Adaptive Resource Views for Containers

Hang Huang[1], Jia Rao[2], Song Wu[1], Hai Jin[1], Kun Suo[2], Xiaofeng Wu[2]
[1] Huazhong University of Science and Technology
[2] The University of Texas at Arlington

HUAZHONG UNIVERSITY
OF SCIENCE & TECHNOLOGY

UNIVERSITY OF
TEXAS
ARLINGTON

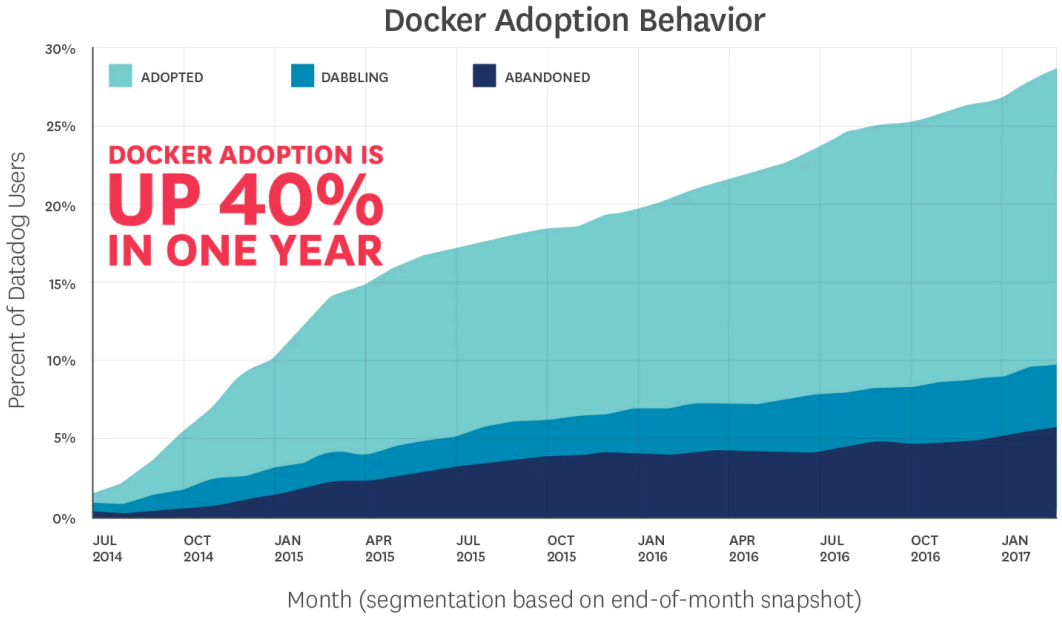
June 28 2019

Containers

- A lightweight alternative to virtual machines for application packaging
  - Ease of management
  - Fast and cross-platform deployment
  - On-demand elasticity
  - **Low overhead, high efficiency and density**

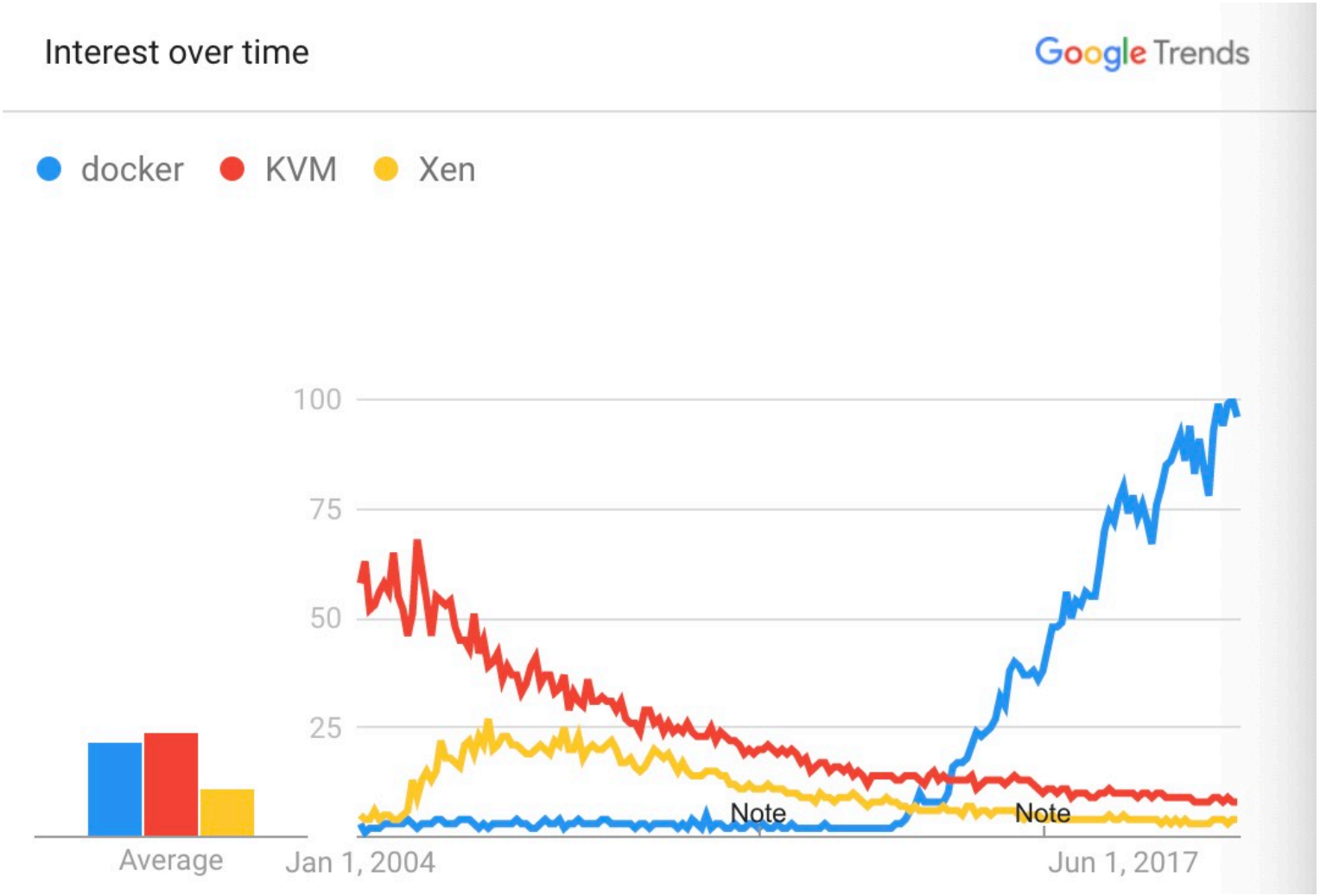
The Rise of Containers

- Increasingly and widely adopted in datacenters
  - Google Search launches about 7,000 containers every second

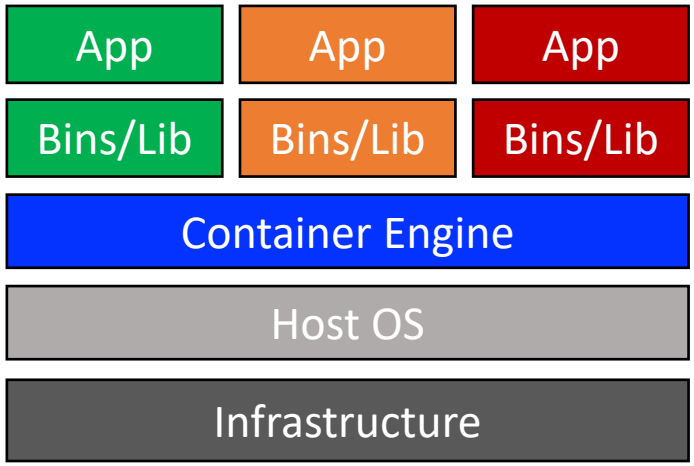


Source: Datadog

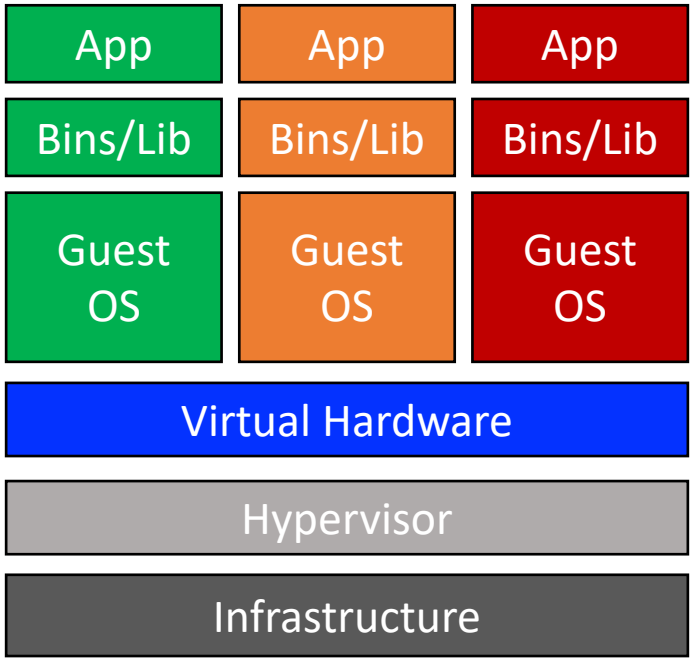
https://codefresh.io/docker-tutorial/migrating-to-docker/


Interest over time
Google Trends
docker
KVM
Xen
100
75
50
25
Note
Note
Average
Jan 1, 2004
Jun 1, 2017

Container vs. Virtual Machine



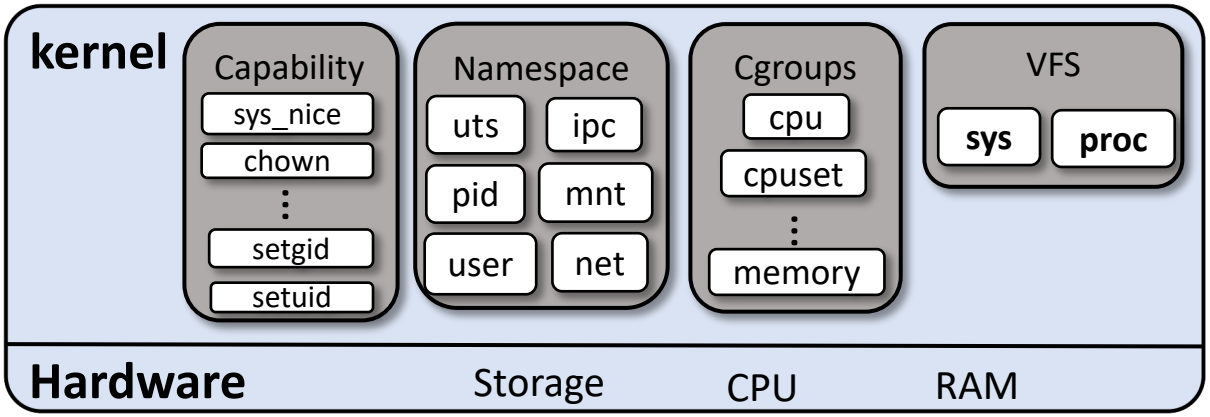
Container



Virtual machine

Shallower virtualization stack, less overhead while still providing application isolation

Docker Containers



Docker containers share the same OS kernel and are isolated through namespaces and control groups (cgroups)

Preliminaries - `cgroups`

- CPU isolation
  - Mask – specify on which CPUs a container can run
  - Limit – the maximum amount of CPU allocation
  - Share – the relative weight in CPU allocation
- Memory isolation
  - Hard limit – the upper bound on memory allocation, beyond which swapping is triggered
  - Soft limit – If system has free memory, containers can use more memory than soft limit. But if there is memory shortage, containers that are beyond their soft limit first reclaim memory

While weaker isolation between containers helps attain near-native performance, it creates an important ***semantic gap***

Application

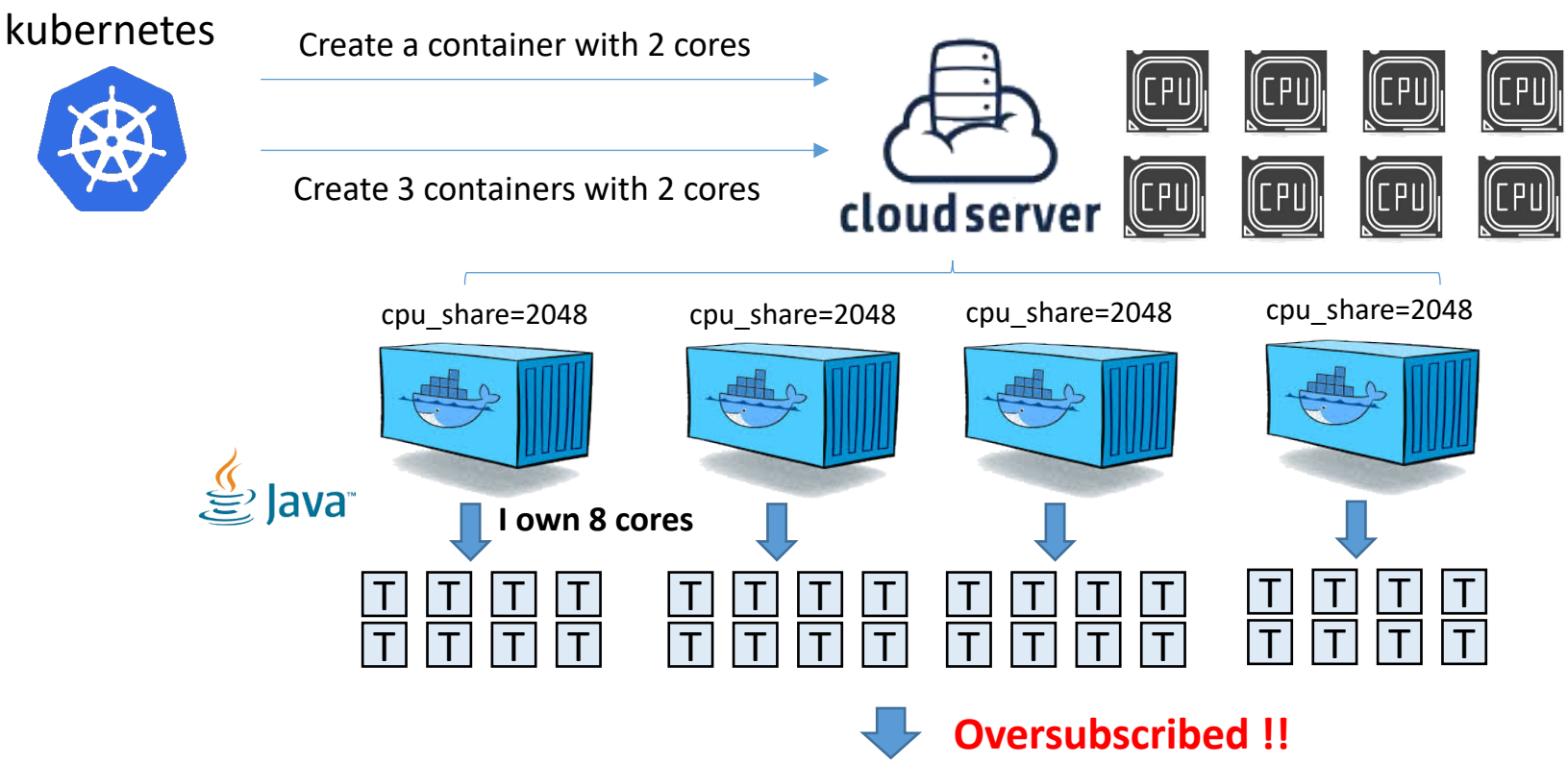
The absence of the virtual hardware abstraction allows containers to *observe* the *total available* resources on a host

Containers are only limited to *access* a *subset* of system resources due to resource capping or multiplexing

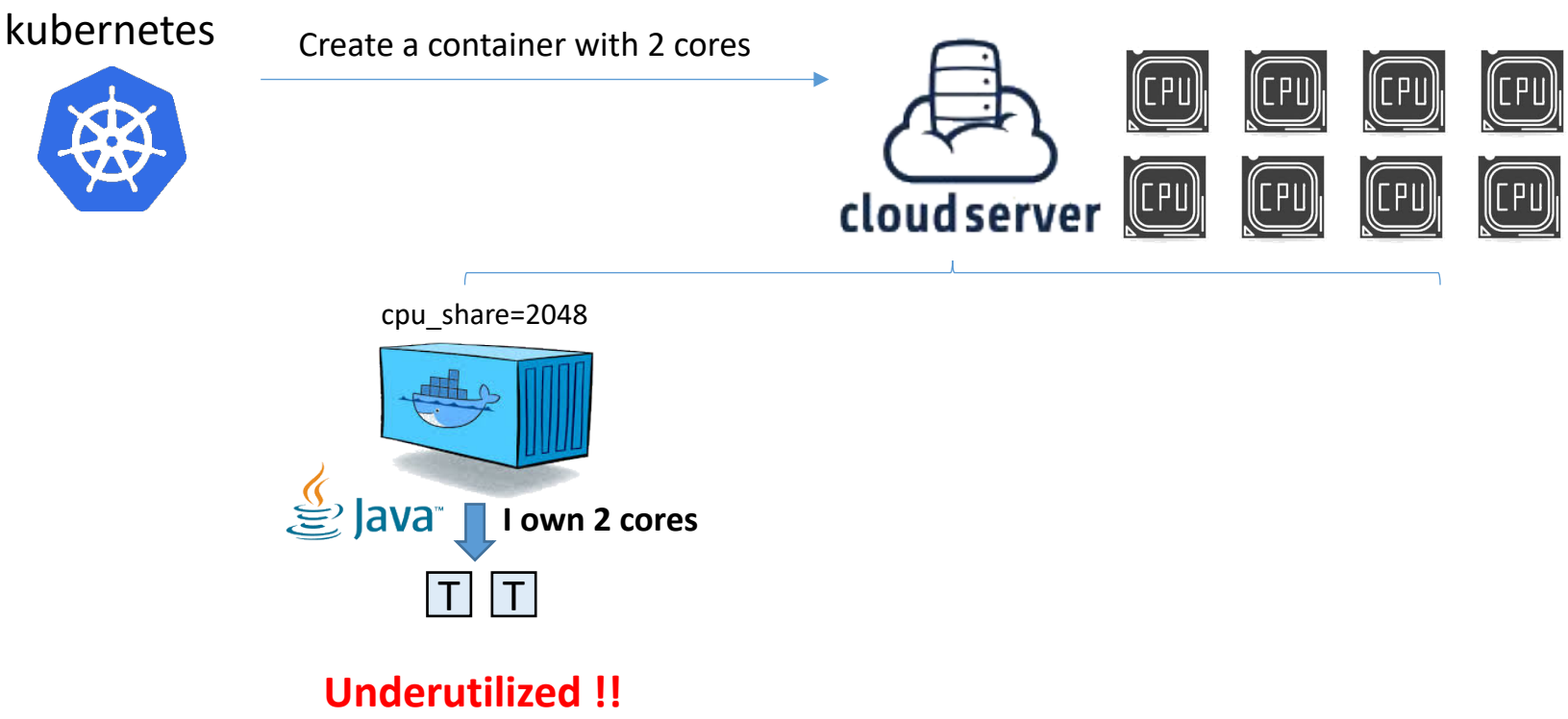
System admin

Many applications behave differently when containerized due to the gap between *resource availability* and *constraints*

Resource Availability vs. Constraints



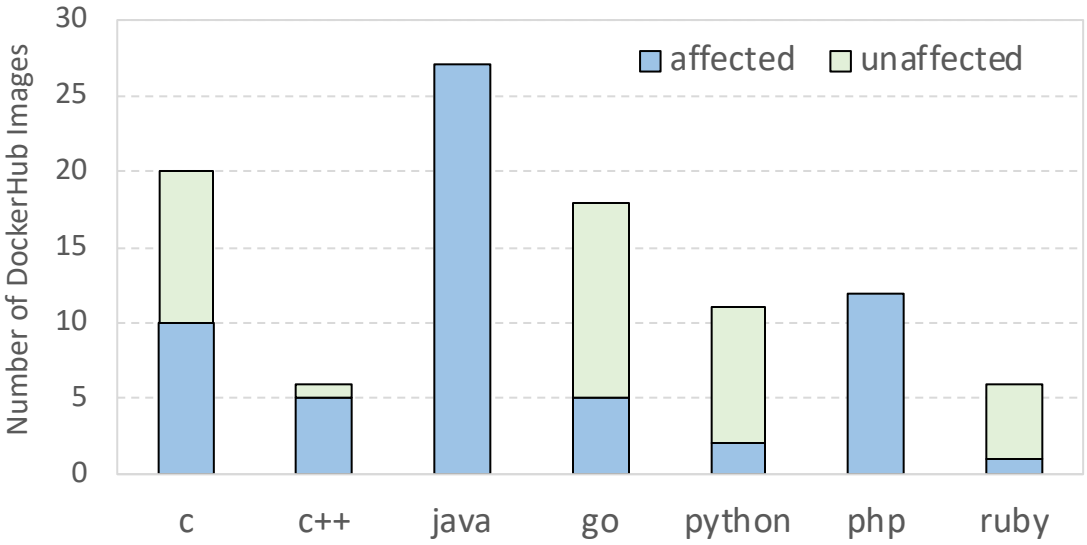
Resource Availability vs. Constraints



Resource Discovery in Native Linux

- Containerized applications detect resource availability through the OS kernel
  - Many runtime systems use the `sysconf` in the GNU C library (glibc) to probe system resource
  - `sysconf` queries `sysfs` to determine the number of online CPUs and available memory size

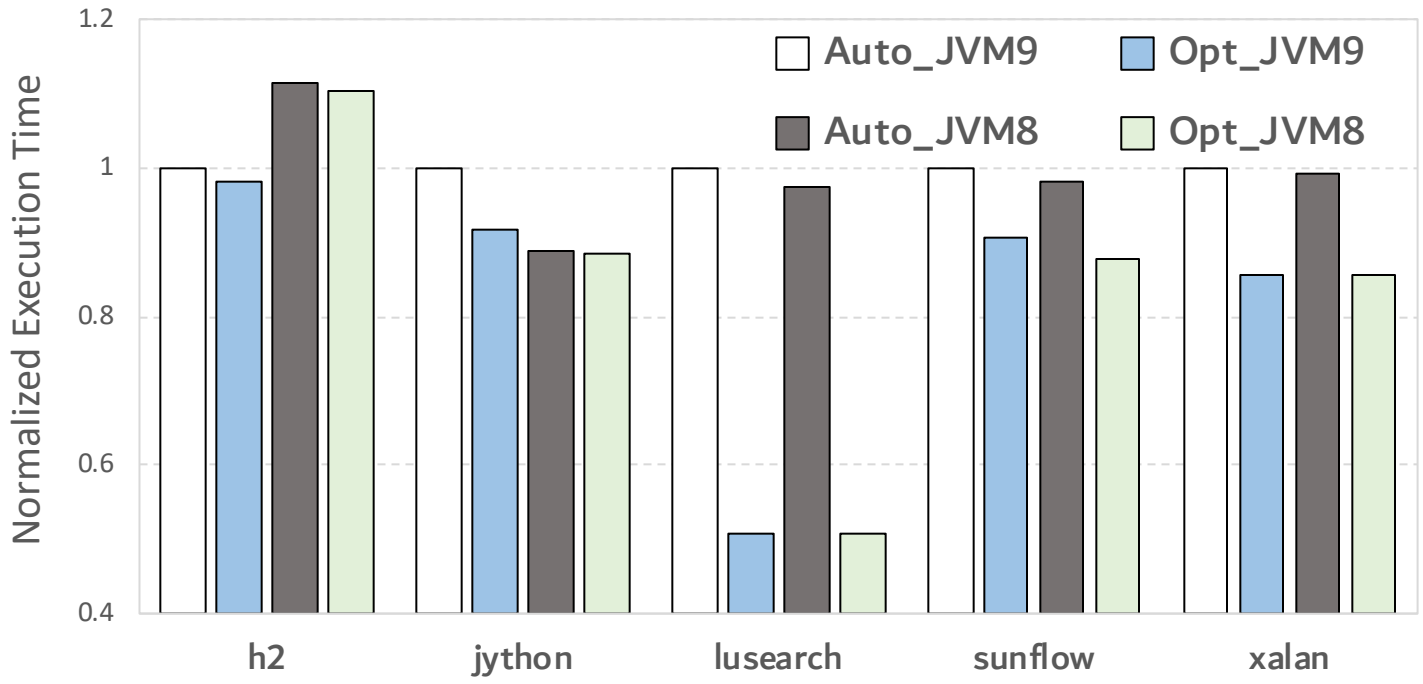
The Prevalence of the Issue



Analysis of the top 100 application images on DockerHub.

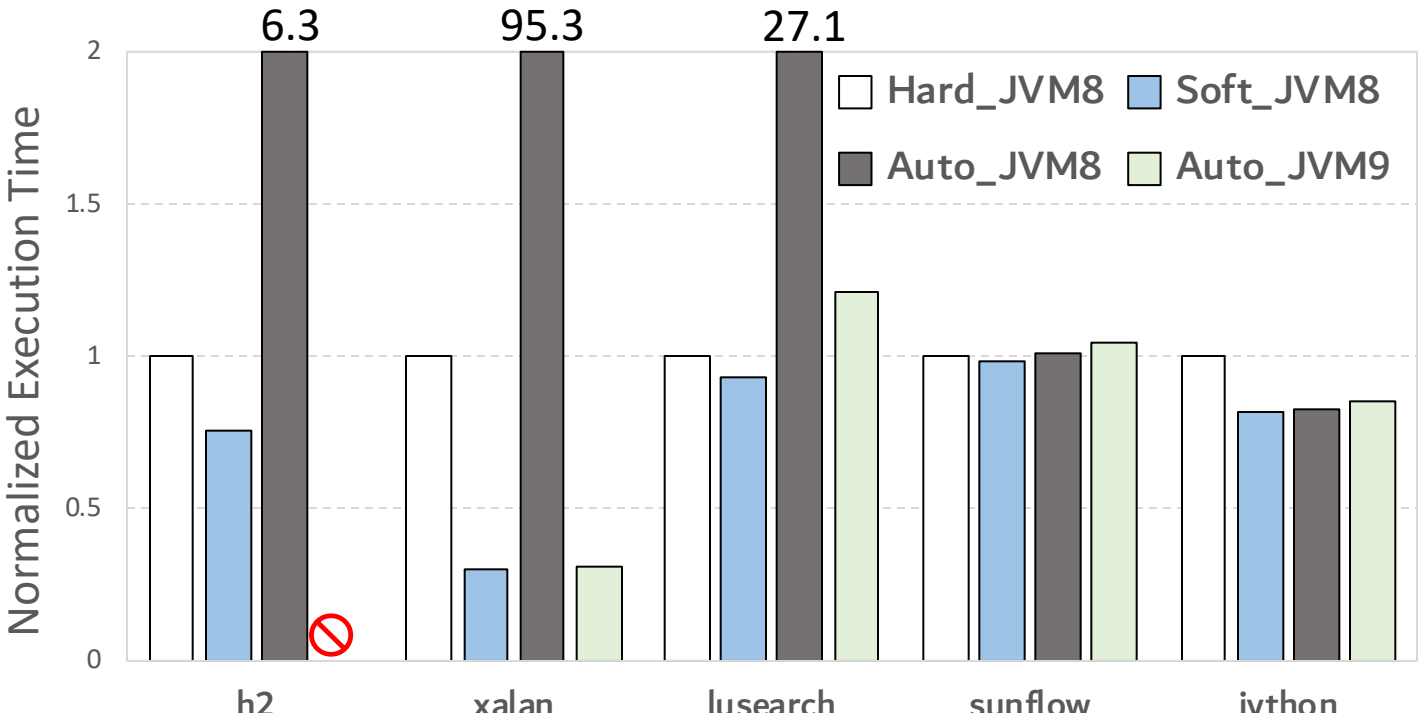
More than *60%* of the top 100 apps on DockerHub automatically configure app based on detected resource availability and are *affected* by the semantic gap

CPU Oversubscription



Oversubscribing CPU leads to considerable performance degradation

Memory Oversubscription



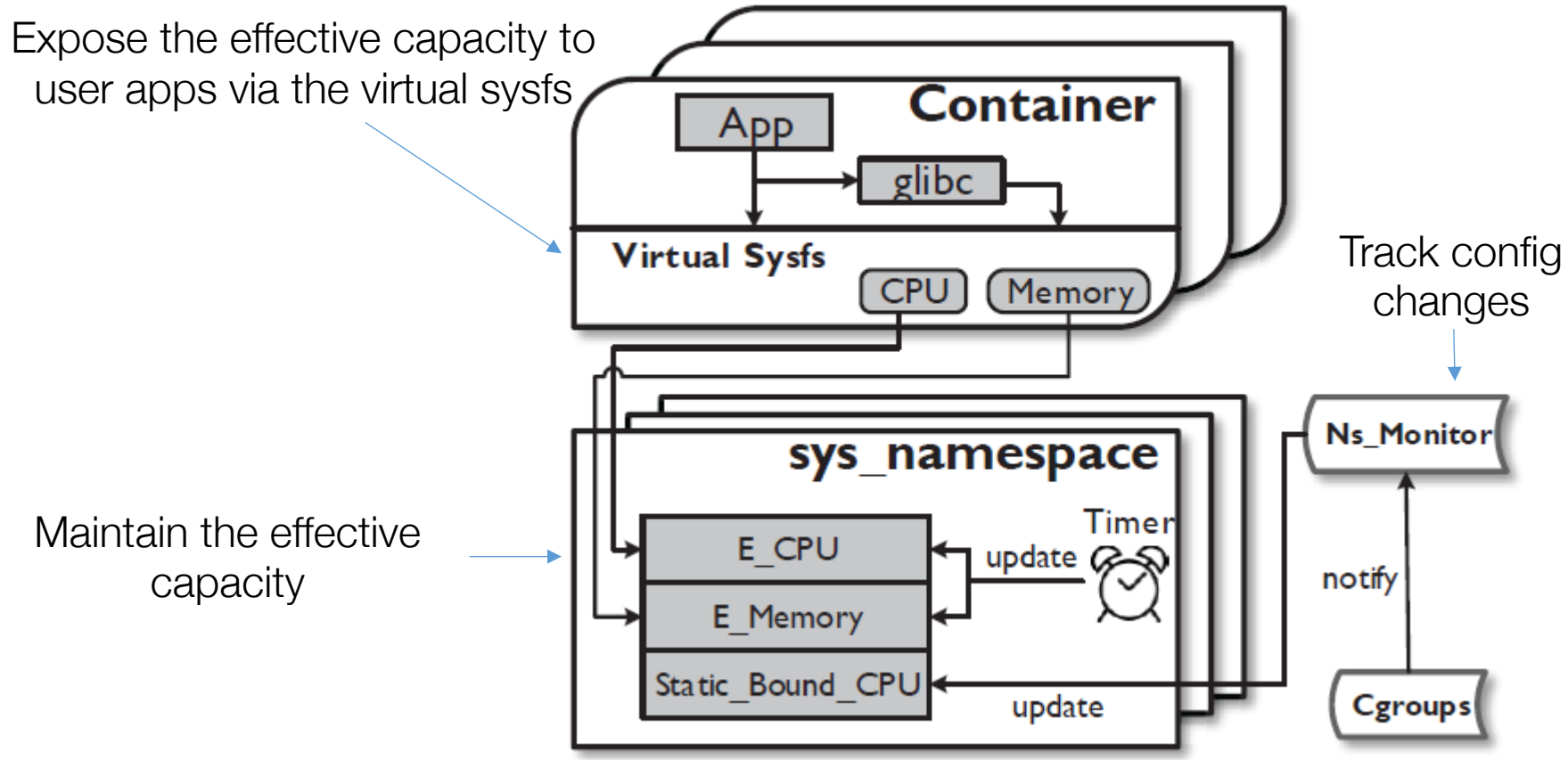
Oversubscribing memory can lead to program crashes or more severe degradation

Problem Statement

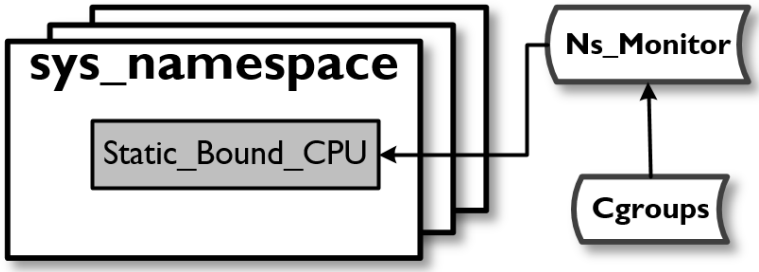
- *Objective*
  - Bridge the semantic gap between application-observed resource availability and the actual resource allocation
  - Build elastic apps that efficiently utilize variable resources
- *Challenges*
  - How to determine a container's real-time, actual resource allocation given various resource management schemes, e.g., limits, shares and resource affinity?
  - How to develop a general method to export such information to apps?
  - How can apps be made elastic, leveraging variable resource allocation?

Adaptive Per-container Resource View

System architecture that enables the resource view :
a new sys_namespace, a virtual sysfs, and a system-wide daemon Ns_Monitor.



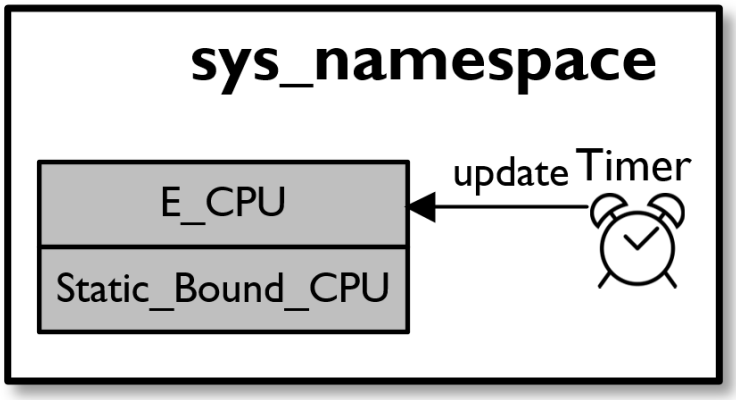
Effective CPU– Static Bounds



- Lower bound
  - Reservation, guaranteed CPU allocation
  - *min(*$\text{limit}, |M_i|, \left\lceil \frac{w_i}{\Sigma wj} \cdot |P| \right\rceil$)
- Upper bound
  - The maximum CPU usage
  - *min(*$\text{limit}, |\text{M}_\text{i}|$)

The lower and upper bounds do not change unless administrator reconfigures the container

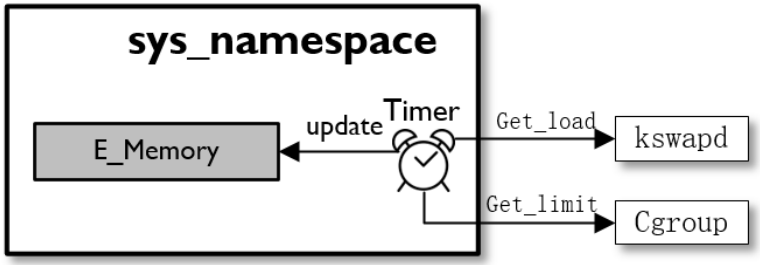
Effective CPU - Dynamic CPU Adjustment



- Periodic CPU adjustment
  - [lower bound, upper bound]
  - Effective CPU = # of CPU a container can fully utilize
  - Expressed as a discrete *CPU count*

If $\mathrm{p_{slack}} > 0$ then
  if $\frac{\mathrm{u_i}}{\mathrm{E\_CPU_i} \cdot \mathrm{t}} > 95\%$ and $\mathrm{E\_CPU_i} < \mathrm{UPPER\_CPU_i}$ then
    $\mathrm{E\_CPU_i} + +$
  end if
Else
  if $\mathrm{E_CPU_i} > \mathrm{LOWER_CPU_i}$then
    $\mathrm{E\_CPU_i} - -$
  end if
End if

Effective Memory



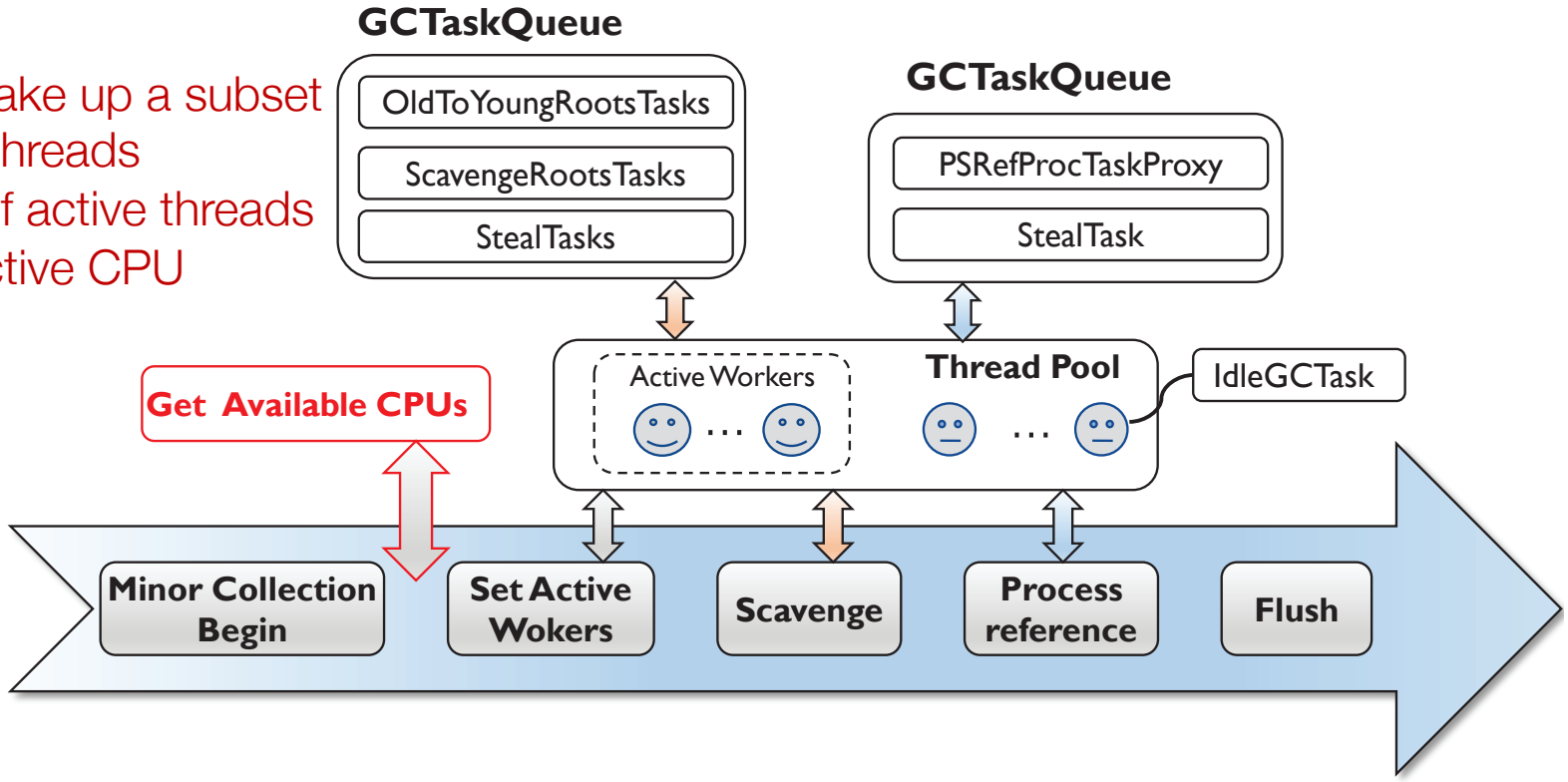
- Memory adjustment
  - [soft_limit, hard limit]
  - Gradually increase memory by 10% if
    1. Container memory usage > 90%
    2. The projected system-wide memory increase does not bring system-wide free memory below high_watermark
  - $\Delta_{system} = \frac{change\ in\ system\ free\ mem}{change\ in\ container\ mem\ usage} \times \Delta_{mem}$

Building Elastic Applications

- *Method*
  - Instruct applications to check resource availability in per-container virtual `sysfs` and adjust program-level resource management accordingly
  - CPU elasticity - adjust thread-level parallelism
  - Memory elasticity – adjust memory bounds in runtime systems

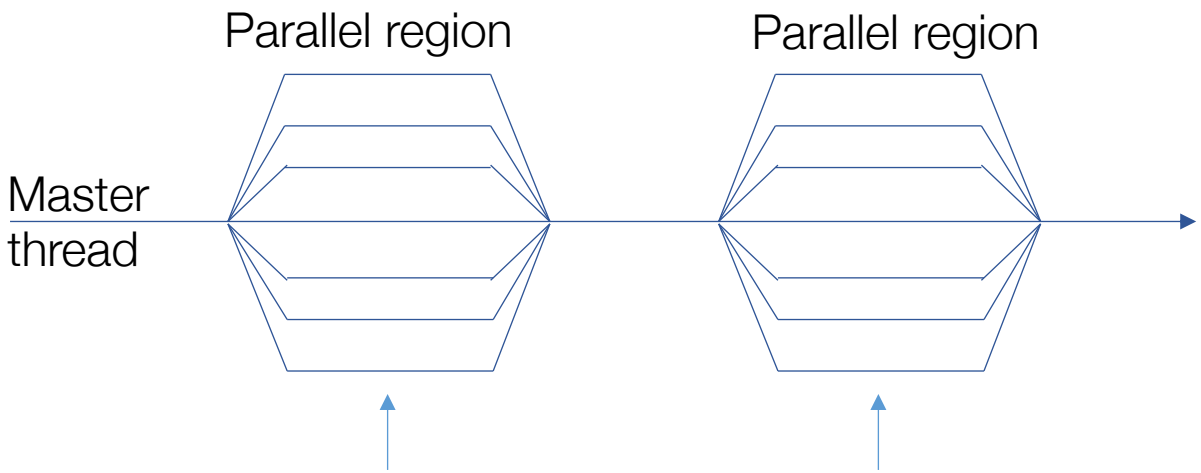
Dynamic Parallelism – HotSpot JVM

1. Only wake up a subset of GC threads
2. Set # of active threads to effective CPU



The process of a Parallel GC

Dynamic Parallelism – OpenMP

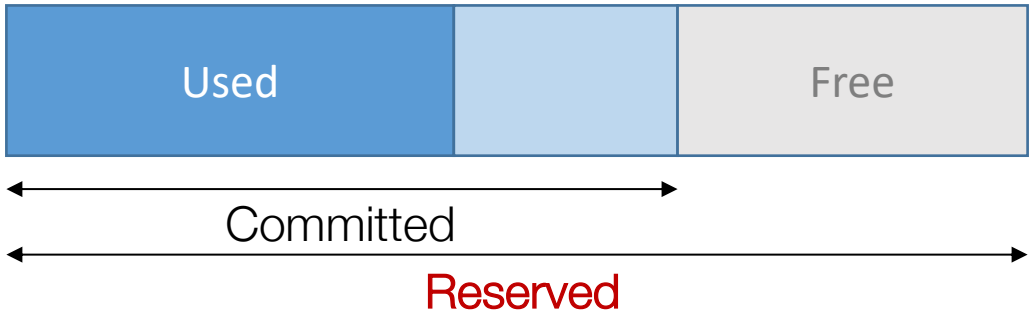


gomp_dynamic_max : $n\_thread = n\_onln - loadavg$

gomp_dynamic_max : $n\_\mathrm{thread} = \mathrm{E\_CPU}$

Elastic Memory – HotSpot Heap Management

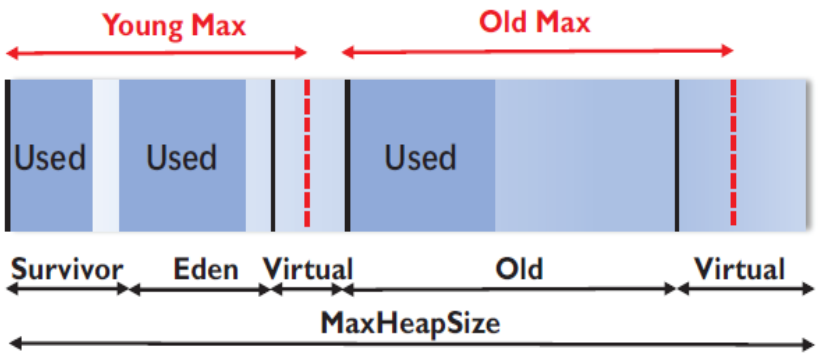
- The structure of JVM heap



- Adaptive heap sizing algorithm
  - Dynamically change the committed size
  - Expand or shrink the heap based on feedbacks of GCs.

Elastic Memory – cont'd

- Scale up
  - Set reserved memory to effective memory and rely on the sizing algorithm to expand committed space
- Scale down
  - Trigger GC to free memory if: 1) new memory size is smaller than used and 2) adjustment is needed to maintain the ratio of young and old generations



Evaluation

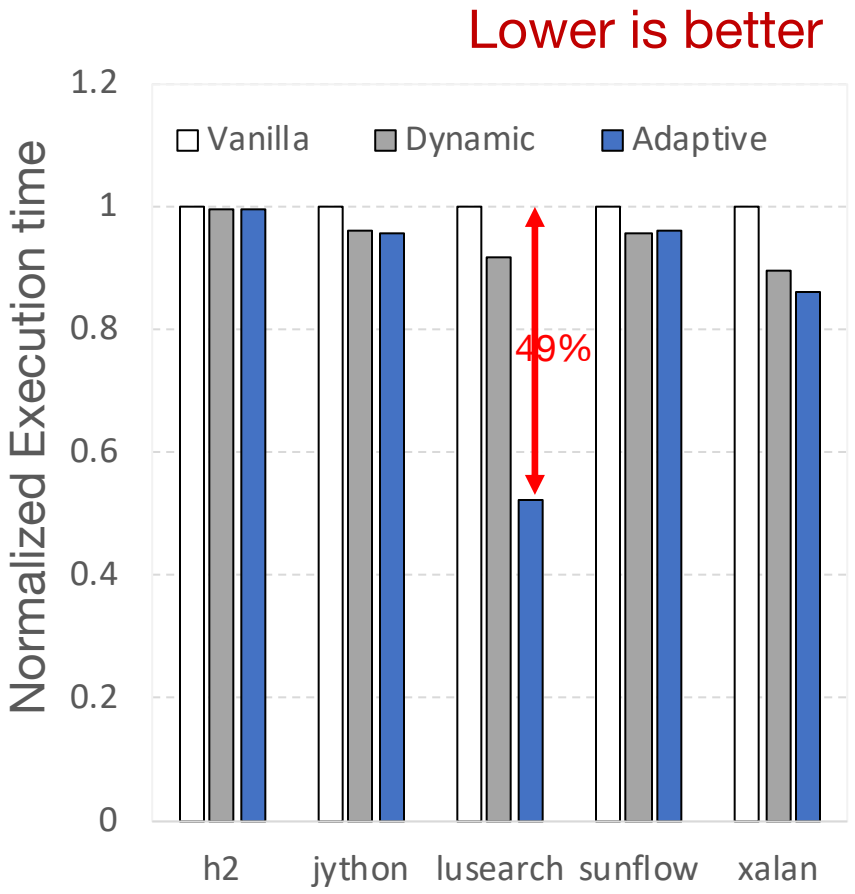
Experimental Settings

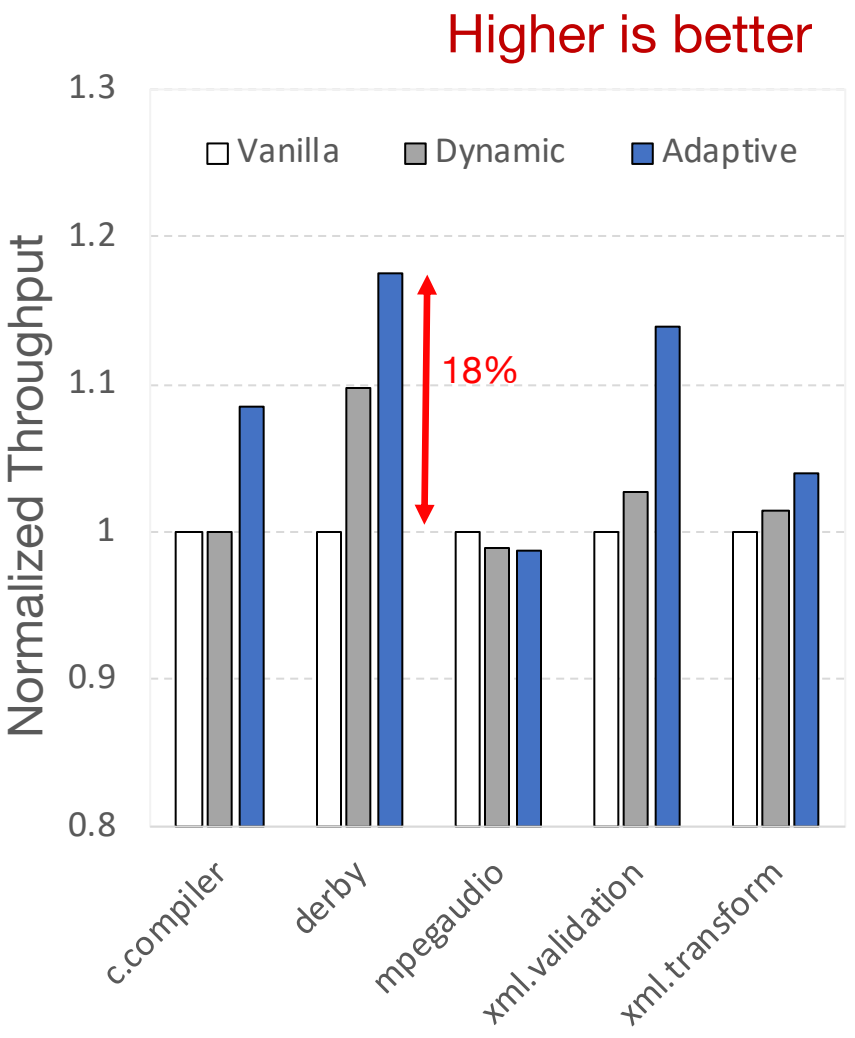
Hardware	
Server	PowerEdge R730
CPU	dual 10-core Intel Xeon 2.30 GHz processors
RAM	128GB
Storage	1TB SATA hard drive
Software	
System	CentOS 7 64bit
Kernel	4.12.3
Docker	Version 17.06.1
Benchmarks	Dacapo, PECjvm2008, HiBench, NAS parallel benchmarks

Methodology

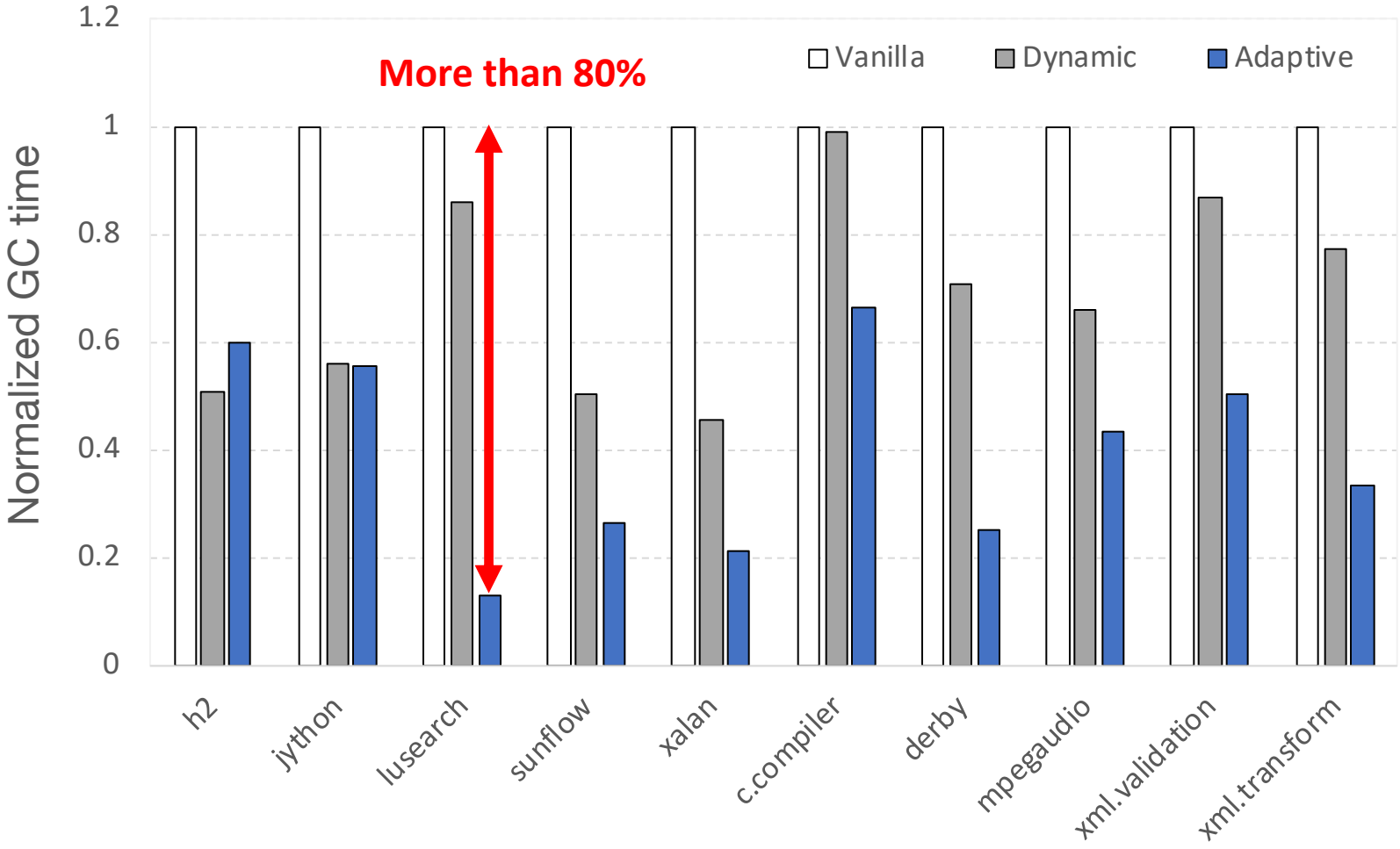
- Study various workload consolidation scenarios to evaluate
  - The (performance) benefits of per-container resource view
  - How well do applications adapt to variable resource availability
- Evaluated approaches
  - ***Vanilla:*** original JVM 8 and OpenMP without container awareness
  - ***Dynamic***: JVM 8 and OpenMP with dynamic threads enabled and JVM 8 with adaptive heap sizing
  - ***JVM 9 and JVM 10***: The new versions of JVMs that add container awareness. JVM 9 detects static CPU limit and affinity; JVM 10 detects static CPU share
  - ***Adaptive:*** Our proposed elastic apps based on effective CPU and memory

Improvement on Application Performance

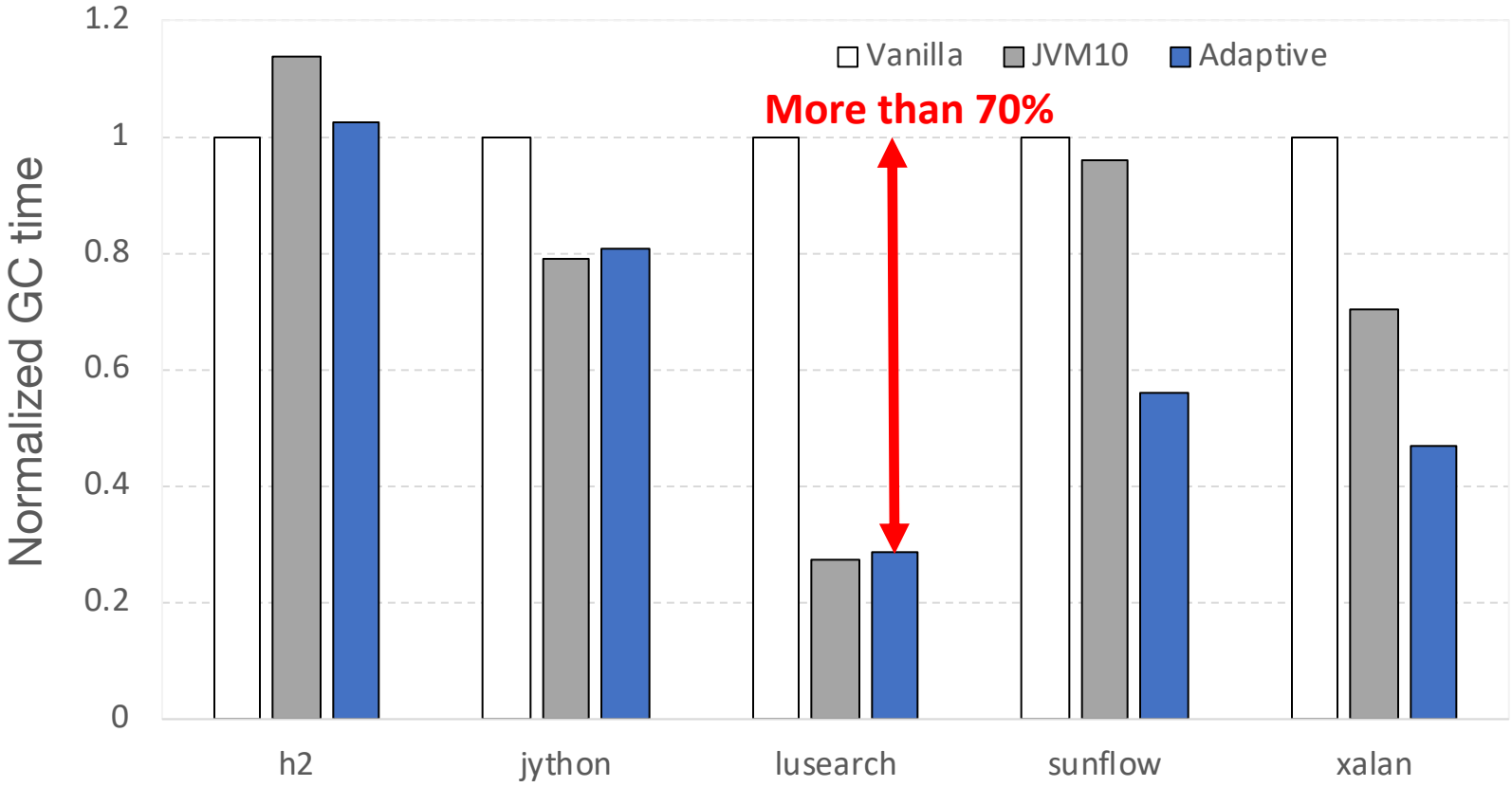




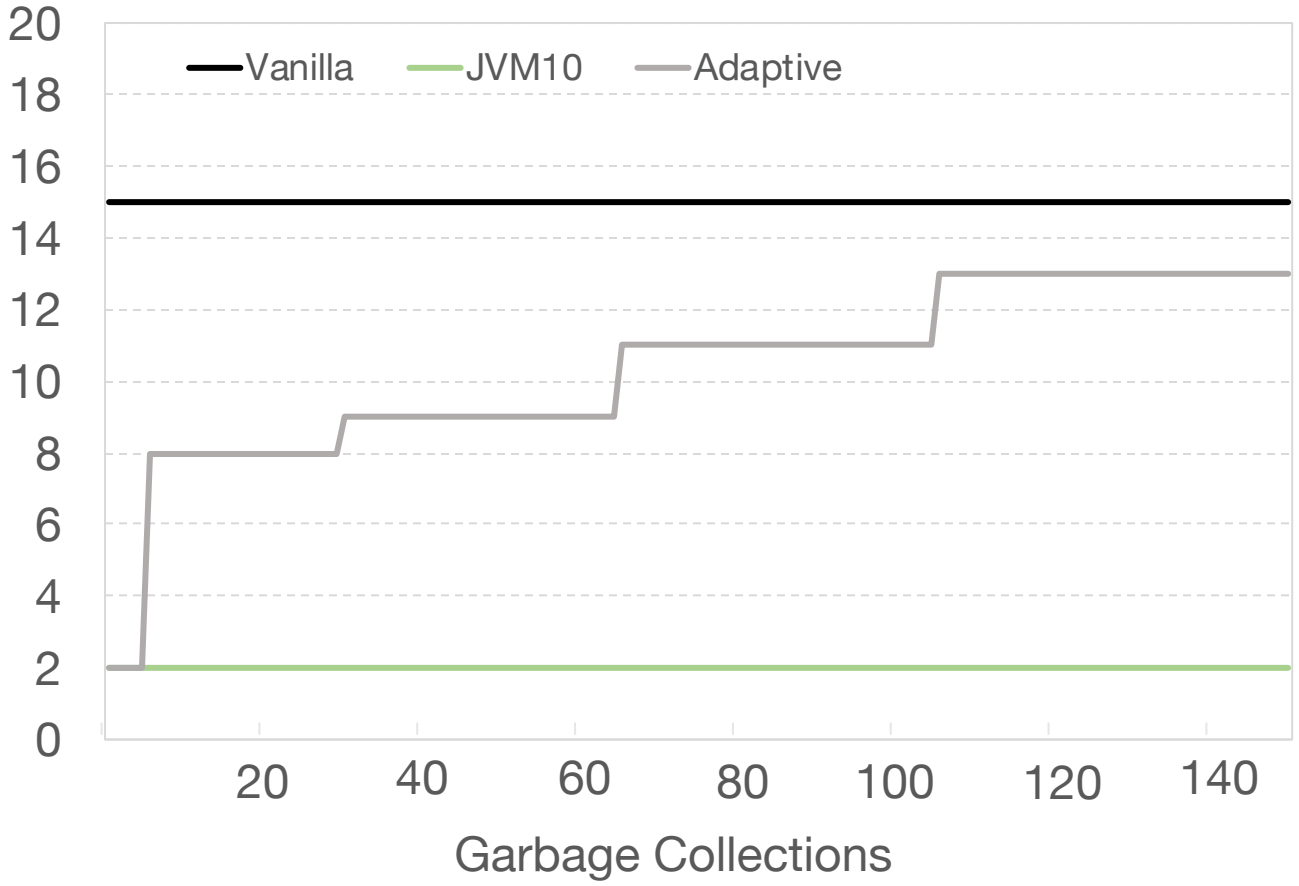
Improvement on GC time



Improvement with varying CPU availability

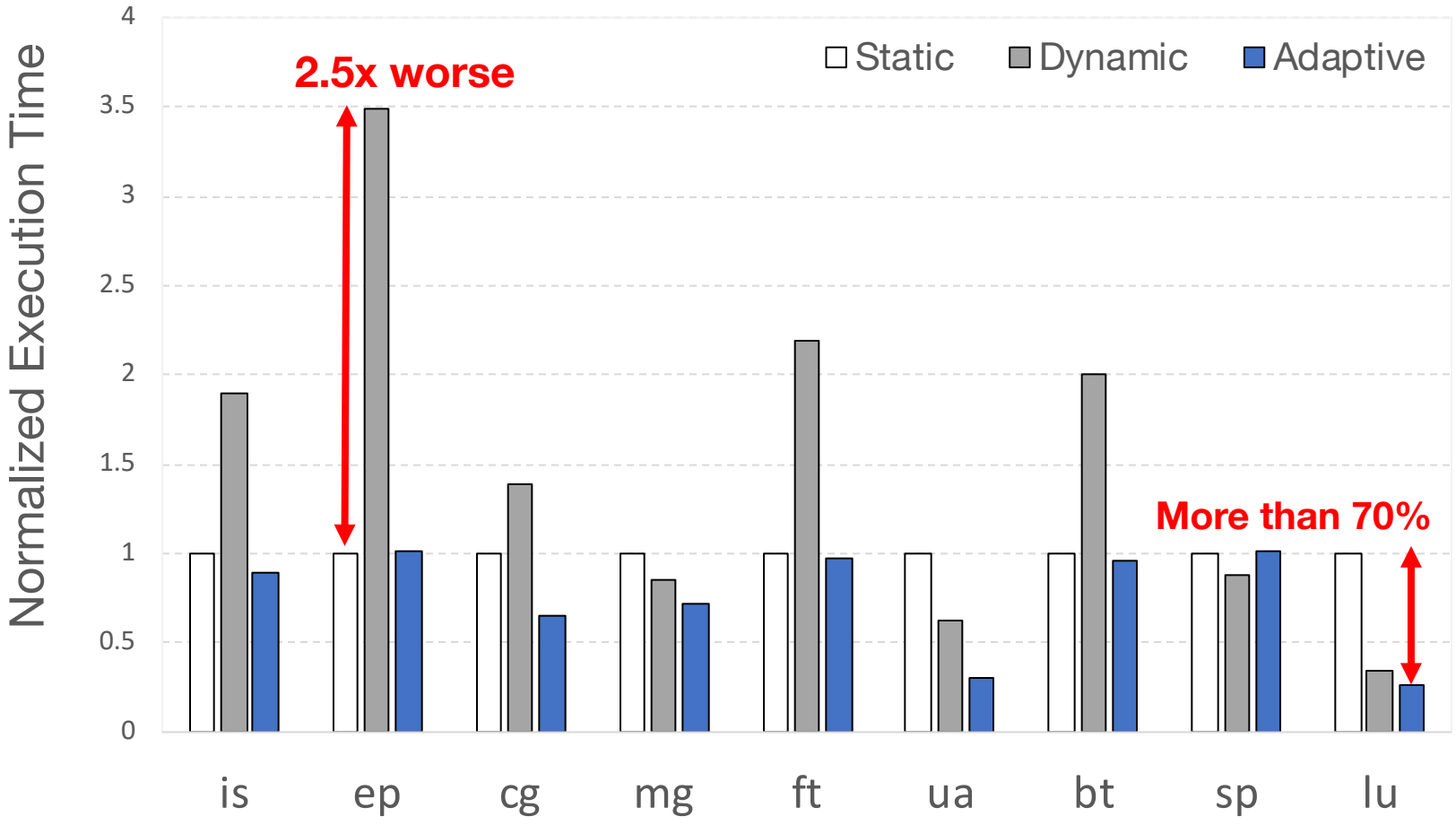


Results on Different Versions of JVM

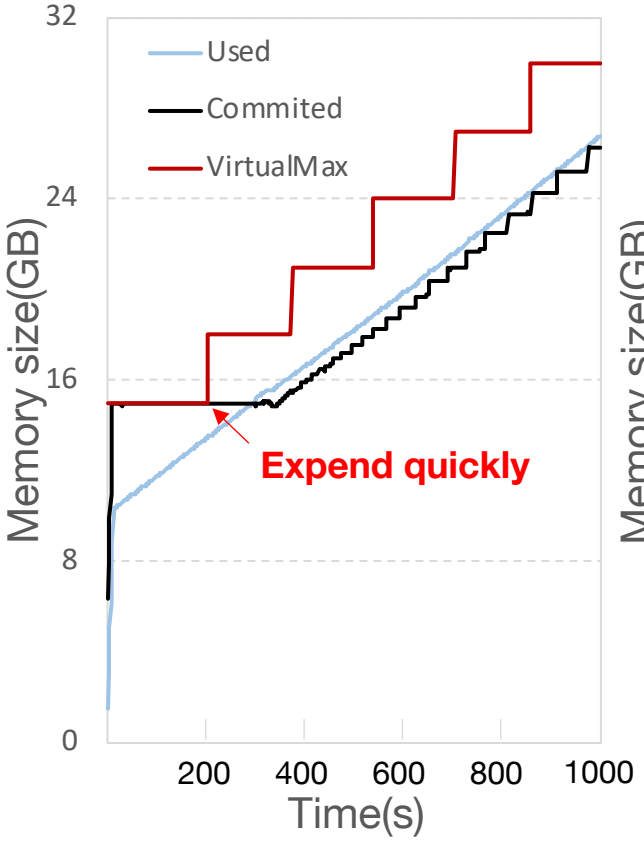


Number of GC threads in sunflow

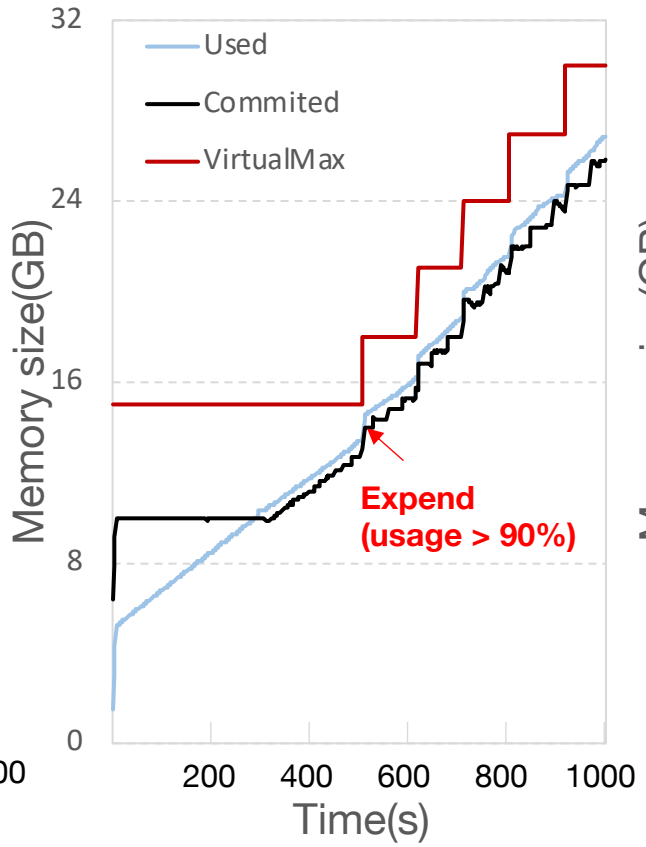
Improvement on OpenMP



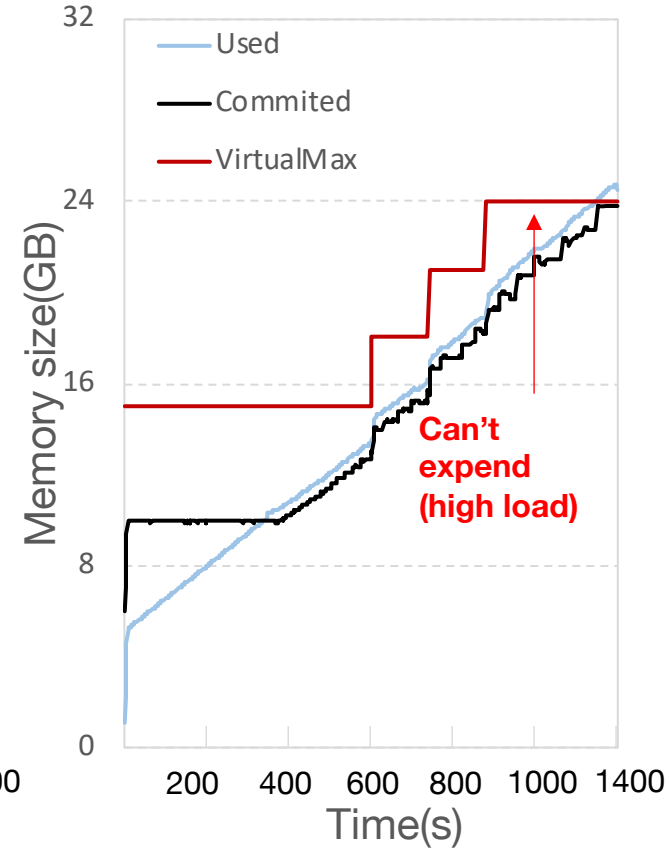
Results on Elastic Heap



Vanilla JVM(Single)



Elastic JVM(Single)



Elastic JVM(Multiple)

Conclusions and further research

Motivation:

We identified an important semantic between *resource availability* and *constraints that can lead to* containerized applications performing inefficiently.

Solution:

We created new sys_namespace to propose per-container view of resources and developed a virtual sysfs to expose resources view to user space application.

Results:

Experiment results showed consistent performance improvement for Java and OpenMP applications.

Limitations :

Our system is still inconvenient for applications or runtime systems without mechanisms to alter thread-level parallelism or memory allocation at runtime.

Thanks!

Questions?