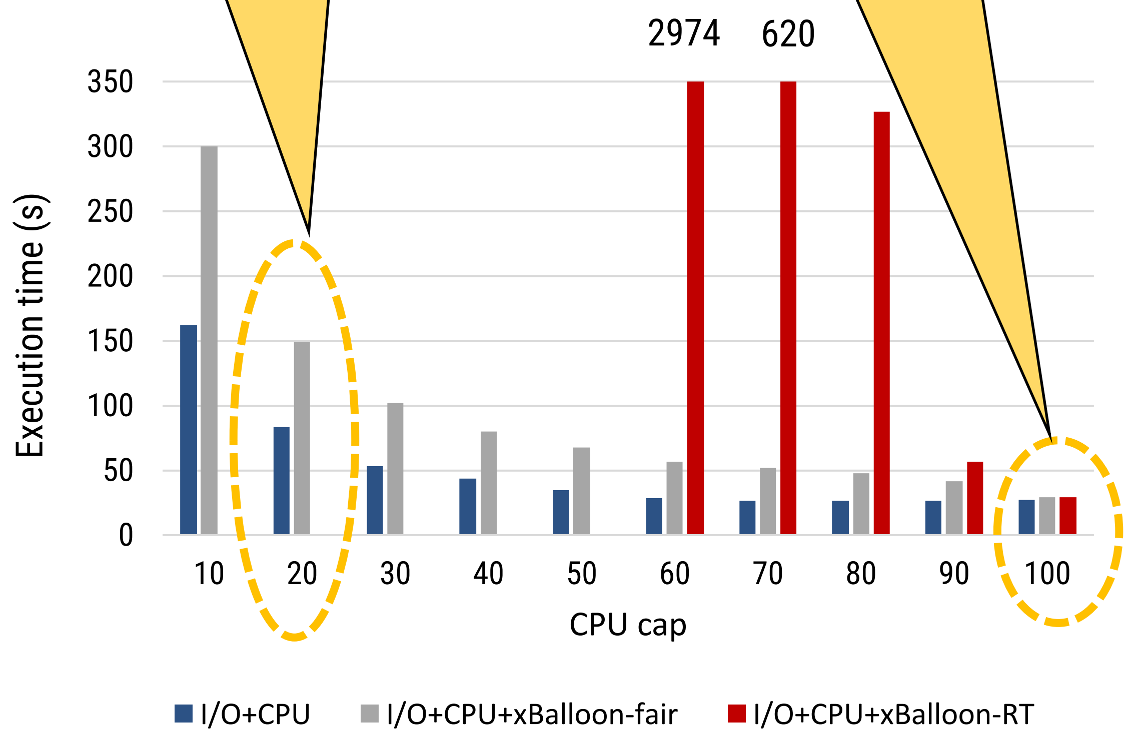
VMs were configured with one vCPU (four vCPUs) and 4GB memory.

Simulate the time discontinuity through CPU capping and CPU sharing.

Impact on Compute tasks

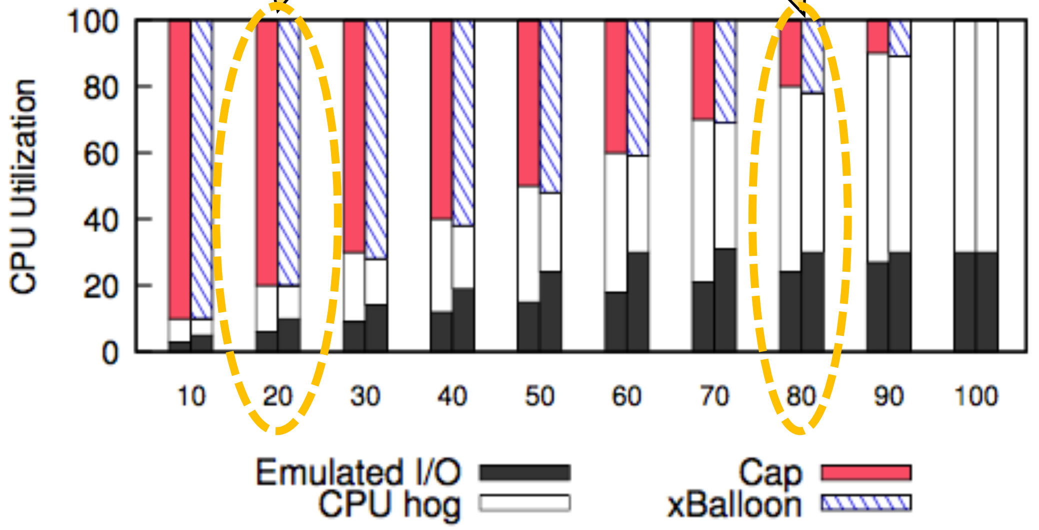
xBalloon deprives the compute task of CPU when the cap is lower than the I/O demand

xBalloon allows the compute task to use slack CPU resources when CPU cap increases

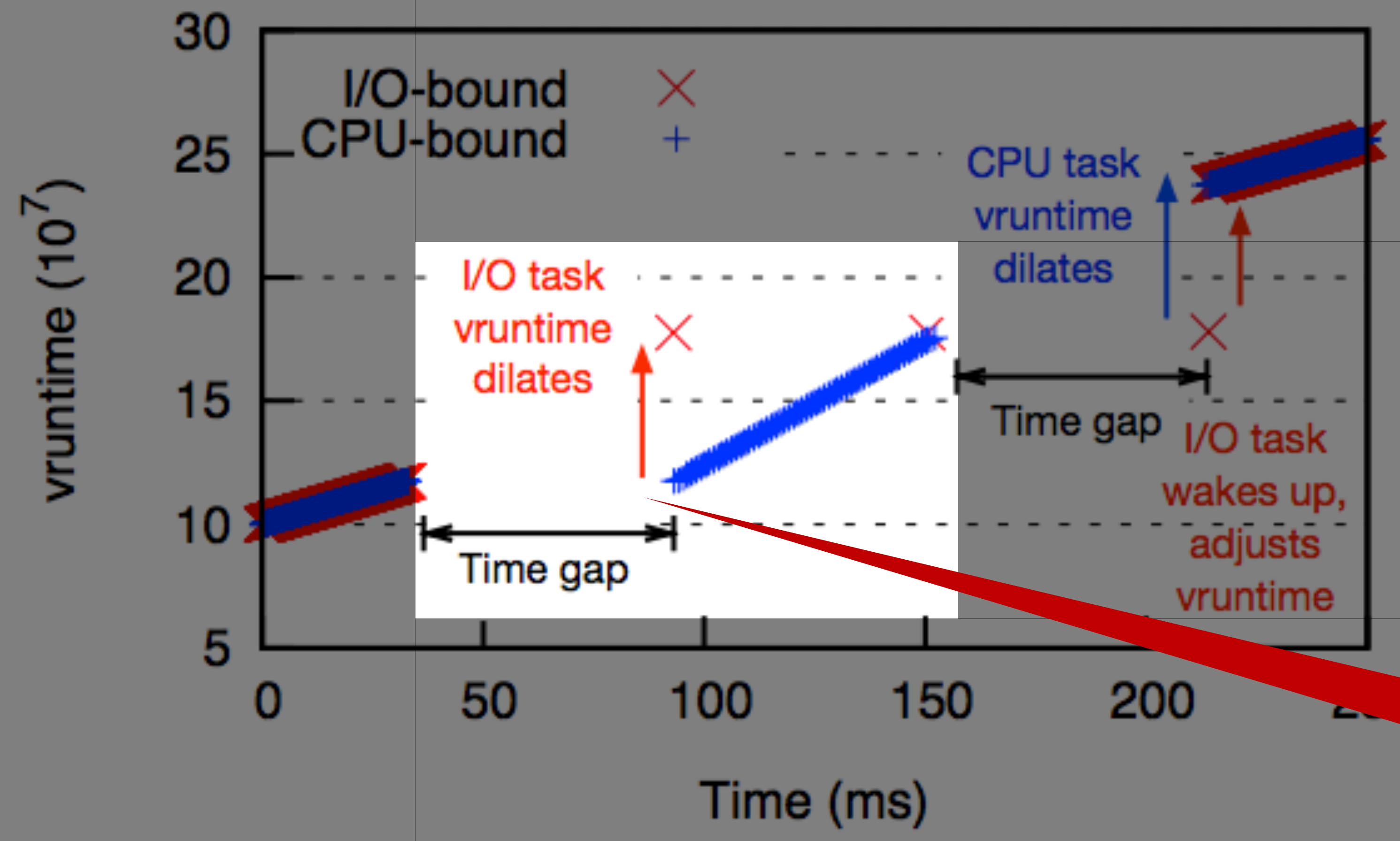


When CPU cap is 20%, I/O task : Compute task = 10:10

When CPU cap is 80%, I/O task : Compute task = 30:50

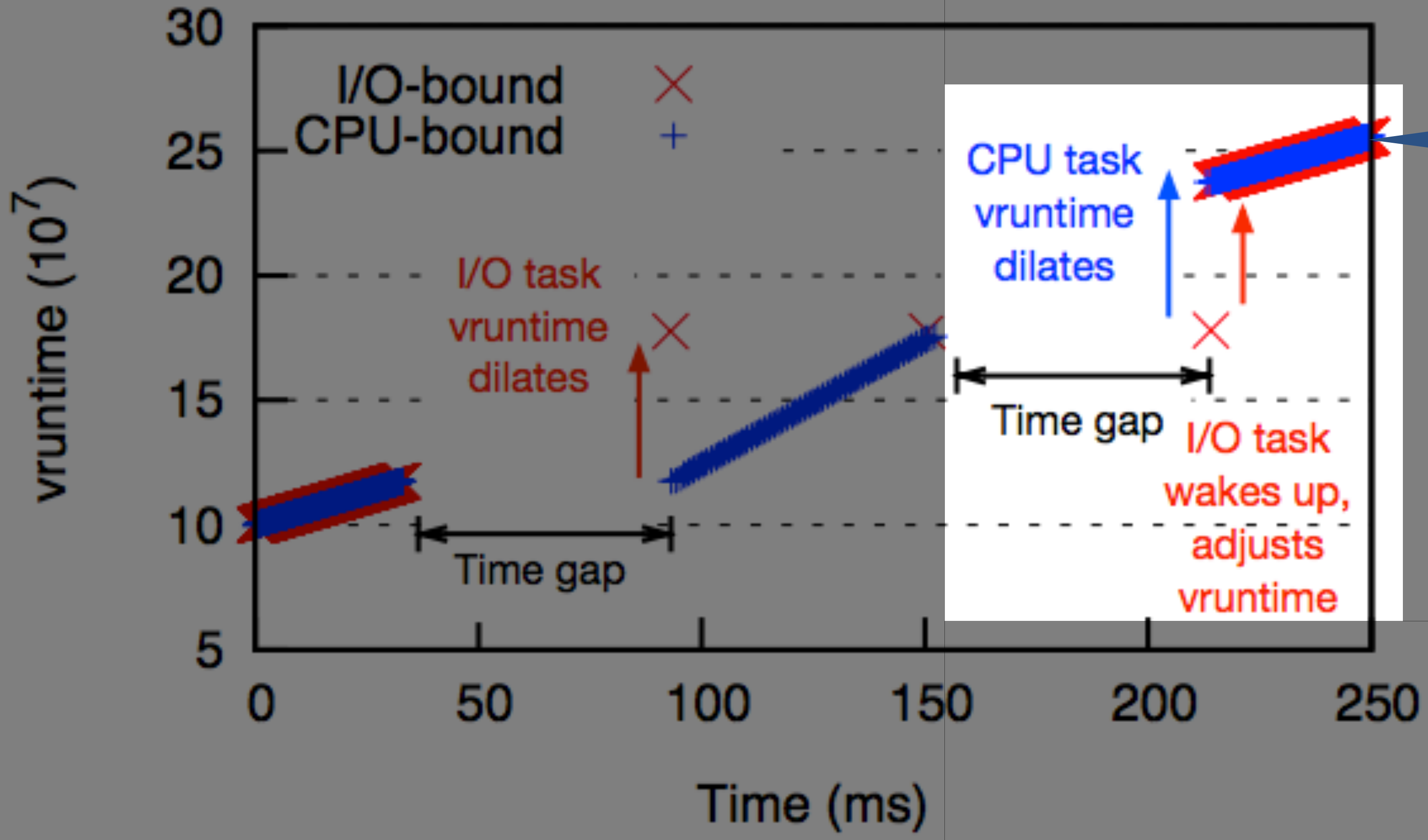


SPI only hurts I/O workload



I/O-intensive app stops running for a long time when its vruntime dilates

SPI only hurts I/O workload

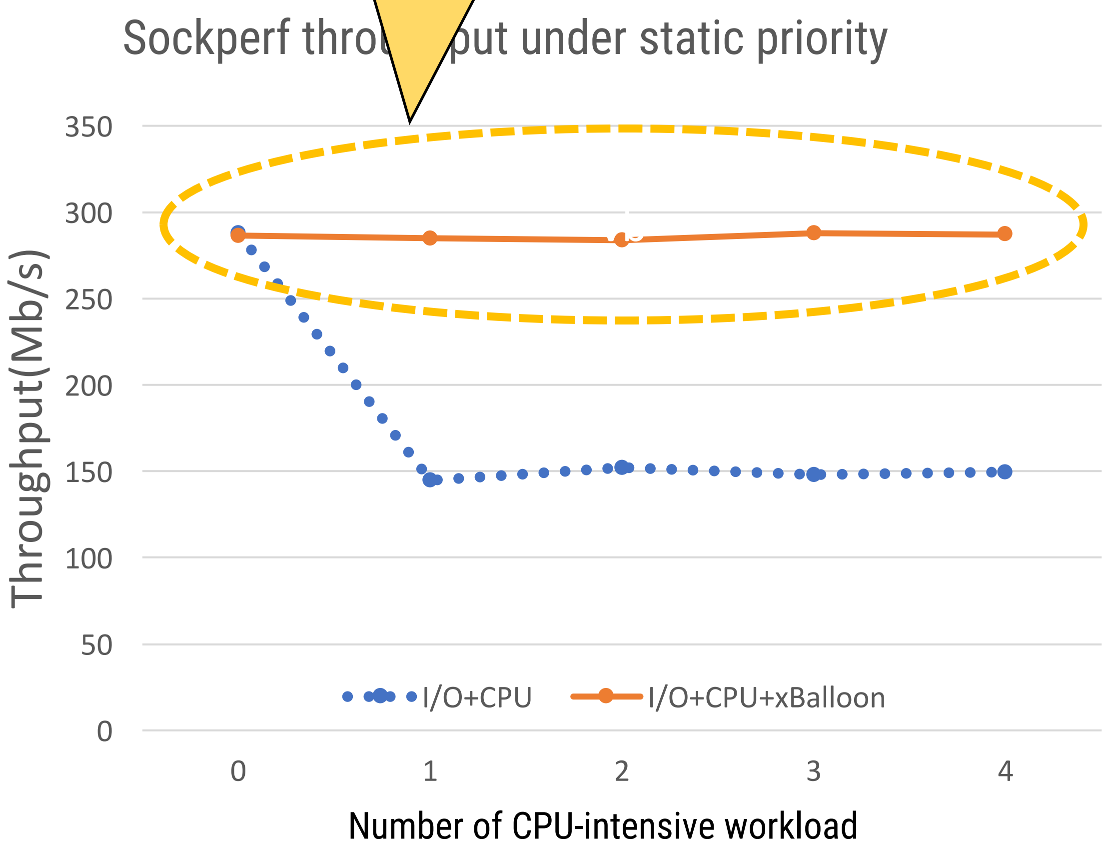


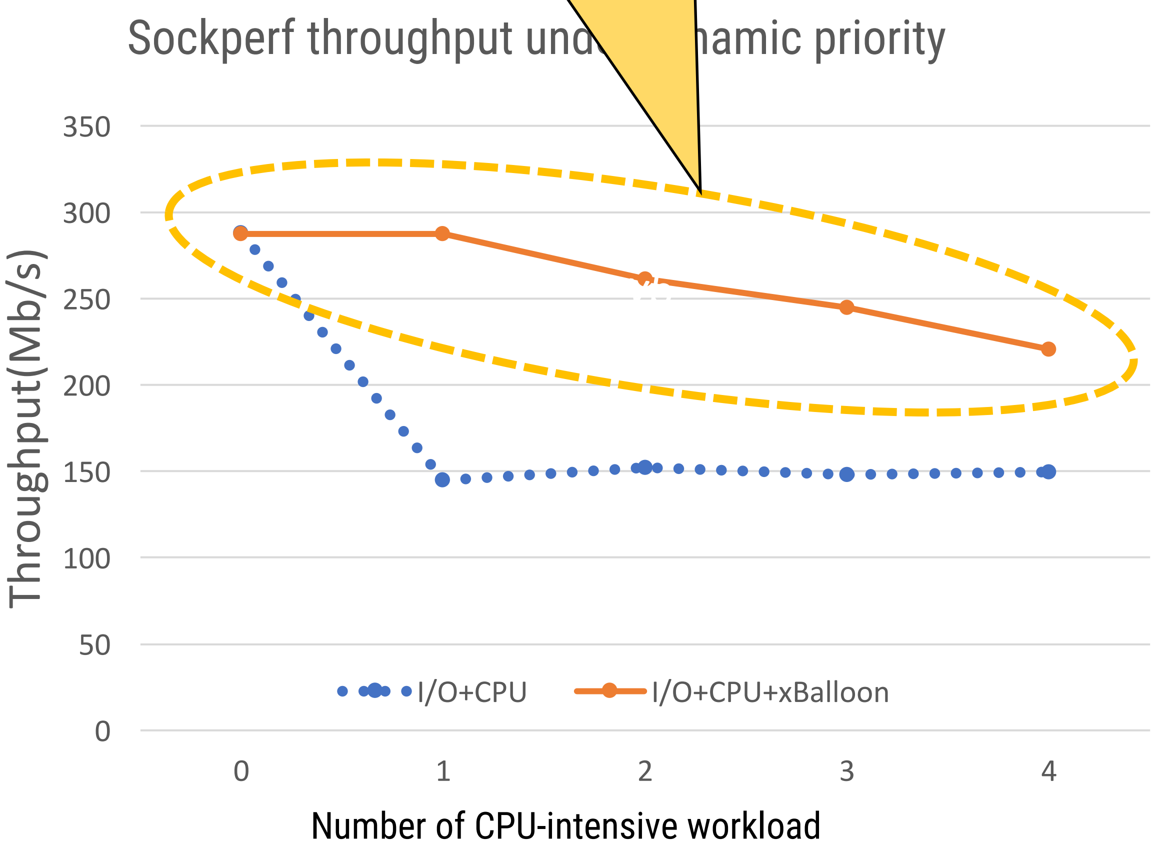
CPU-intensive app keeps running for a long time when its vruntime dilates

e the I/O performa

The throughput keeps the same when number of CPU-intensive apps increases.

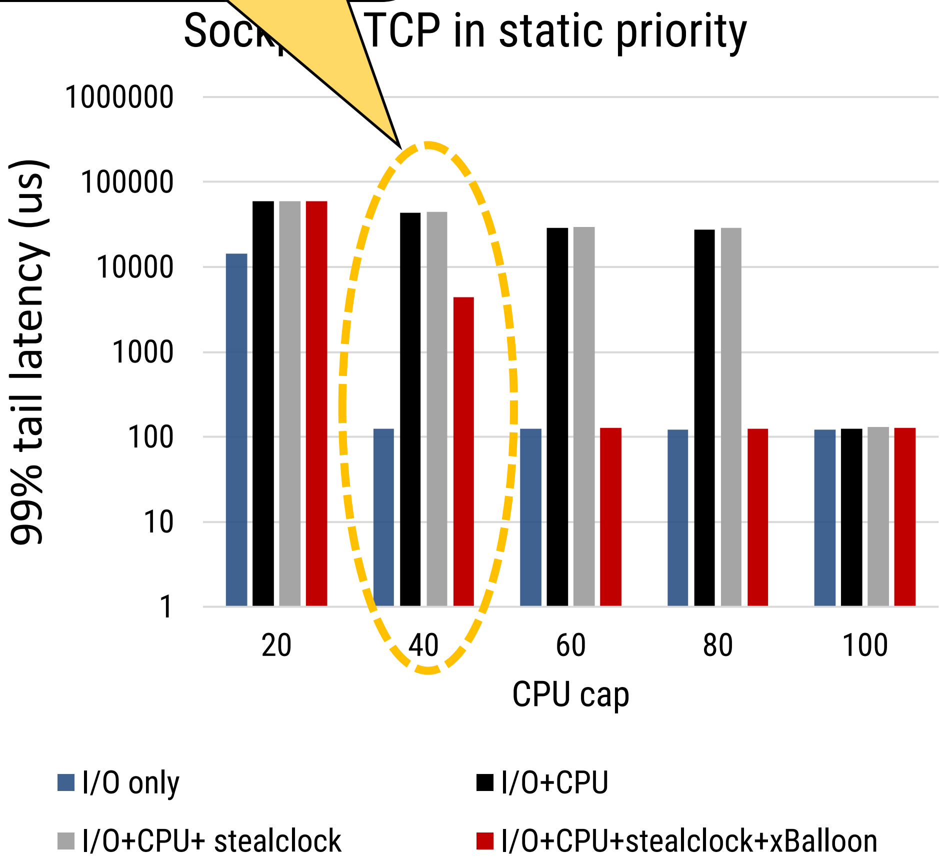
The throughput decreases as the number of CPU-intensive apps increases.



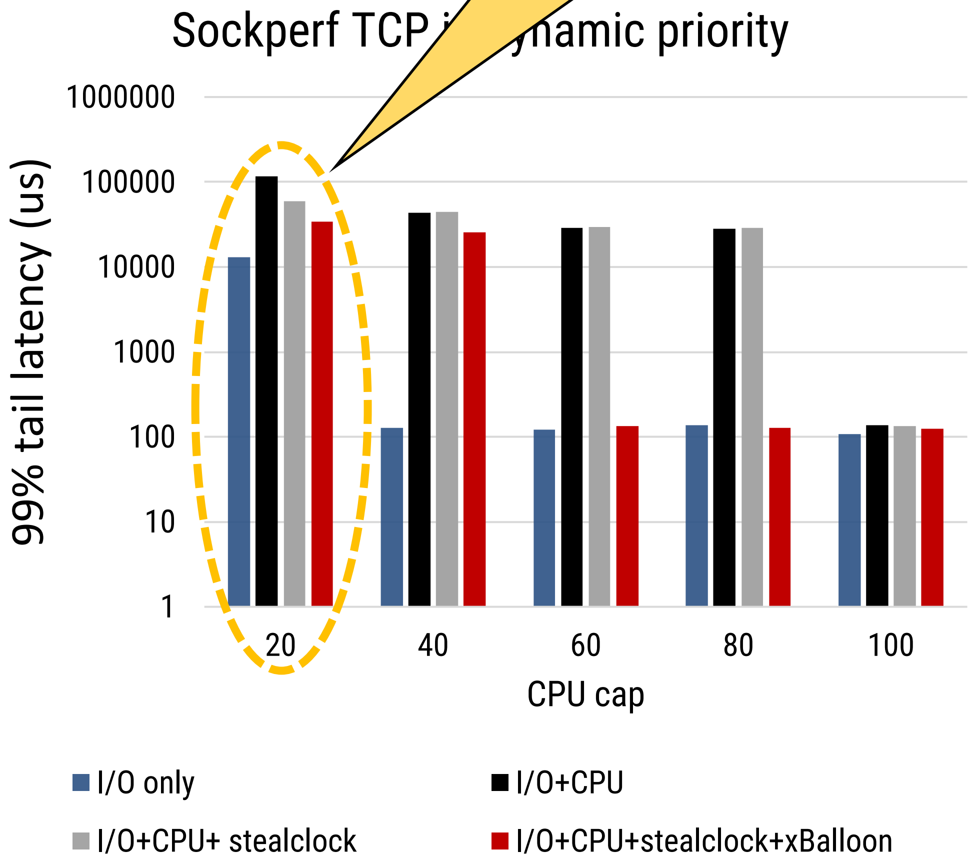


Improve the I/O performance

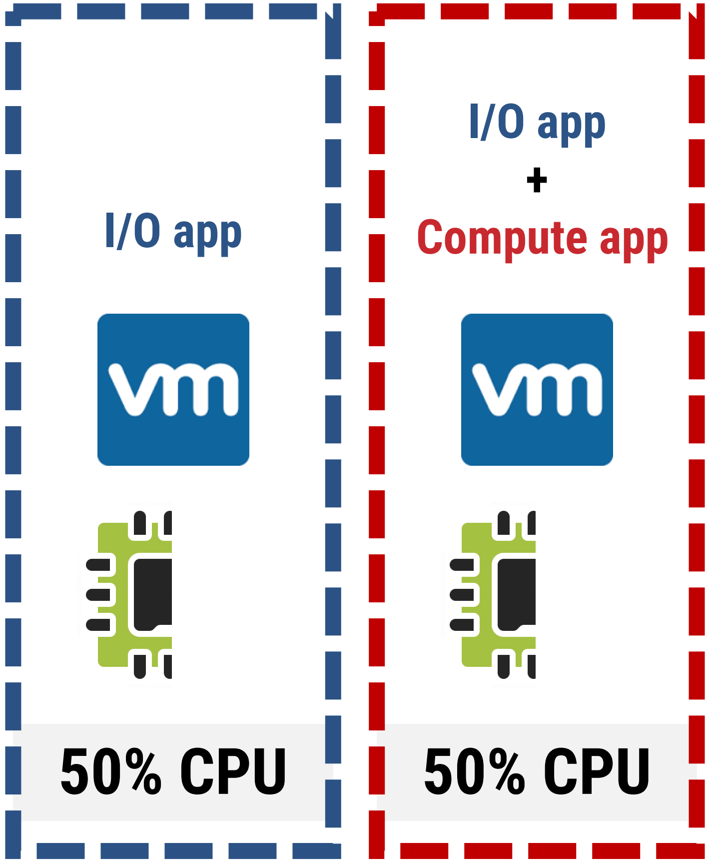
Stealclock is not effective to I/O task in static priority while xBalloon is.

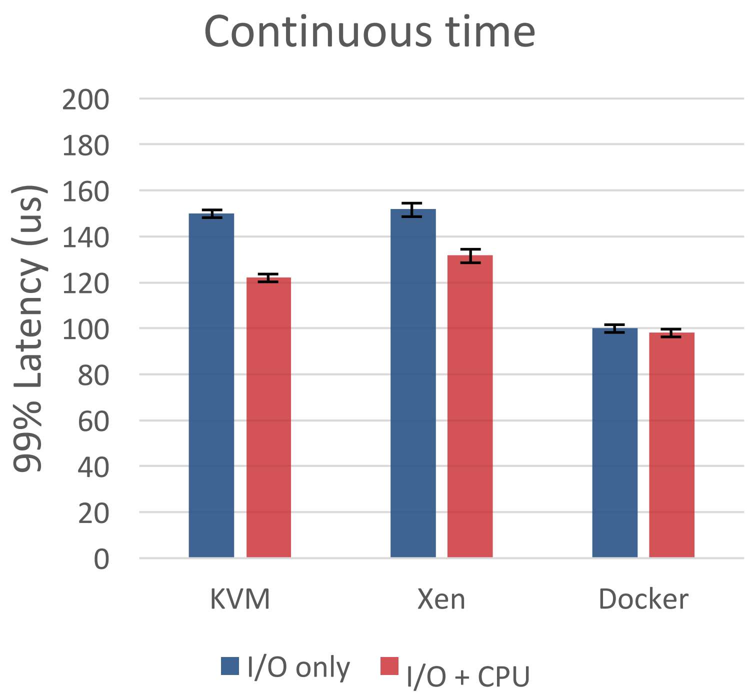


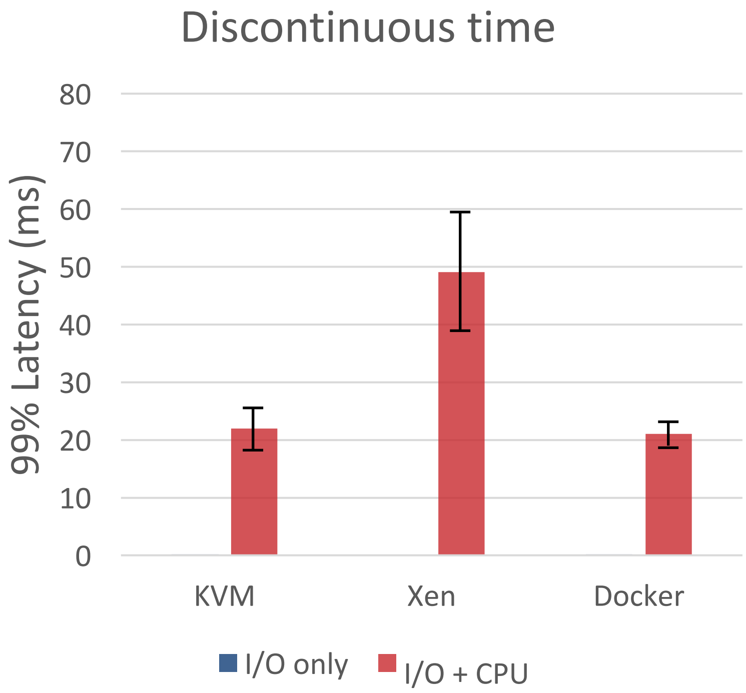
xBALLOON improves the 99% tail latency by 71%.



Violation of I/O Prioritization







- **Virtualization and multi-tenancy do not preserve the property of I/O prioritization**
- **I/O tasks are affected by compute-bound tasks with significantly degraded and wildly unpredictable performance**