virtualization and cloud computing @ synerg.cse.iitb

Systems and Networks Research Group
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

SynerG@ CSE IIT Bombay

100s of students and some faculty

Kameswari

Varsha

Mythili

Bhaskaran

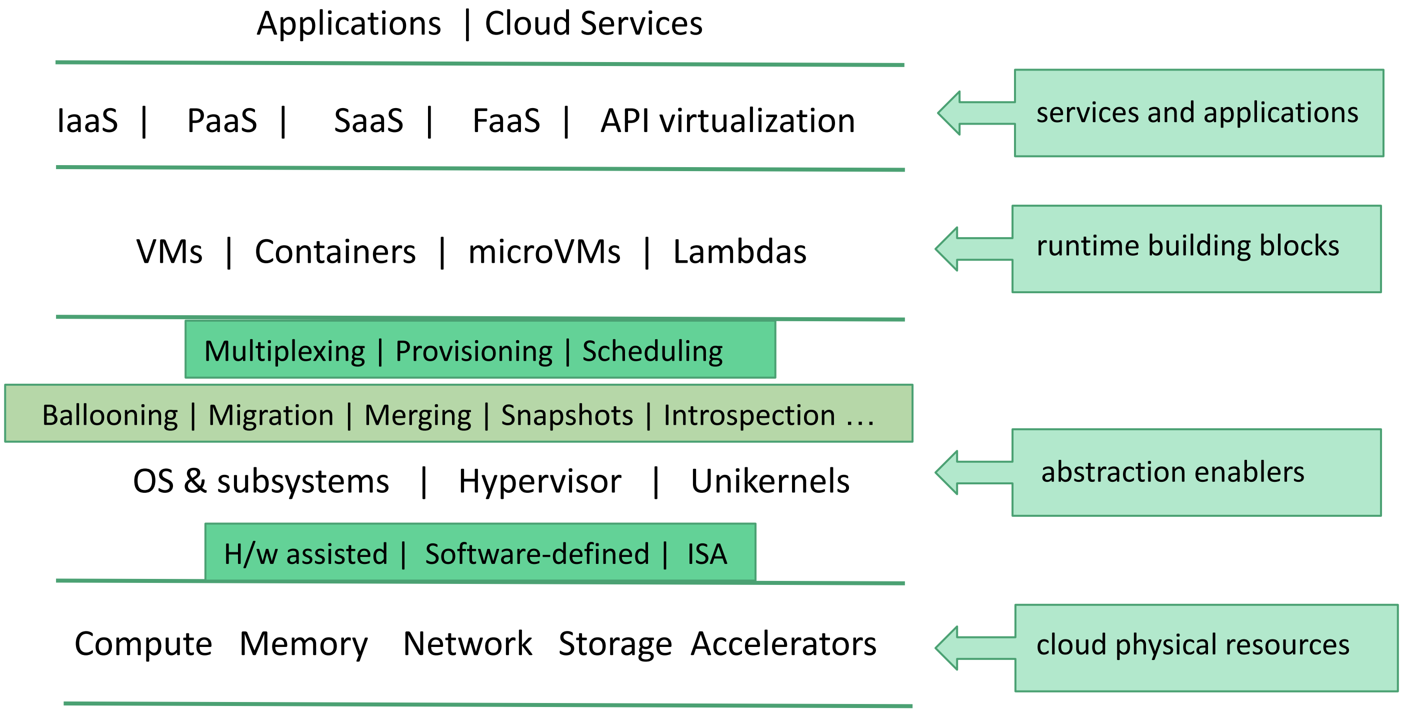
Vinay

Puru

Umesh

Biswa

The cloud services stack



Applications | Cloud Services

IaaS | PaaS | SaaS | FaaS | API virtualization

VMs | Containers | microVMs | Lambdas

Multiplexing | Provisioning | Scheduling

Ballooning | Migration | Merging | Snapshots | Introspection …

OS & subsystems | Hypervisor | Unikernels

H/w assisted | Software-defined | ISA

Compute Memory Network Storage Accelerators

services and applications

runtime building blocks

abstraction enablers

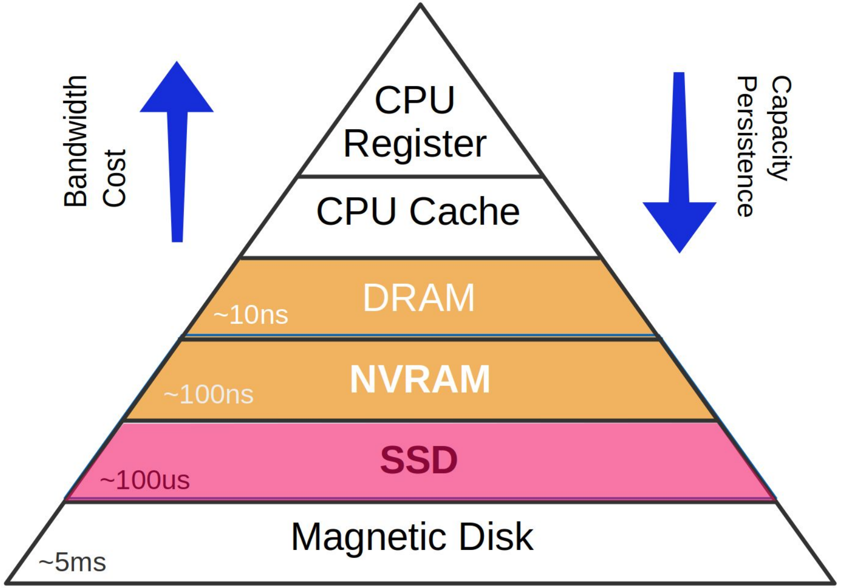
cloud physical resources

SymFlex: Elastic, Persistent and Symbiotic SSD Caching in Virtualization Environments

Muhammed Unais P, Purushottam Kulkarni

ACM/SPEC International Conference on Performance Engineering (ICPE 2021)

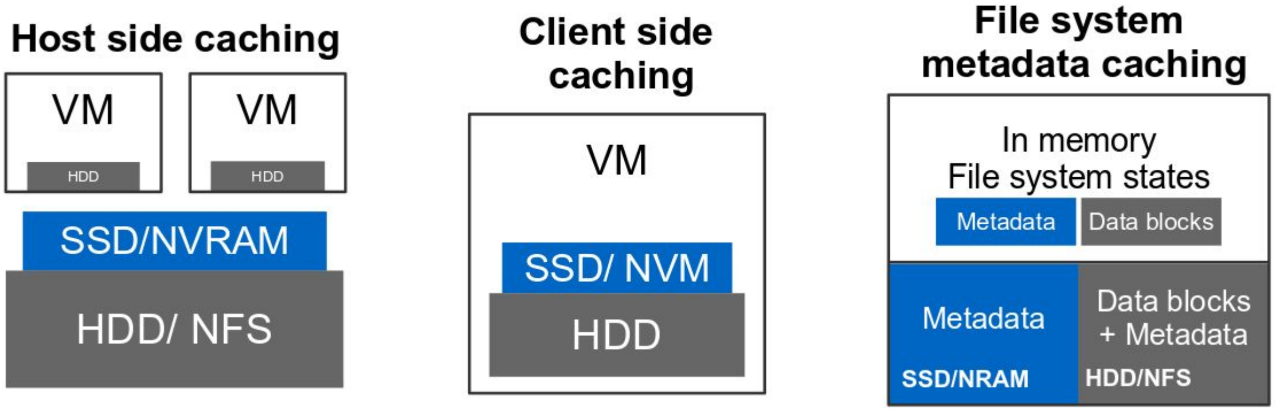
The (IO) caching hierarchy



The wishlist

Low latency, High bandwidth, Byte addressable, Persistence

SSD caching options



- Multiple feasible configurations and usages with SSDs (as caches)
- Focus of this work,
    - SSD caches with virtualization based IaaS setups

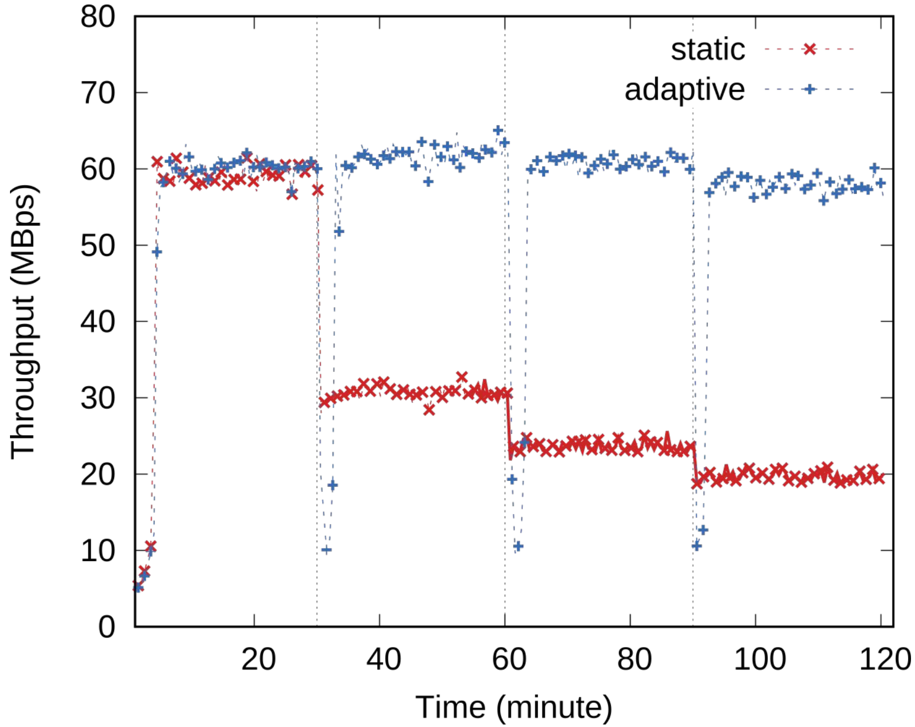
NO overcommitment, NO IaaS!

- Resource ***overcommitment*** a key motivation of IaaS based service provisioning
  - E.g., Four 8 GB VMs GB on a 16 GB machine, 16 vCPUs on a 4 CPU machine …

- The overcommitment secret sauce …
- Relies on statistical multiplexing of resources
- Requires ***dynamic*** resource provisioning/multiplexing mechanisms
  - CPU and IO scheduling, demand paging, memory ballooning, …
  - Employ temporal and spatial multiplexing of resources
  - **Elastic** resources are vital building blocks

- w.r.t SSDs used for caching
  - Cache sizes need to be dynamically resized to account for load, and performance and usage policies

Elastic SSD in action



- With change in load, change in SSD cache size maintains throughput levels

The Symbiotic Game Plan

- Who resizes the SSD cache?
- Option 1: **The hypervisor**
  - Operates transparent to guest OSes
  - Cache usage semantics and load behaviour unknown to hypervisor
    - Metadata information, index of important objects, upcoming events, …
- Option 2: **The guest OS**
  - Guest level semantics can be incorporated for eviction decisions
  - Statically sized and pass-through assignment of SSD partition to virtual machine
    - Limits elasticity options, and consolidation options with SSD caching

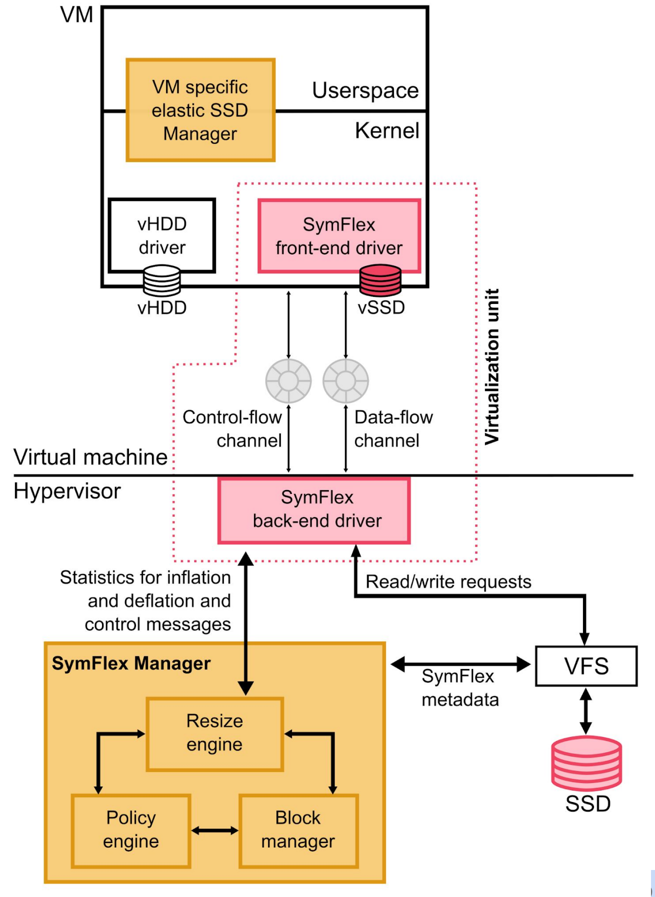
- The **symbiotic** plan
  - Hypervisor manages sizing (based on performance, usage policies etc.)
  - Guest OS manages cache membership based on semantics of relevance

Problem description

- The **symbiotic** plan
  - Hypervisor manages sizing (based on performance, usage policies etc.)
  - Guest OS manages cache membership based on semantics of relevance

- **Missing mechanism**: An ***elastic*** virtualized SSD device
- Design and engineer an elastic virtualized device for VMs
- Build a framework for symbiotic management of SSD caches across VMs
- Demonstrate efficacy of elasticity for IO caching in virtual machines

SymFlex architecture



Registration

<vm-id, size, current-size, persist flag>

Read/write operation via frontend and backend driver

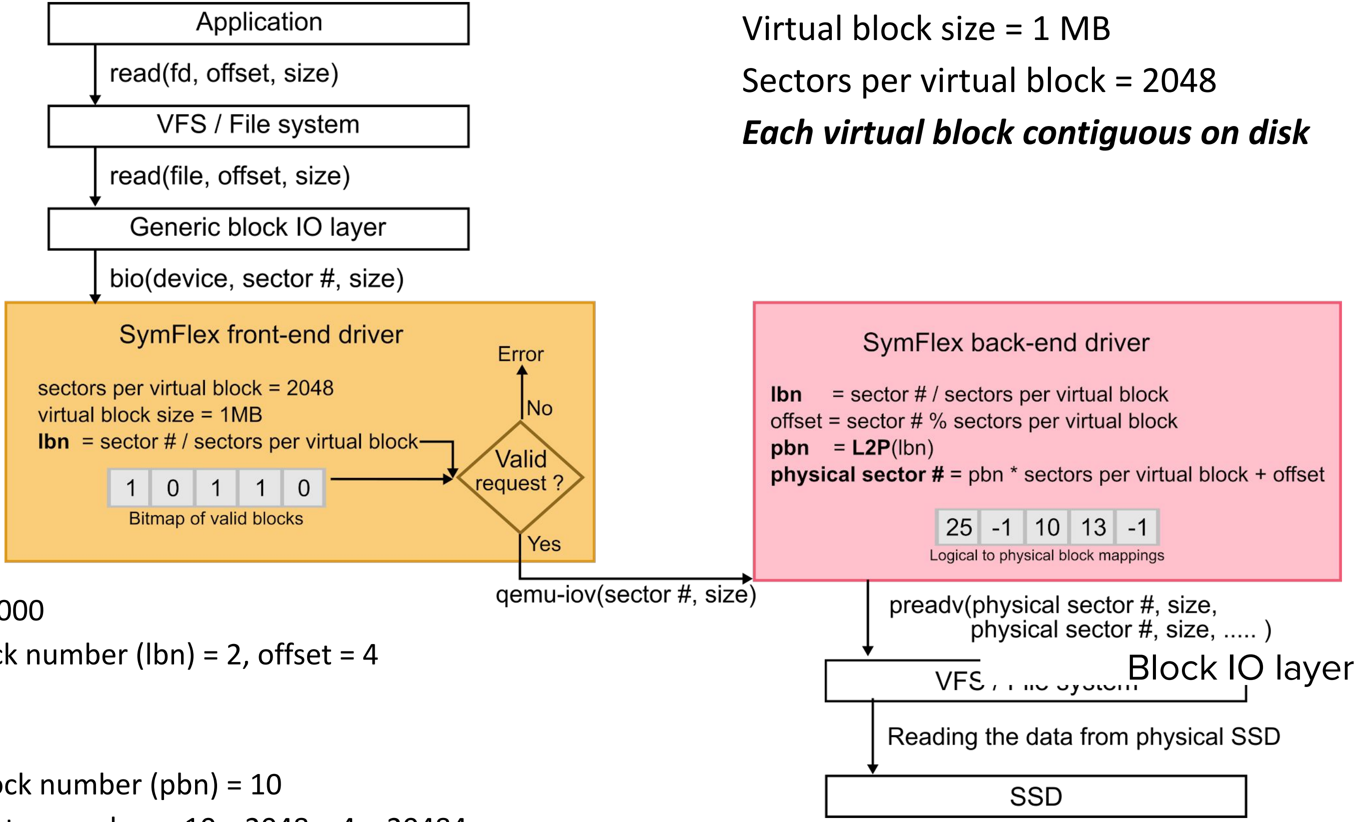
Inflation/deflation of SSD triggered by SymFlex manager

SymFlex IO operations

Virtual block size = 1 MB

Sectors per virtual block = 2048

Each virtual block contiguous on disk


Application
read(fd, offset, size)
VFS / File system
read(file, offset, size)
Generic block IO layer
bio(device, sector #, size)
SymFlex front-end driver
sectors per virtual block = 2048
virtual block size = 1MB
lbn = sector # / sectors per virtual block
1 0 1 1 0
Bitmap of valid blocks
Valid request ?
Error
No
Yes
qemu-iov(sector #, size)
SymFlex back-end driver
lbn = sector # / sectors per virtual block
offset = sector # % sectors per virtual block
pbn = L2P(lbn)
physical sector # = pbn * sectors per virtual block + offset
25 -1 10 13 -1
Logical to physical block mappings
preadv(physical sector #, size, physical sector #, size, ...)
VFS / File system
Block IO layer
Reading the data from physical SSD
SSD

Front-end

sector# = 5000

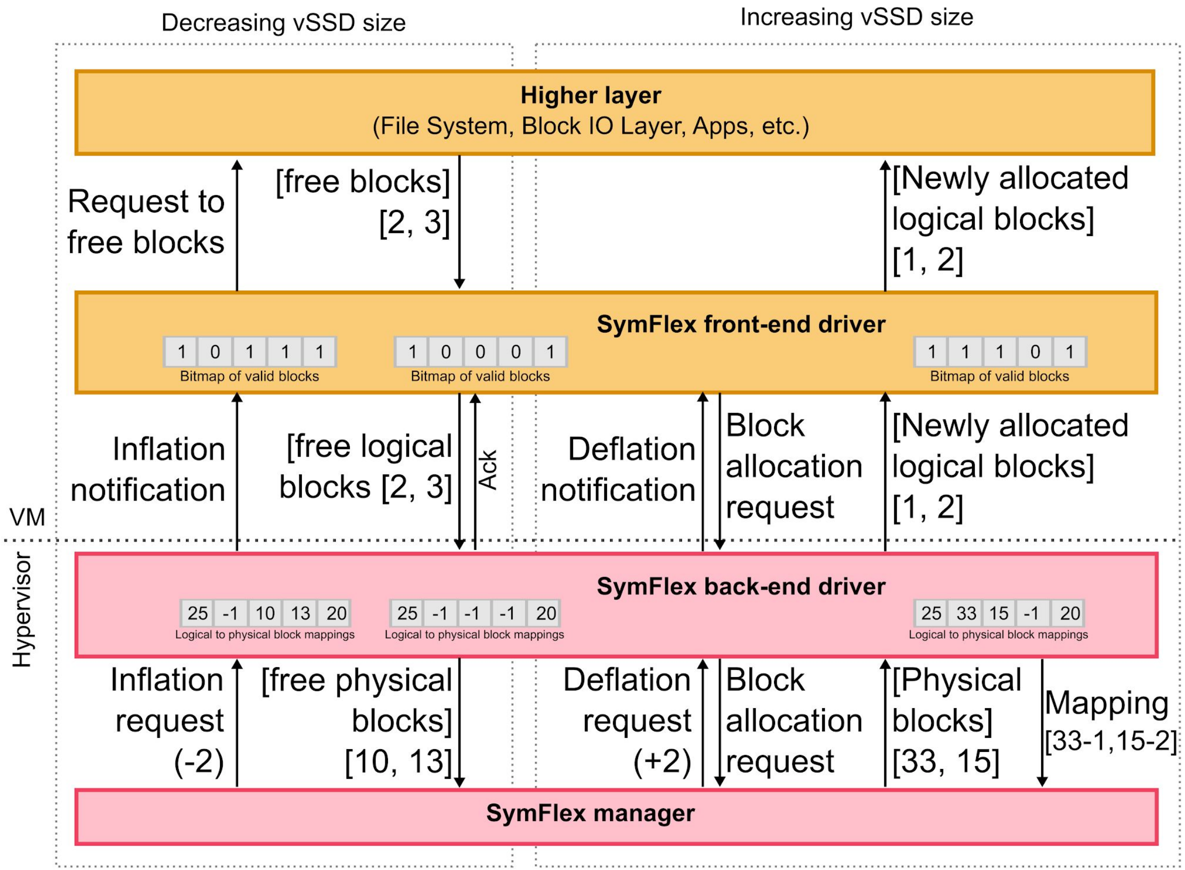
Logical block number (lbn) = 2, offset = 4

Back-end

Physical block number (pbn) = 10

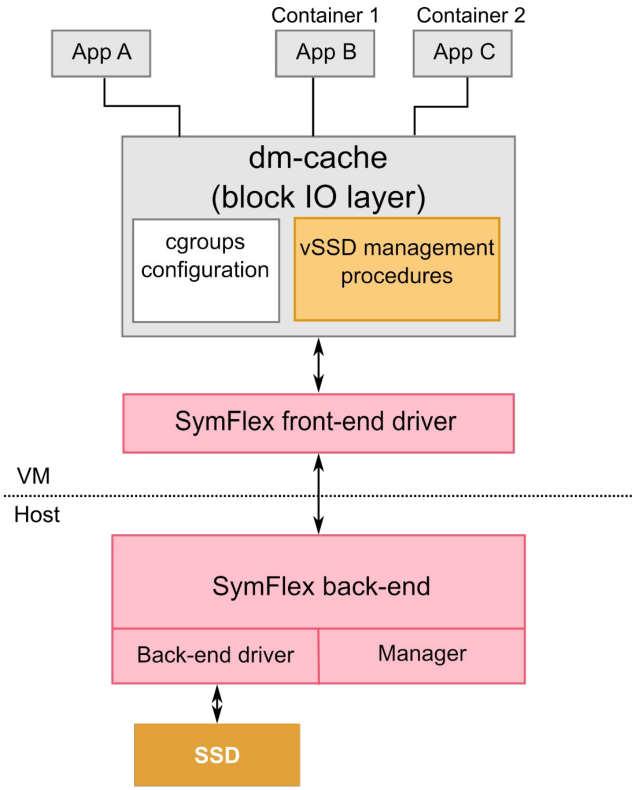
Physical sector number = 10 x 2048 + 4 = 20484

Resizing in action

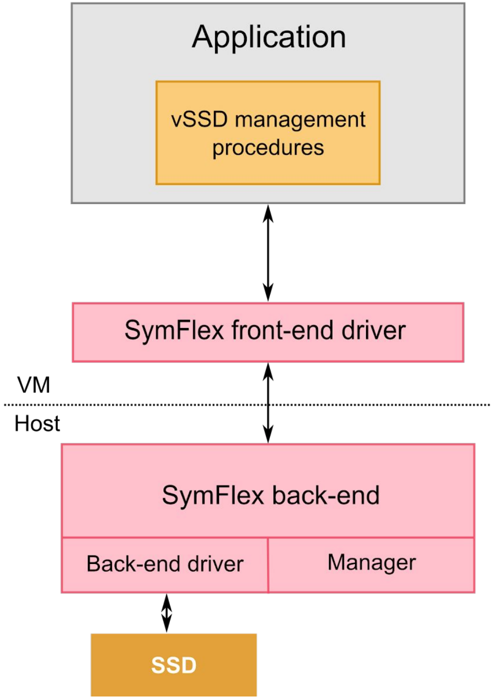


Where to place SSD management procedure?

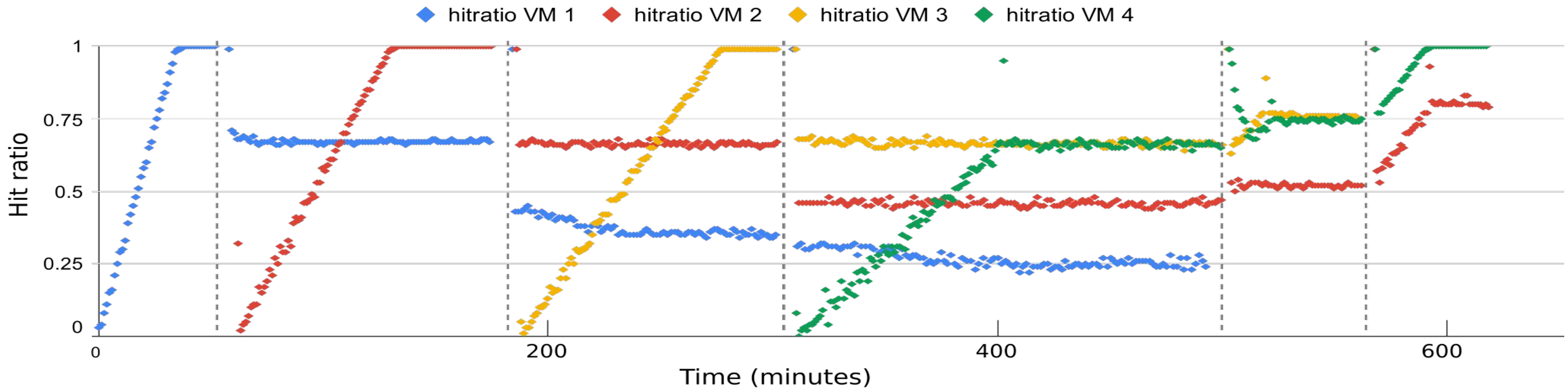
cgroups + block IO layer

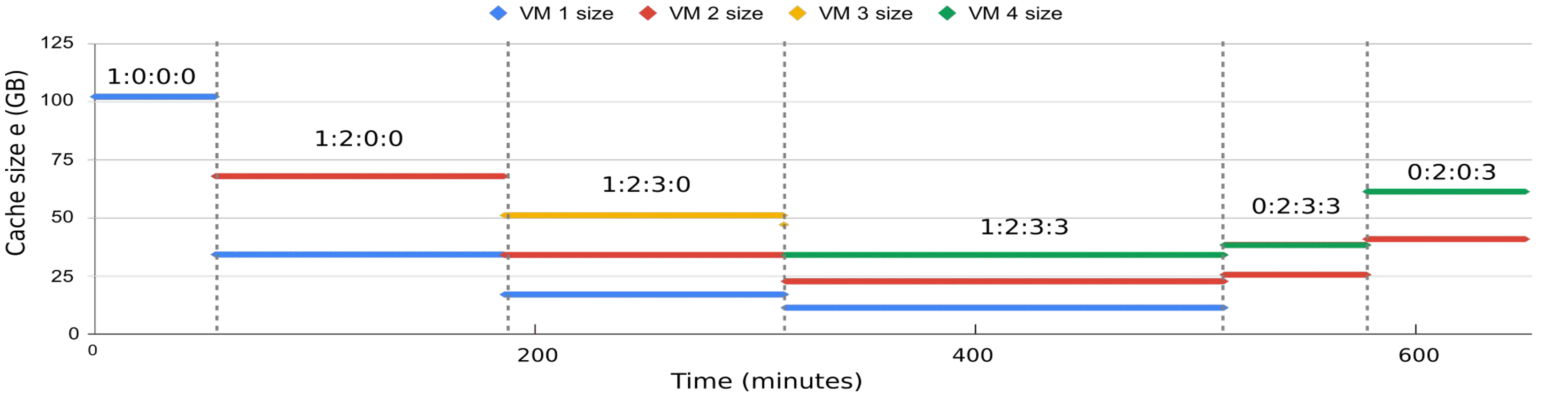


Application layer

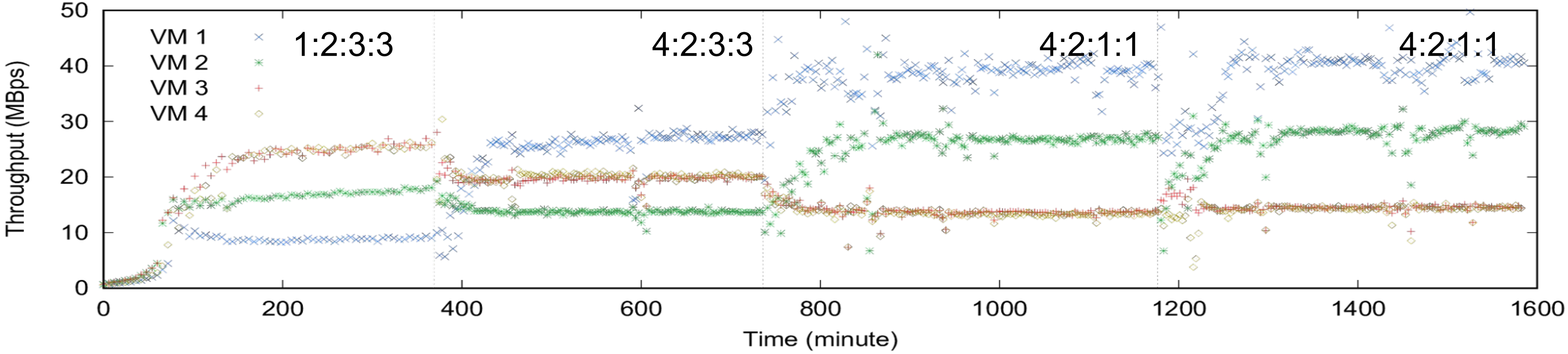


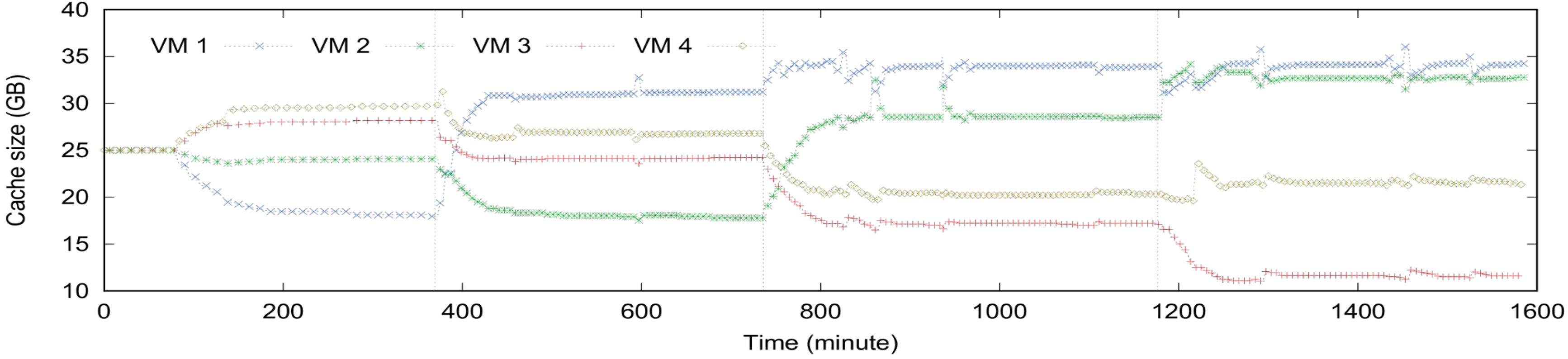
Share based cache allocation





Cache allocation with proportionate throughput





Catalyst: GPU-assisted rapid memory deduplication in virtualization environments

Anshuj Garg, Debadatta Mishra, Purushottam Kulkarni

Virtual Execution Environments (VEE) 2017

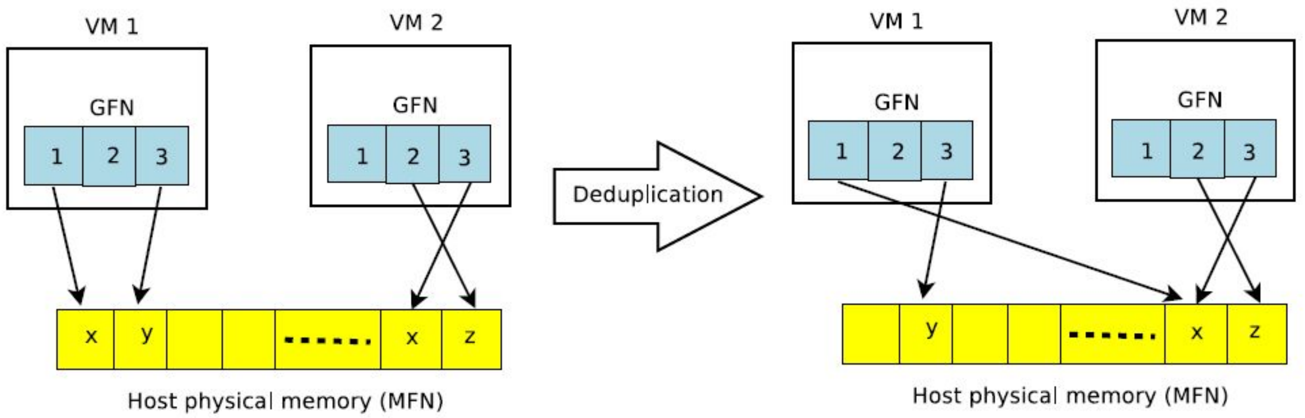
Cloud VMs and content redundancy

Several standardized software components inside cloud VMs

Memory contents across VMs can tend to be similar

Implications on memory efficiency and VM consolidation

Memory deduplication



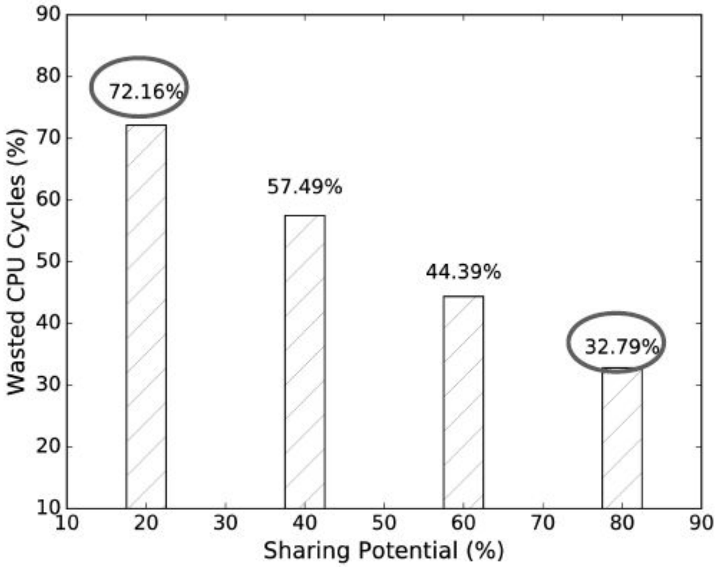
In-band and **out-of-band techniques**

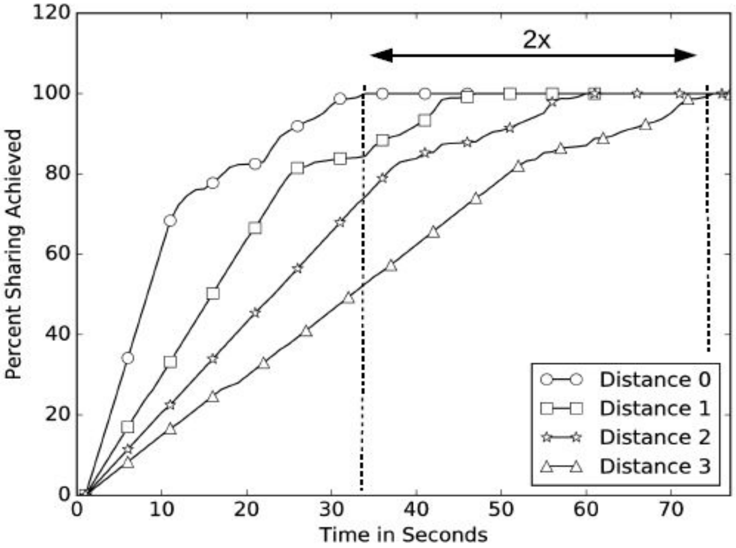
Scan and de-duplicate same pages (to maintain) single copy

Need to access and assess each page for deduplication check

Scope of this work: ***Improve efficiency of out-of-band memory dedup techniques***

Out-of-band sharing inefficiency





CPU cost is non-trivial to share pages

Wasted CPU cycles high with low sharing potential

Sharing characteristic determines time required to achieve sharing potential

Free-riding the GPU

Basic idea

Hashing contents (of a page) and hash comparison are SIMD!

Opportunistically use GPU (to save CPU cycles)

Hash page contents, Sort hash values, Compare and increment

Challenges

Memory mappings in kernel & kernel does not have direct access to GPU.

GPU cannot (could not) access physical memory directly.

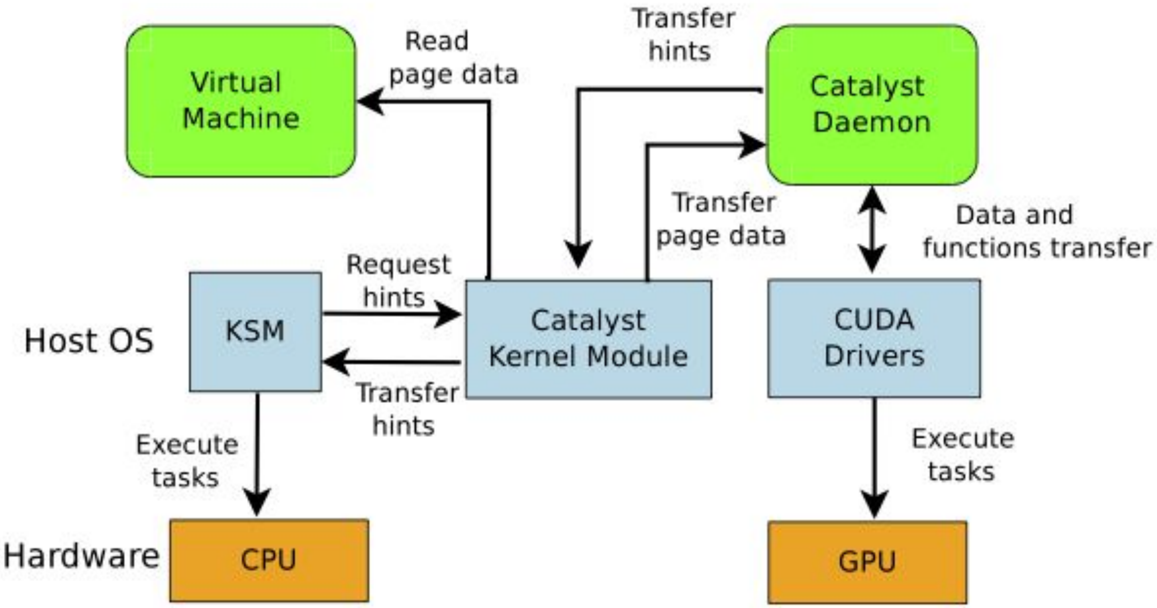
Data transfer overheads to GPUs are non-trivial.

Catalyst design

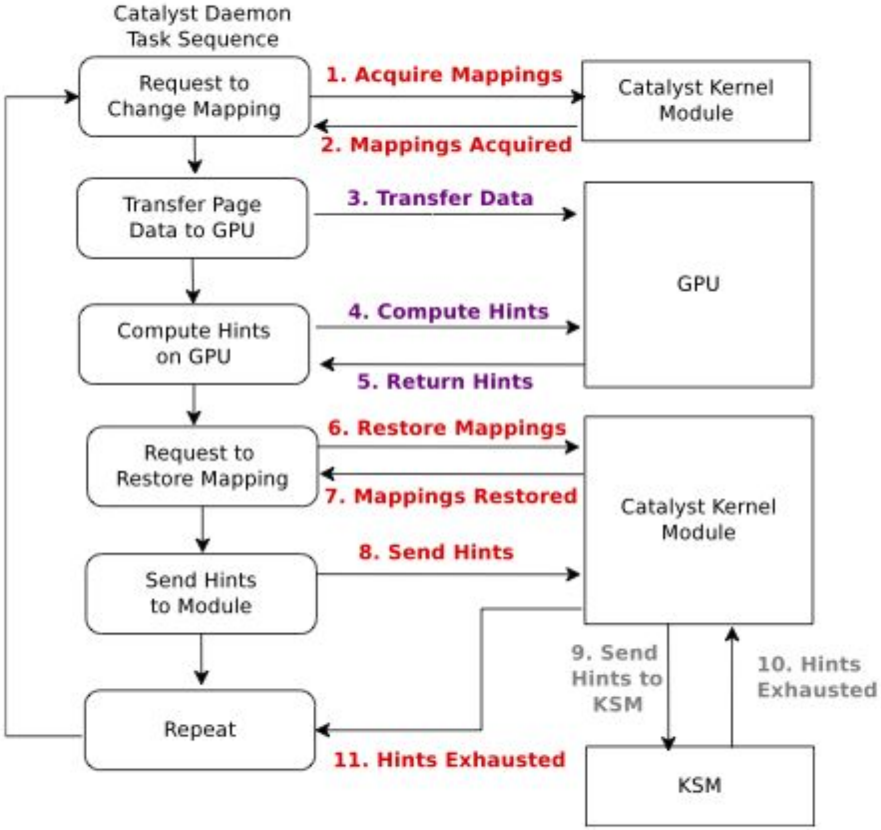
KSM --- Kernel Samepage Merging

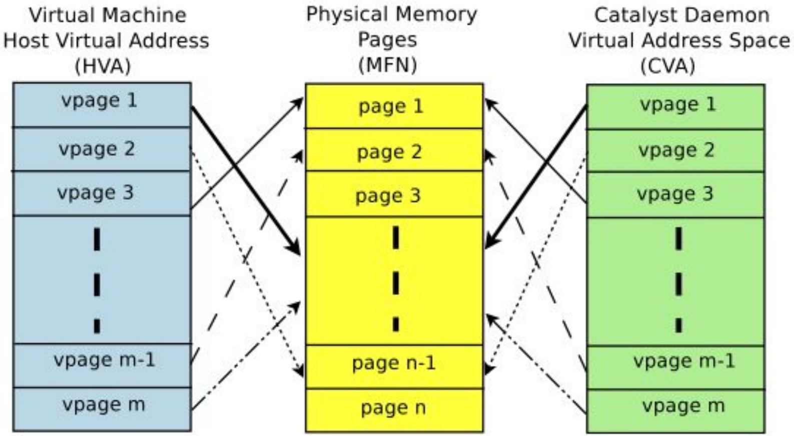
Generates ***hints*** for pages with same hash values

KSM performs ***targeted*** scanning

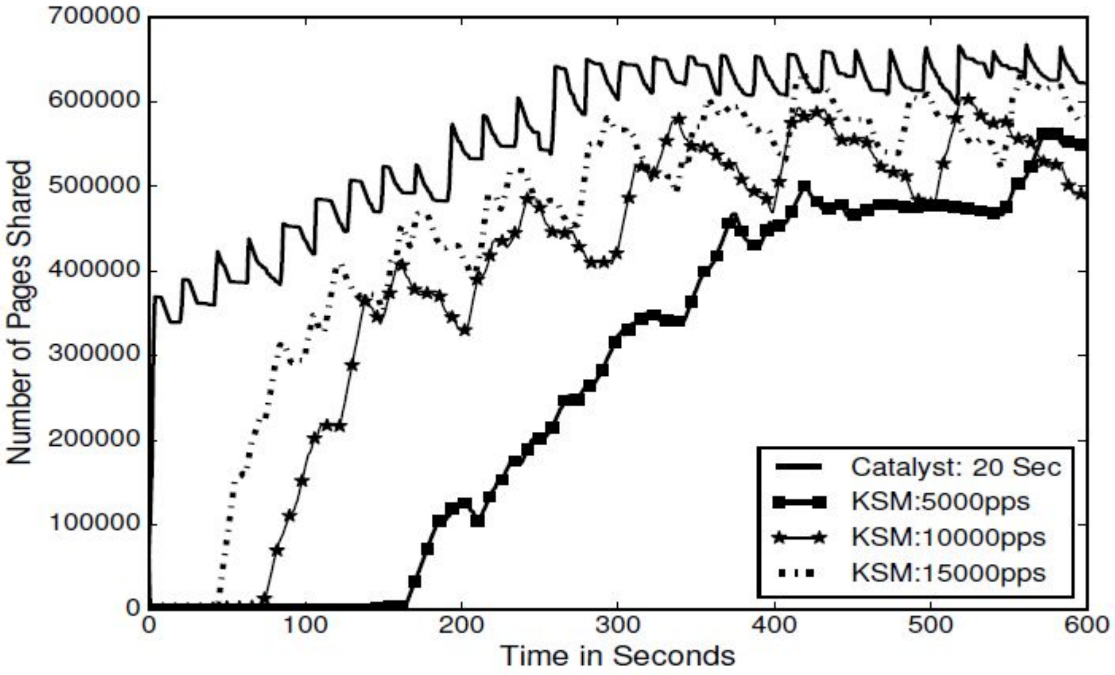


Catalyst sequence of operations




Virtual Machine Host Virtual Address (HVA)
vpage 1
vpage 2
vpage 3
vpage m-1
vpage m
Physical Memory Pages (MFN)
page 1
page 2
page 3
page n-1
page n
Catalyst Daemon Virtual Address Space (CVA)
vpage 1
vpage 2
vpage 3
vpage m-1
vpage m

Catalyst performance



3 VMs

Fileserver, varmail, synthetic

Memory sharing benefits

1.25x to 1.5x

CPU cycles saved

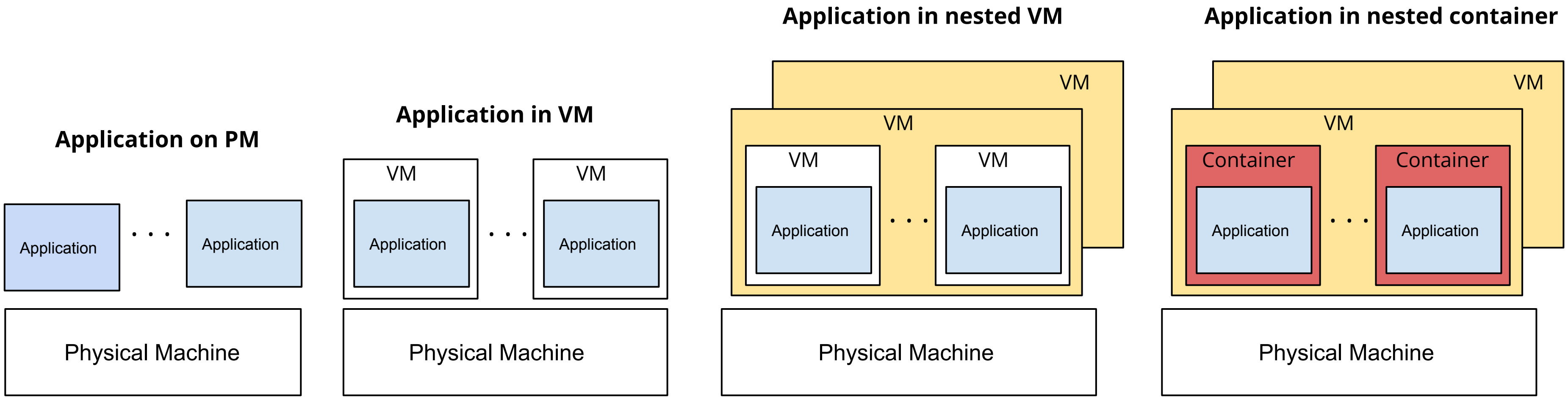
18%

Portkey: Hypervisor-Assisted Container Migration in Nested Cloud Environments

Chandra Prakash, Debadatta Mishra, Purushottam Kulkarni, Umesh Bellur

18th International Conference on Virtual Execution Environments
VEE 2022

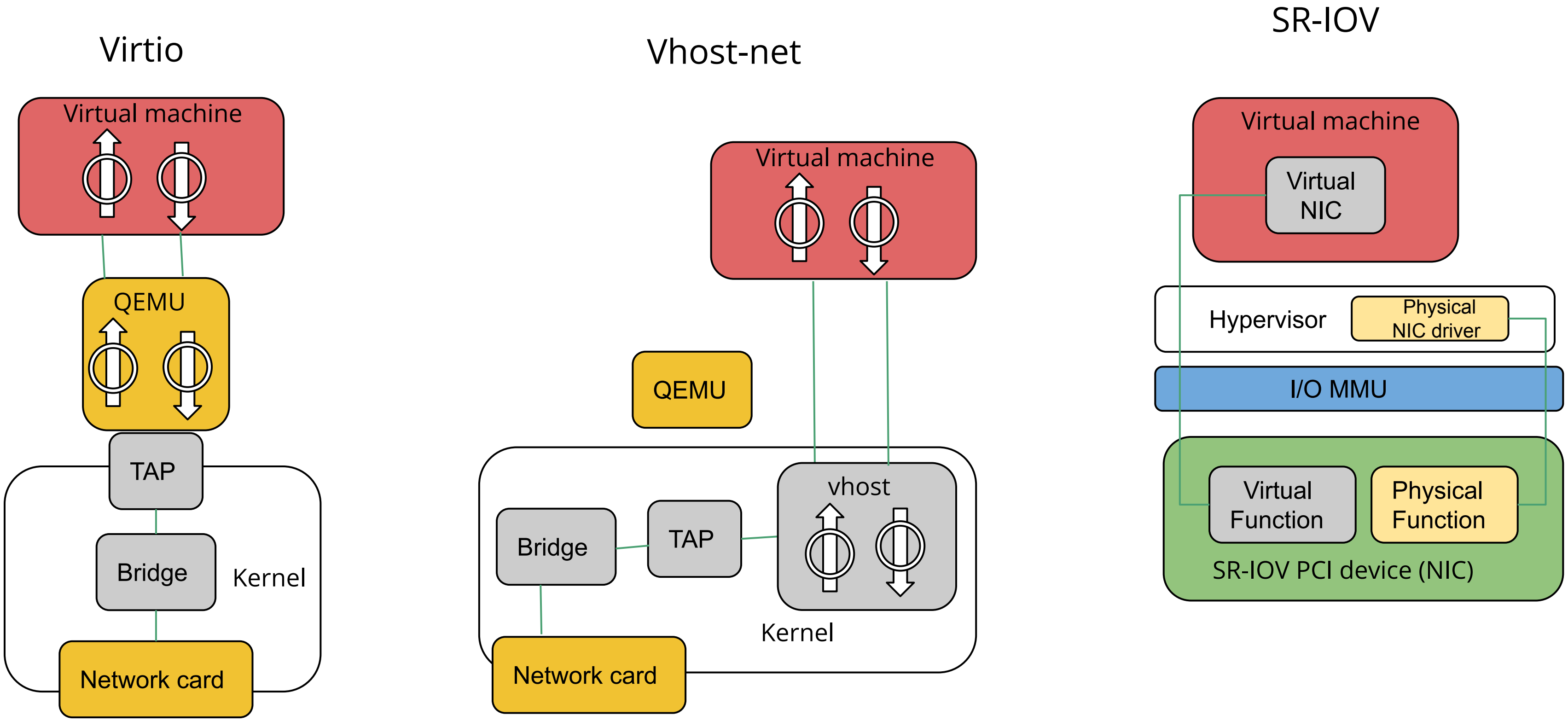
Nested setup and migration



- ➢ Nested containers in VMs employed by cloud providers such as VMware Tanzu, Google Application Engine, Heroku, Amazon elastic containers.

- ➢ Migration is key for Load Balancing, Hotspot Mitigation And Server consolidation.

Network I/O in virtualized environment



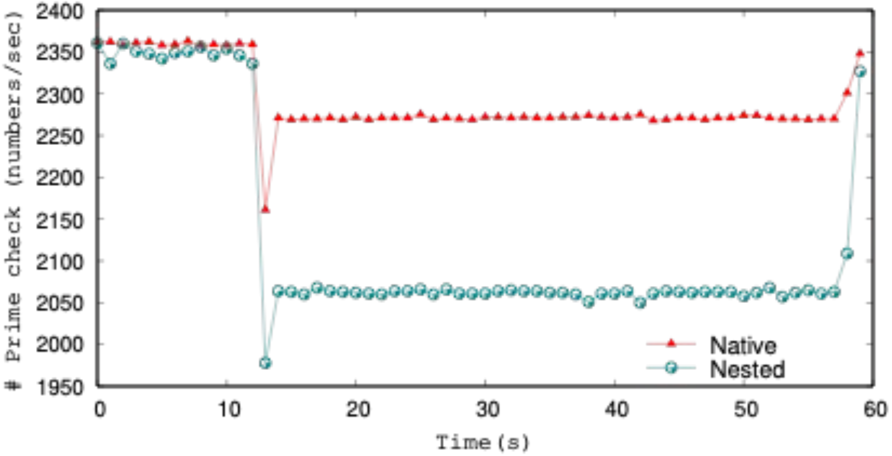
* Image source: Red-Hat

Motivation and Problem Definition

CPU utilization during quiescent container migration

Setup	Source PM (~%)	Destination PM (~%)
Native	18	25
Nested	70	115

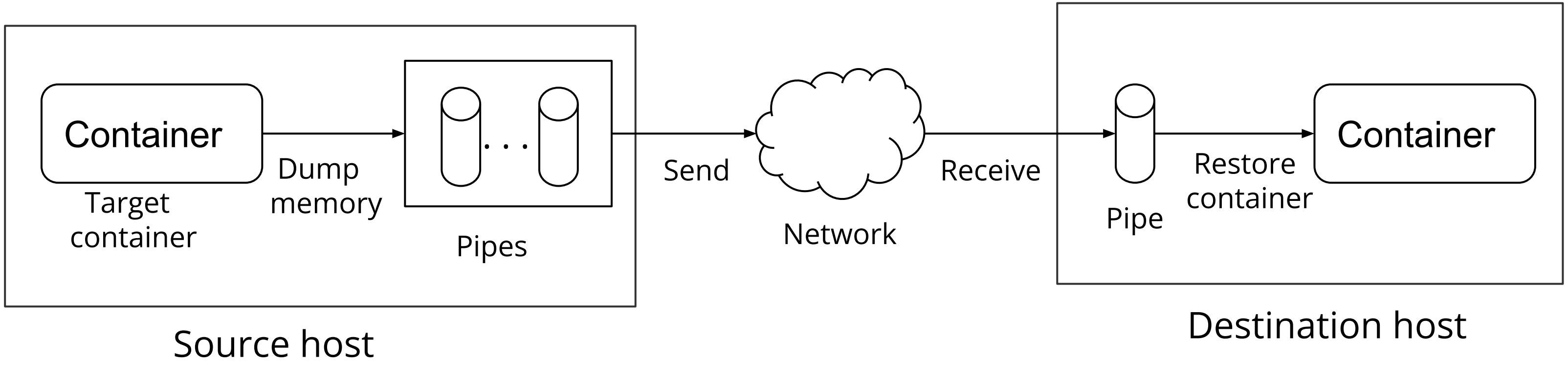
#primes checked per second



Goal

Develop a ***software defined*** framework to reduce CPU overheads without degrading network performance for nested container migration

Diskless Migration using CRIU



- ➢ CRIU collects target process' memory in several pipes and sends over the network
- ➢ Maximum size of data per send operation is 4 MB (size of pipe)
- ➢ **With nested setups**
  - ○ Data transfer over the network is main cause for high CPU utilization
  - ○ During migration, of ~70% CPU usage at source PM, ~58% is used by the hypervisor

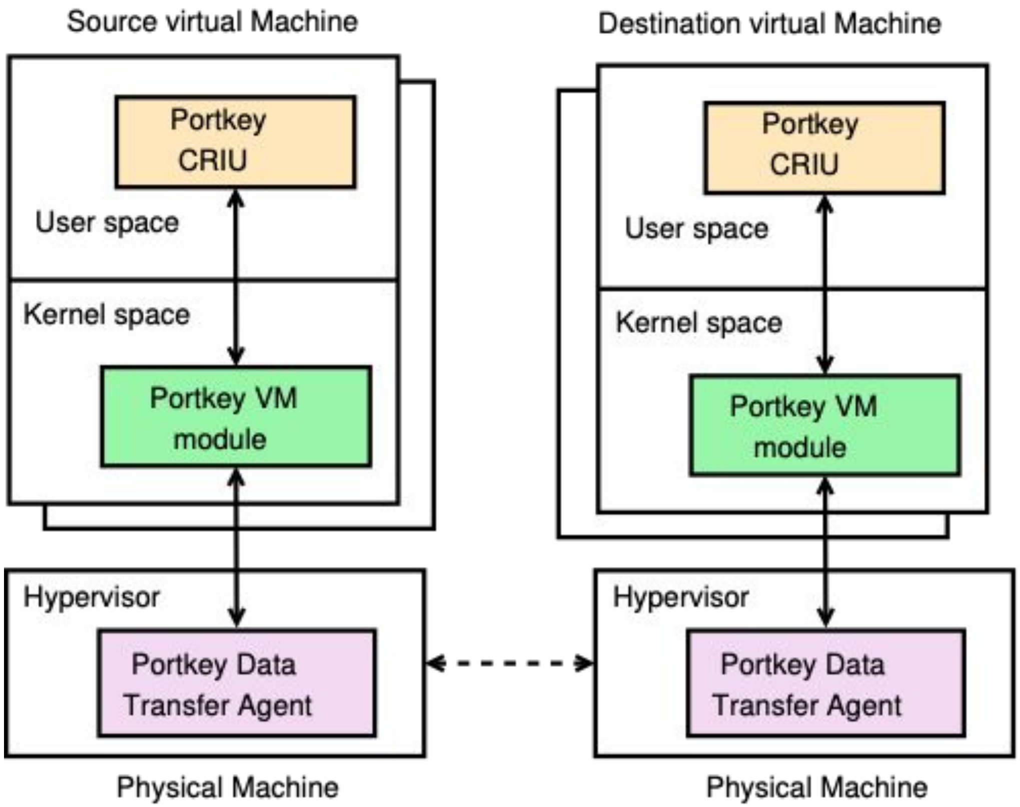
Possible Solution Approach

- Compression of migration data
  - Compression/decompression incurs high CPU overhead and decompression will increase the down time

- Hardware assisted solution (SR-IOV)
  - Additional hardware cost and restrictions such as movement and scalability

- Offload network operations of VM to the hypervisor (para-virtualization)
  - Flexible to use without restrictions and additional hardware cost

Portkey: Hypervisor-Assisted Migration

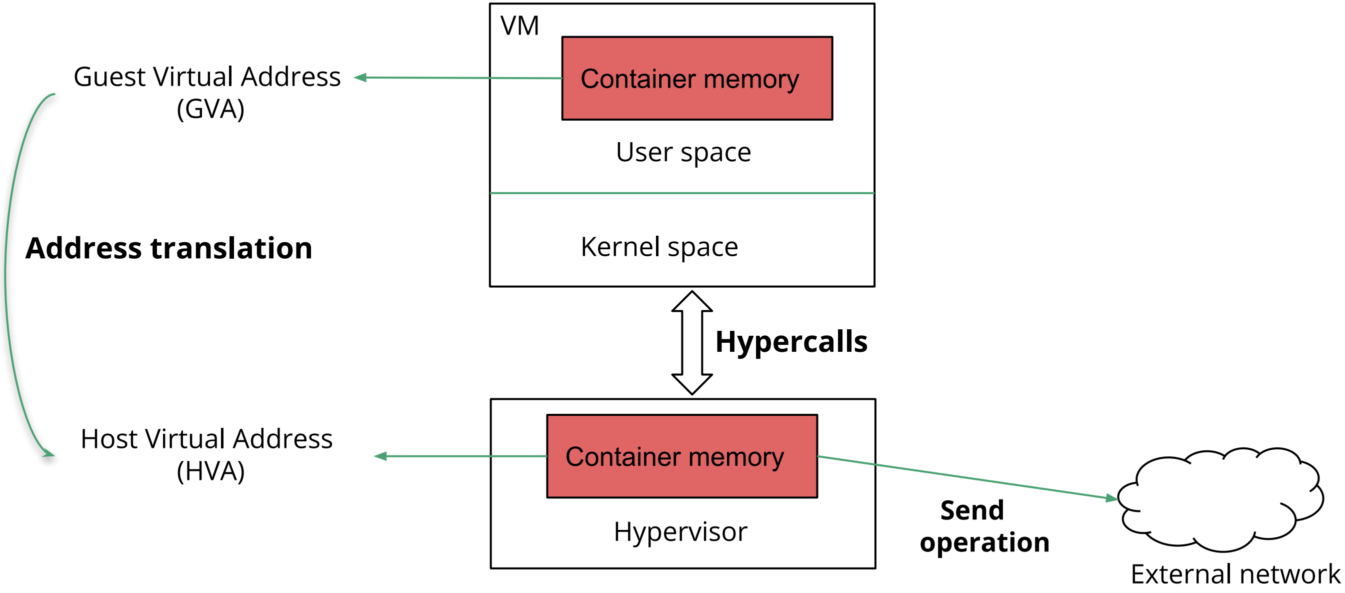


Portkey CRIU
Alternate implementation of network operations in user space of VM

Portkey VM Module
Forwards operations initiated by *Portkey CRIU* to the hypervisor using custom hypercalls

Portkey Data Transfer Agent
Performs network operations on behalf of CRIU

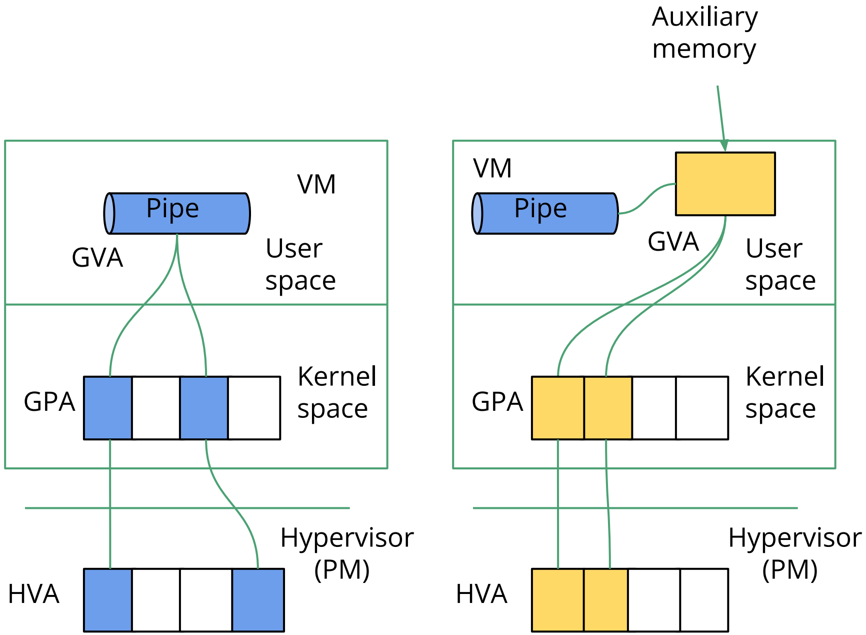
Overview of send mechanism



Challenges

- Reduce address translation overheads
  - Pre-allocated contiguous memory in the guest OS is used as auxiliary memory
- Avoid I/O blocking at the hypervisor
  - Used non-blocking network operations and error handling inside VM
- Reduce VM-hypervisor interaction
  - Estimate and provide delay between send operations inside VM
  - Send maximum amount of data per hypercall without breaking CRIU protocol

Fixed Mapping and Adaptive Send Rate



Non-contiguous memory in pipe

Contiguous fixed mapping

Contiguous fixed mapping requires single address translation (GVA→GPA→HVA)

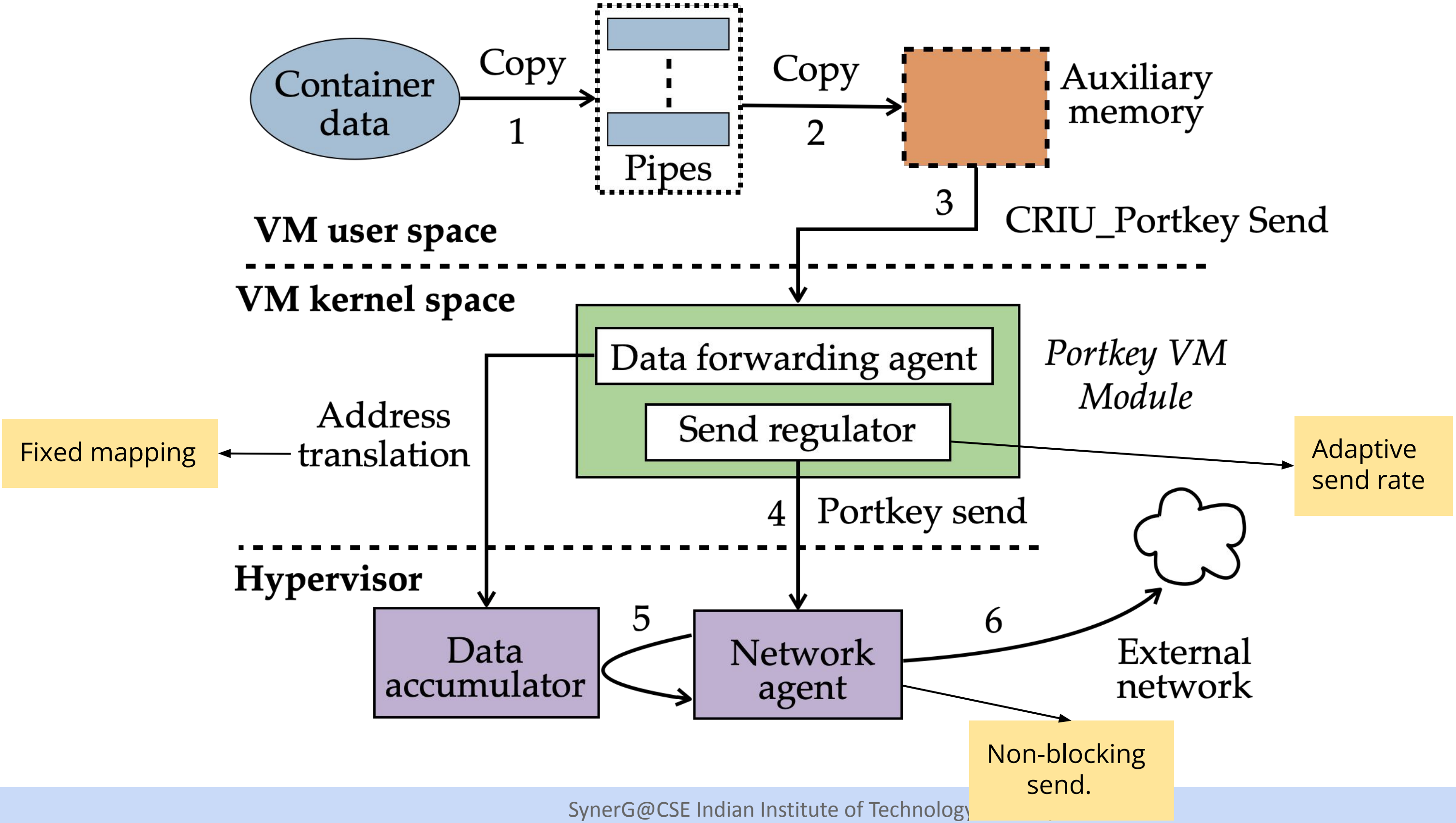
Adaptive send rate to reduce hypecall invocations

Portkey estimates available bandwidth at source PM
Adjusts delay between consecutive send operations

Available bandwidth = 1 Gbps, Data size = 4 MB,
Empty space in send buffer = 3 MB

Estimated delay = (4-3) MB/ 1 Gbps = 7.8 ms
(for 1 MB to be added to send buffer)

Portkey send mechanism



Container data

Copy

1

Pipes

Copy

2

Auxiliary memory

3

CRIU_Portkey Send

VM user space

VM kernel space

Data forwarding agent

Send regulator

Portkey VM Module

Address translation

Fixed mapping

Adaptive send rate

4

Portkey send

Hypervisor

Data accumulator

5

Network agent

6

External network

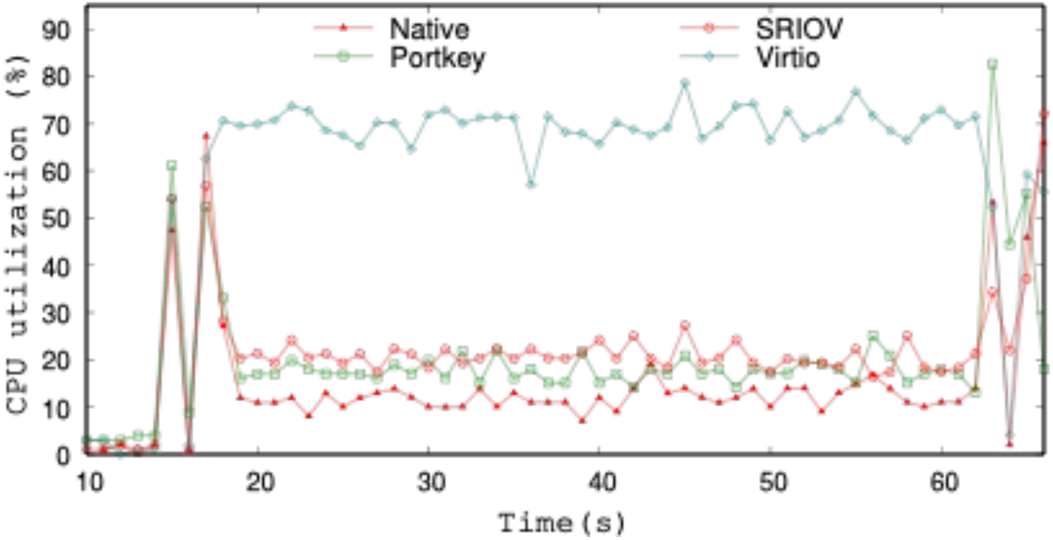
Non-blocking send.

Evaluation questions

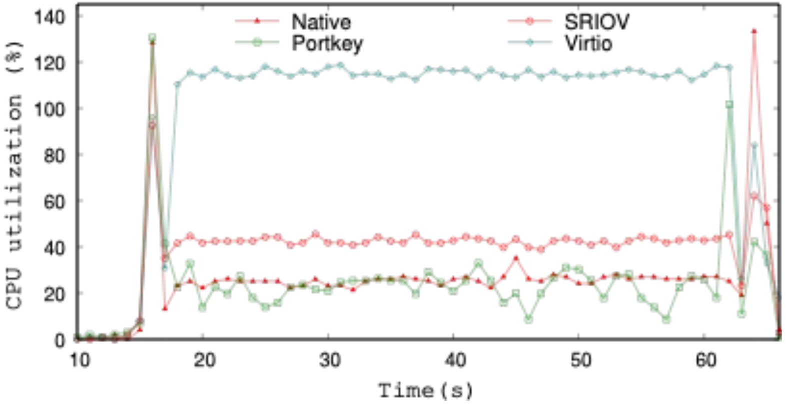
- How effective is Portkey in reducing CPU utilization, compared to virtio (with vhost-net kernel module) and SR-IOV?
- Does Portkey allocates saved CPU to applications (work conserving)?
- How effective is proposed adaptive send mechanism?
- What is the extent of impact of Portkey on the migration metrics (Predump time, Dump time, and performance of application under migration)?

Efficacy of Portkey in Ideal Condition

➢ **Ideal condition:** Migrate a quiescent container without any resource constraint.



CPU utilization at source PM



CPU utilization at destination PM

➢ CPU utilization is close to native setup in case of Portkey without impacting migration time

DoubleDecker: a cooperative disk caching framework for derivative clouds

Debadatta Mishra, Prashanth and Purushottam Kulkarni

18th ACM/IFIP/USENIX Middleware Conference 2017

dynamism in derivative clouds

resource overcommitment is the name of the game!

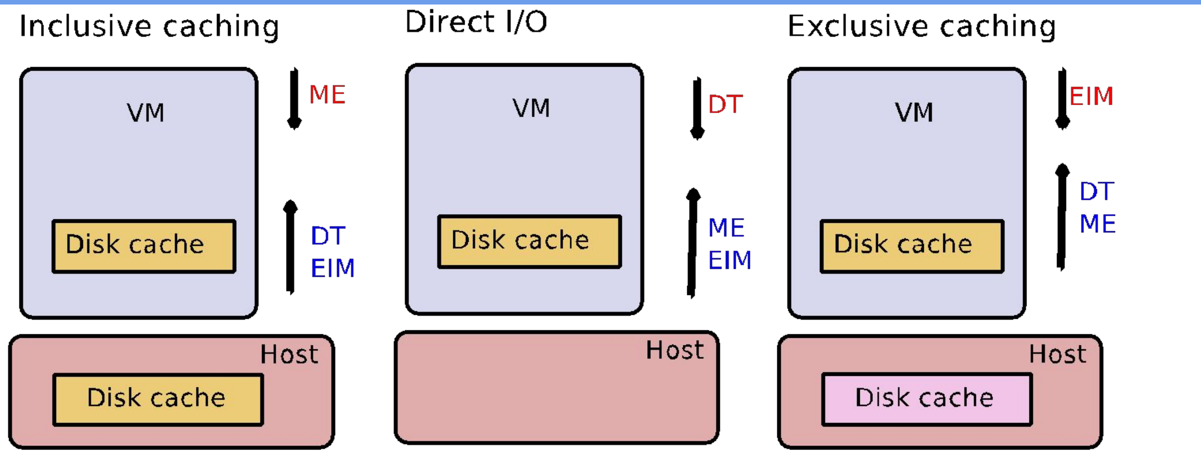
IaaS provider multiplexes resources (paging, ballooning, eviction, scheduling…)
to improve efficiency and performance requirements

challenges with derivative clouds

- for IaaS, VM is a black box, semantic gap about resource importance
  - which resources to reclaim? … different hypervisor and VM views

- derivative provider centric multiplexing policies (different from IaaS policies)

Disk caching and memory efficiency



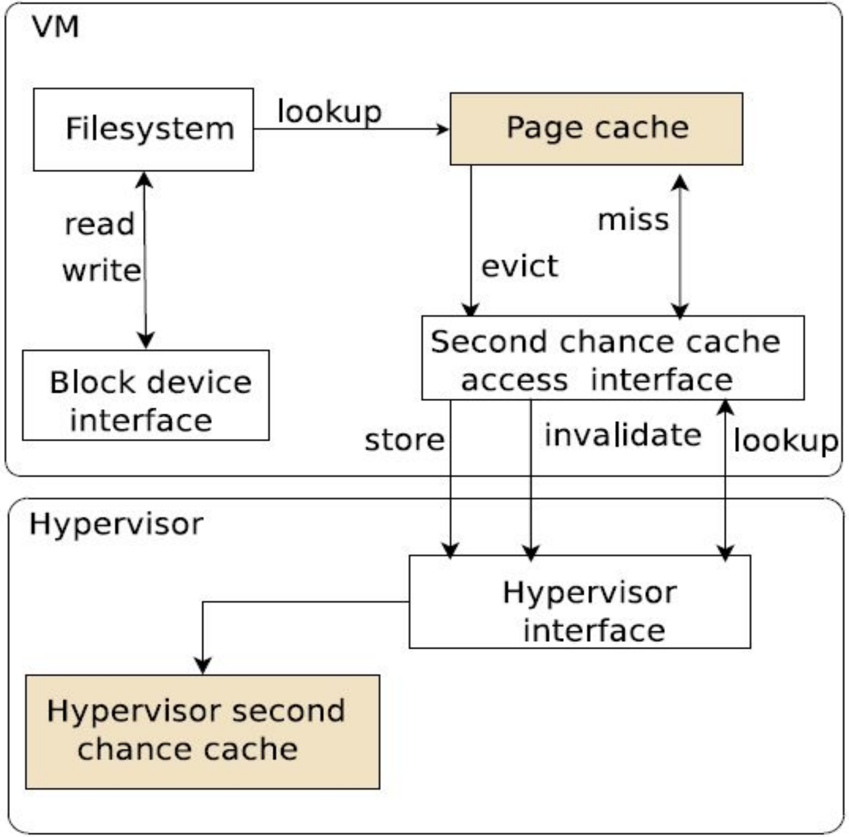
ME: Memory efficiency   DT: Disk throughput   EIM: Ease of implementation and management

Inclusive caching: Low memory efficiency

Direct IO: Low throughput

Exclusive caching: Additional (in-band or out-of-band) overhead

background: hypervisor (disk) caching



cleancache interface part of Linux VFS

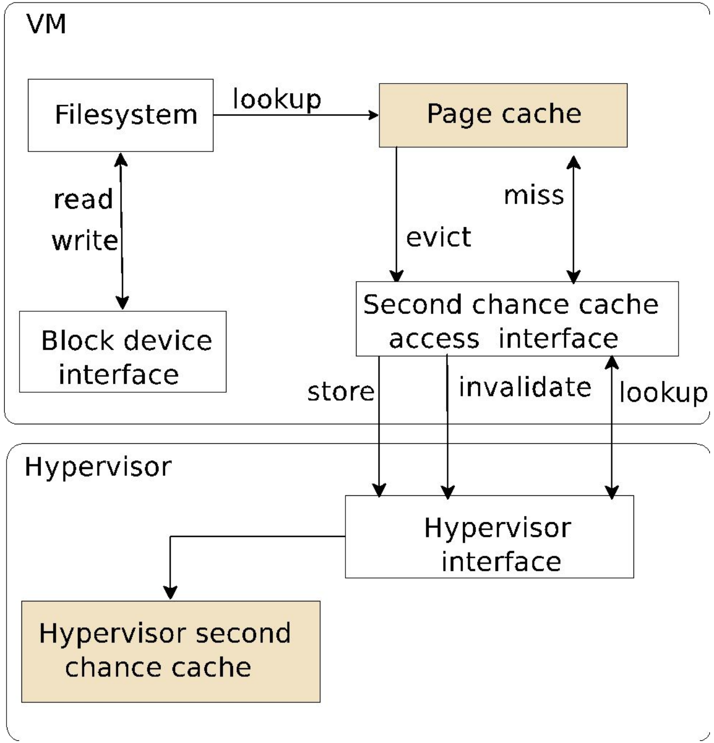
backend implementation extended to hypervisor --- the hypervisor cache

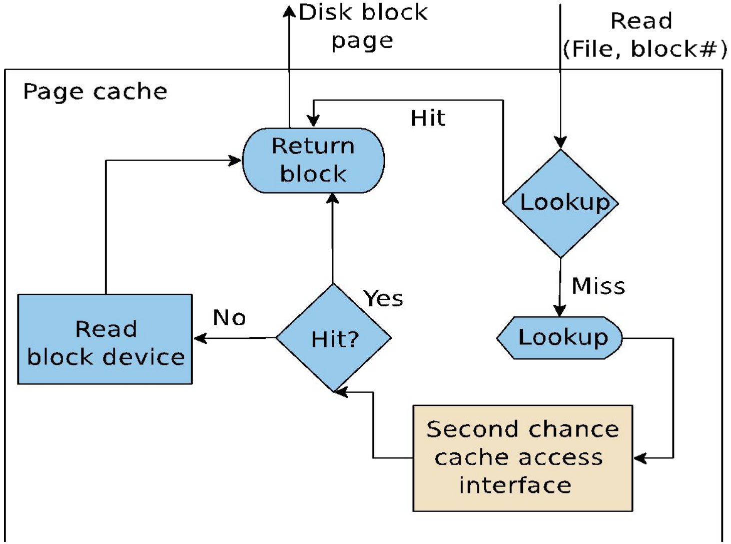
backend stores can be in-memory, SSD, over the network ...

basic mechanism for disk caching --- hypervisor caching

no support for nesting and cgroups

Hypervisor caching: Have a cache and eat it to!

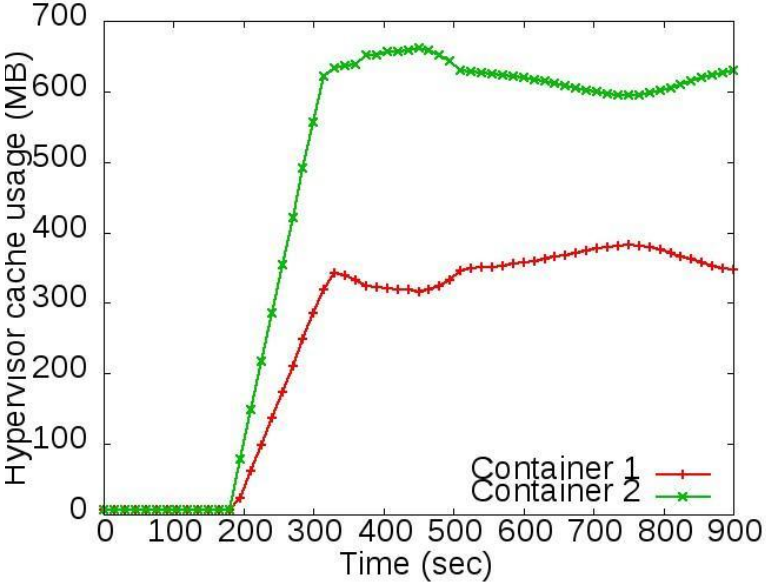


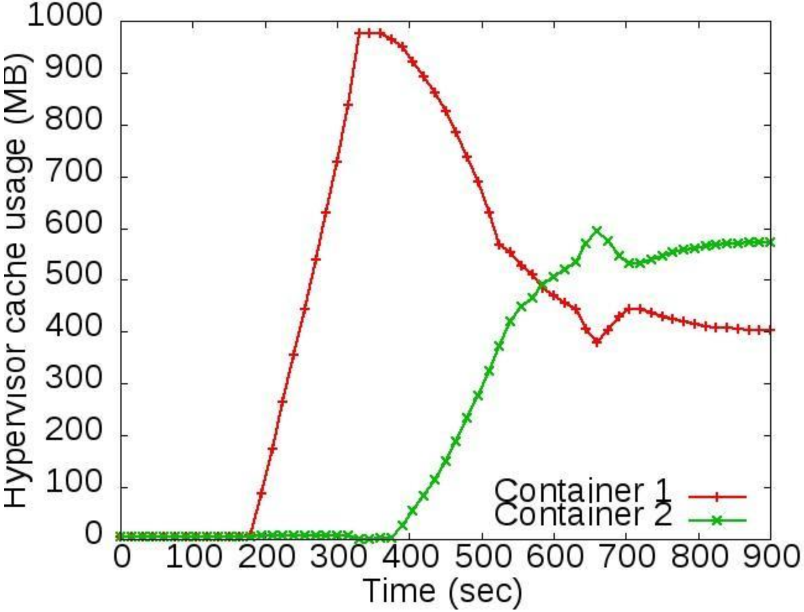


Integrated exclusive page cache management

Extend page cache and store only clean pages

non-determinism of hypervisor cache provisioning

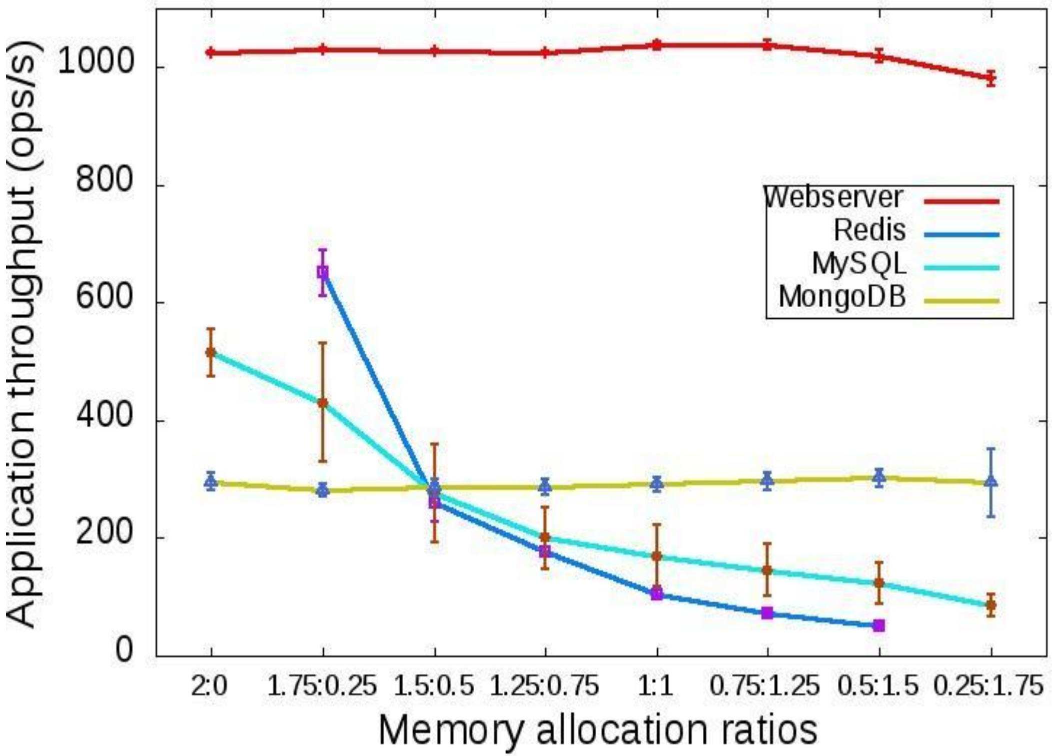




Filebench and webserver occupy cache based on workload characteristic and start times

No mechanism available to partition based on derivative end-points

application characteristics matter for cache distribution



application throughputs affected differently across splits of VM and hypervisor cache

webserver and mongoDB largely agnostic to split

Redis and MySQL prefer large in-VM cache

No mechanism to enforce these application-specific requirements

problem statement

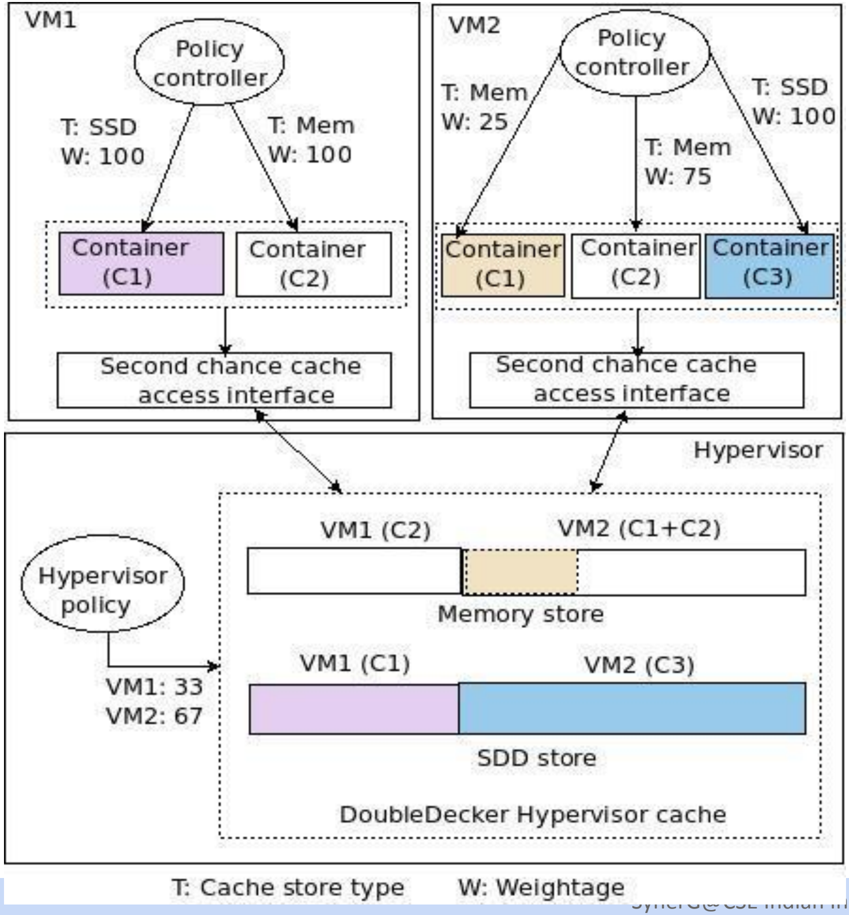
efficiently manage hypervisor disk caching resources with flexible policy support across the two levels in a derivative setup

- deterministic hypervisor cache partitioning
- support for differentiated policy enforcement

contributions

- mechanism for symbiotic disk caching between hypervisor and VM
- KVM+Linux based implementation for memory and SSD caches

doubledecker design



cache usage weight

- hypervisor level (across VMs)
- VM level (across containers)

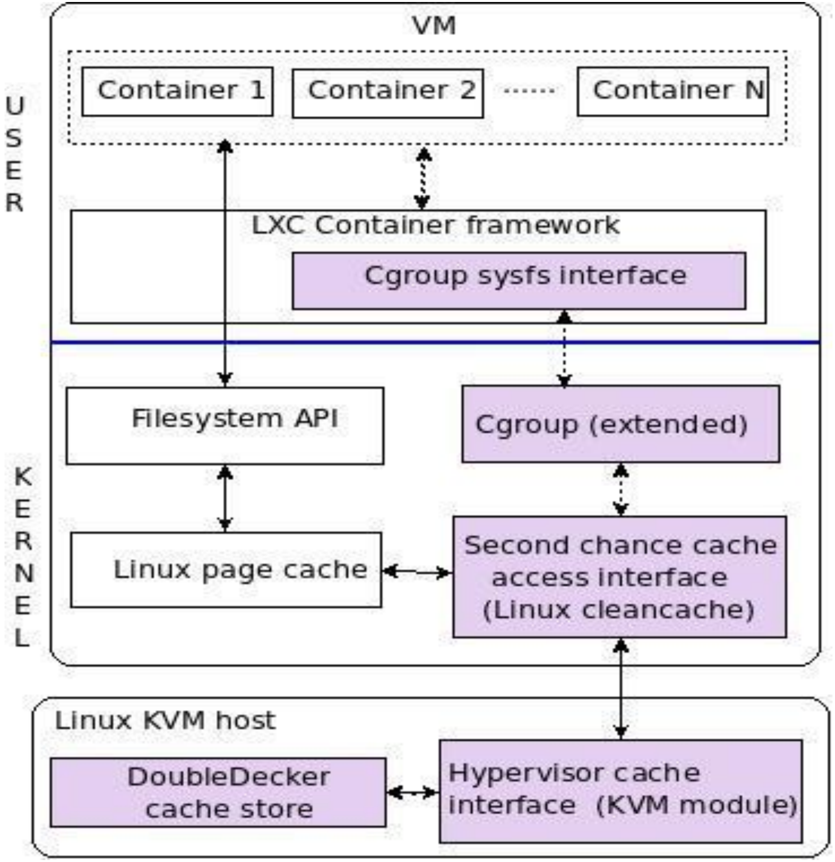
two tuple <T,W> configuration

- T: Cache type (Mem/SSD)
- W: Weight

dynamic reconfiguration possible

support for independent resource management at two level in multi-hosting setups

doubledecker implementation



extension of **cleancache** interface

cgroup integration (instead of FS)

new hypercall+state

(creation/deletion, updates to cache parameters, usage statistics)

cgroup extensions

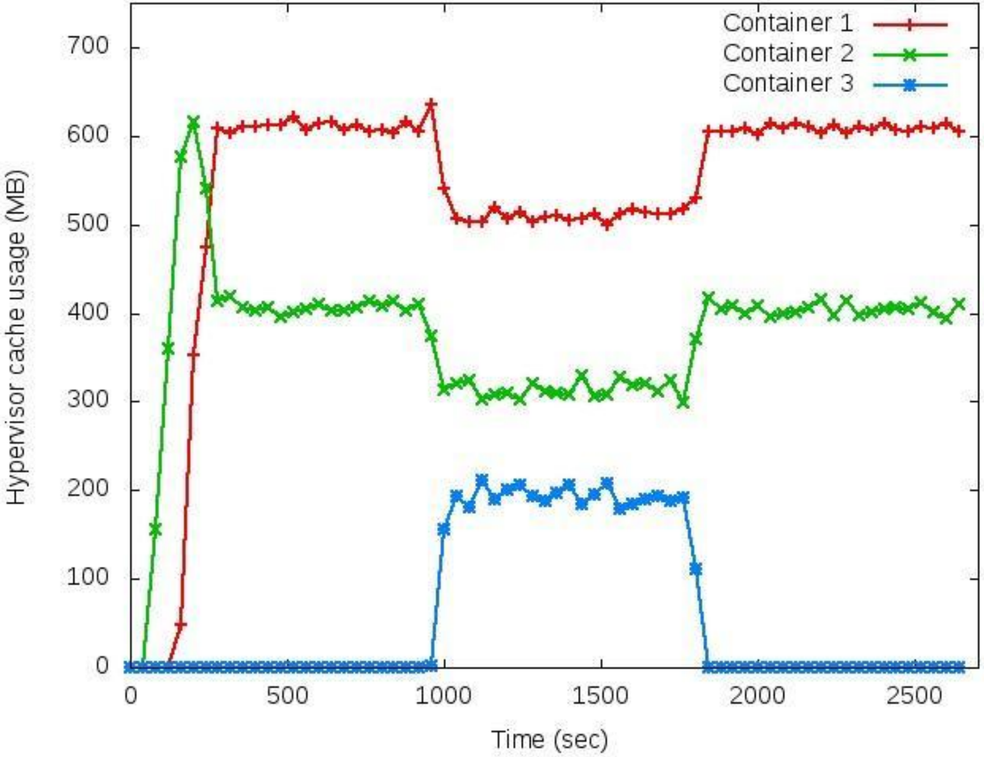
policy interface & **cleancache** integration

DoubleDecker cache

memory and SSD stores

dynamic policy enforcement

deterministic cache partitioning



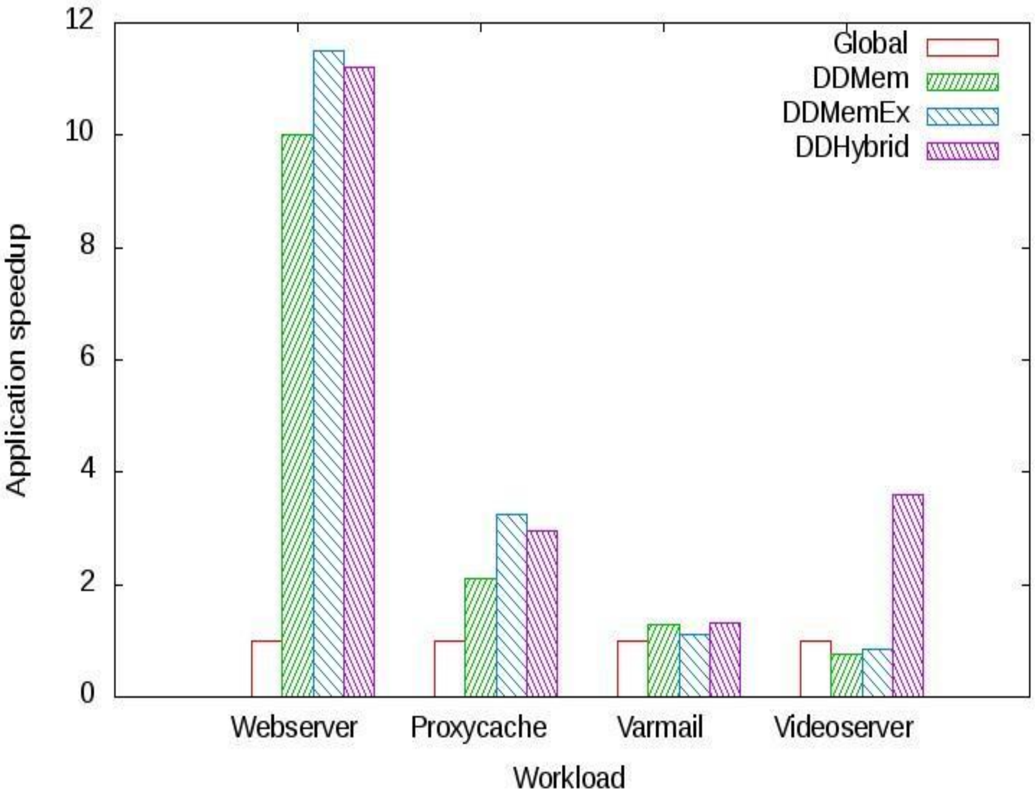
initial weights C1: C2 is 60:40

at 900s, C1:C2:C3 adjusted to 50:30:20

at around 1800s, C3 is move to SSD and C1:C2 re-configured at 60:40

hypervisor can implement dynamic nesting-aware level policies in a deterministic manner

does mem+ssd backend benefit?



four application containers & with different policies

Cache usage ratios

DDMem: <30, 25, 25, 15>

DDMemEx: <40, 30, 30, 0>

DDHybrid: <40, 30, 30, 100>

<SSD,100> for Videoserver

Policy alternatives provide better flexibility and performance

does nested partitioning help?

Workload	SLA requirement	Throughput (DD)	Throughput (Morai++)
MongoDB	**15 ops/sec**	25.1 ops/sec	16.9 ops/sec
MySQL	**100 ops/sec**	132.7 ops/sec	48.5 ops/sec
Redis	**500 ops/sec**	11186 ops/sec	13 ops/sec
Webserver	**900 ops/sec**	988 ops/sec	1289 ops/sec

Morai can only do aggregate VM-level partitioning

Doubledecker can use separate provisioning at both levels to explore larger provisioning space to meet SLAs

Deterministic container resource management in derivative clouds

Chandra Prakash, Umesh Bellur, Purushottam Kulkarni

IEEE Conference on Cloud Engineering IC2E 2018

Resource Management in Derivative Clouds

examples of nesting agnostic resource management by hypervisor in derivative setups

memory and CPU

mechanisms

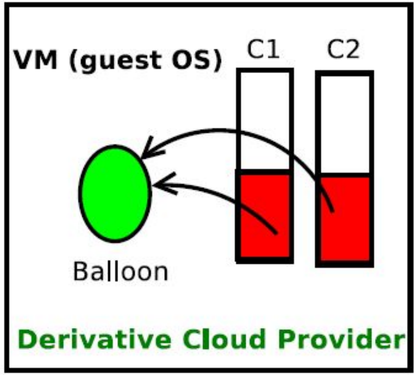
ballooning: memory overcommitment handling

vcpu scaling: cpu-granularity multiplexing

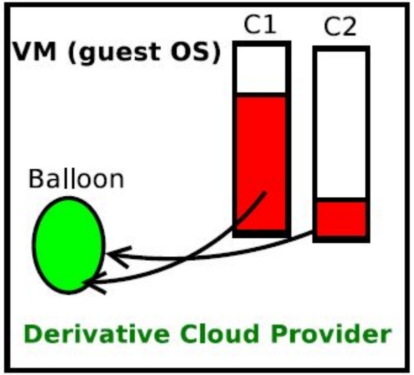
both techniques are nesting agnostic

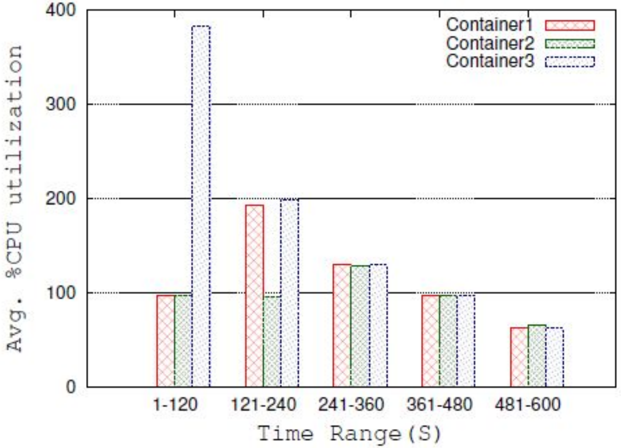
implications of nesting agnostic management

Agnostic memory reclamation



Desired reclamation





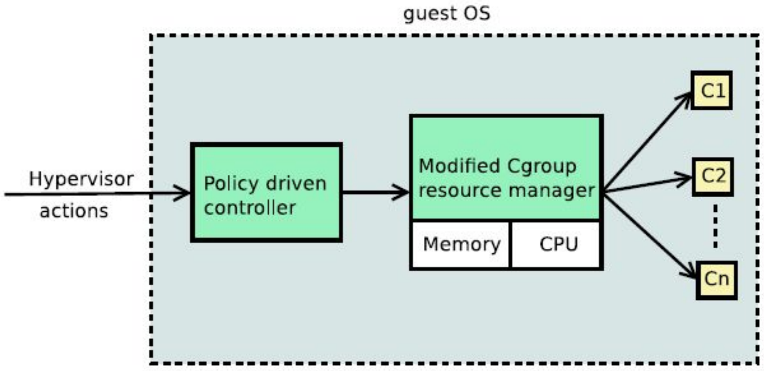
balloon inflation recovers pages from VM for hypervisor

balloon driver is VM-centric, not aware of nesting entities

1:1:4 desired CPU allocation ratio

CPU allocation ratios not maintained after scaling down

nesting-aware memory/cpu management

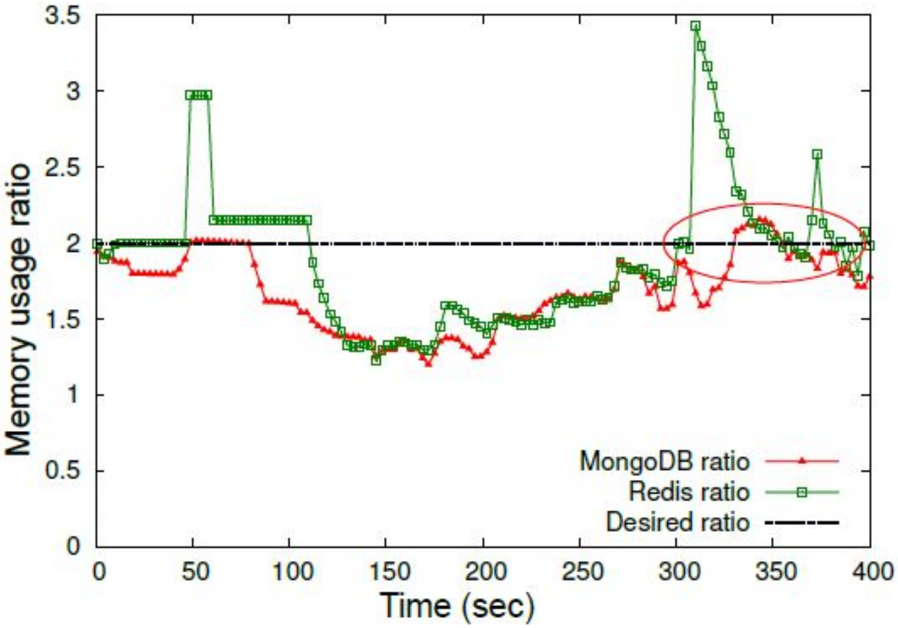


modified cgroup resource manager

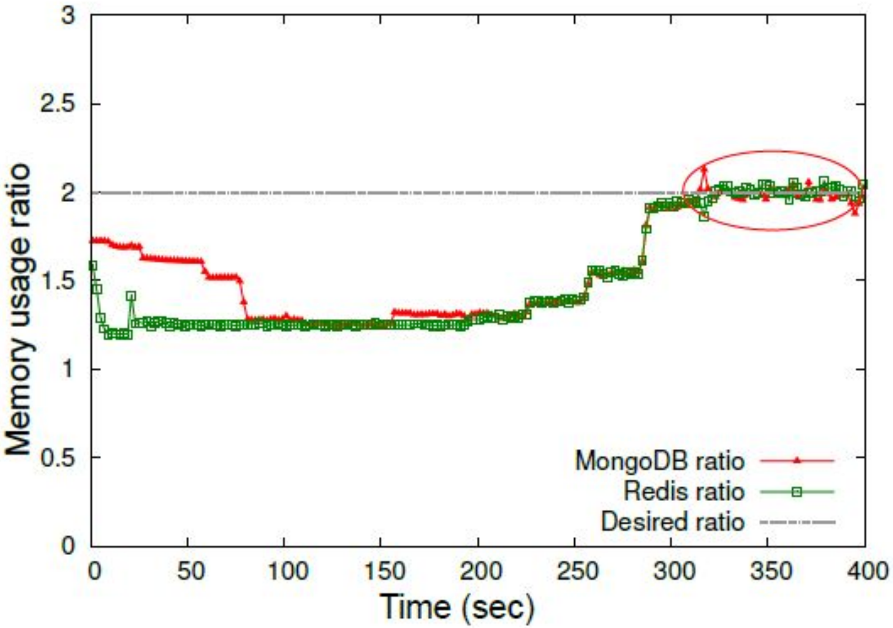
- proportionate memory provisioning and reclamation
- flag nested entities for no-reclamation

- update cpu allocation of cgroups (combination of pinning+sharing)

evaluation of nesting-aware memory allocation



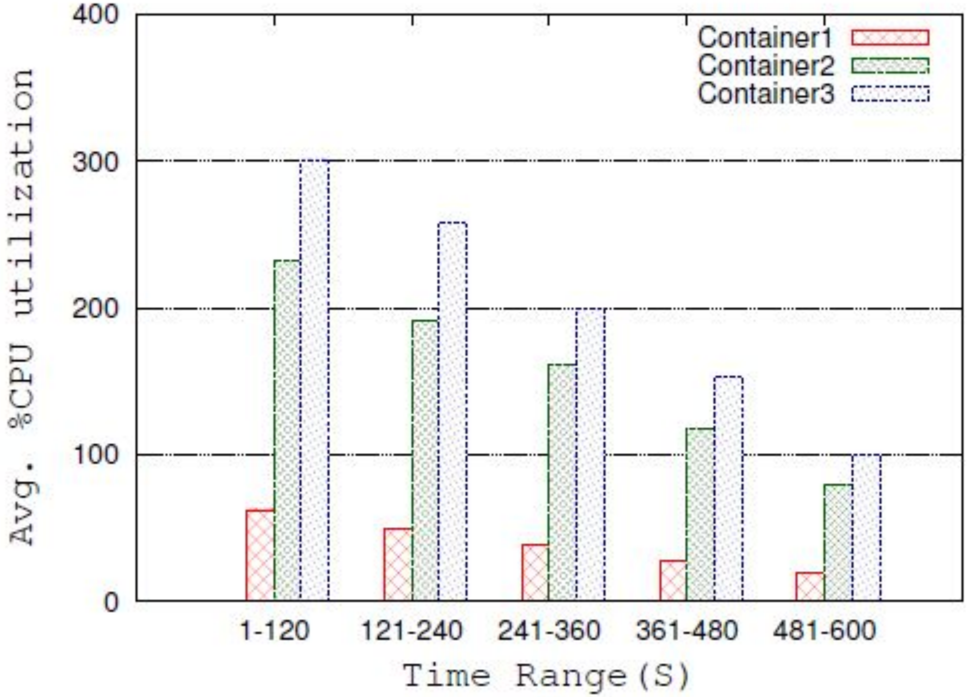
Memory usage ratio (default)



Memory usage ratio (with control)

2:1 ratio between MongoDB and Redis instances maintained

nesting aware CPU provisioning



CPU utilization by each container

Synergy: A Hypervisor Managed Holistic Caching System

Debadatta Misha, Purushottam Kulkarni

IEEE Transactions on Cloud Computing 2016

causes of memory usage inefficiency

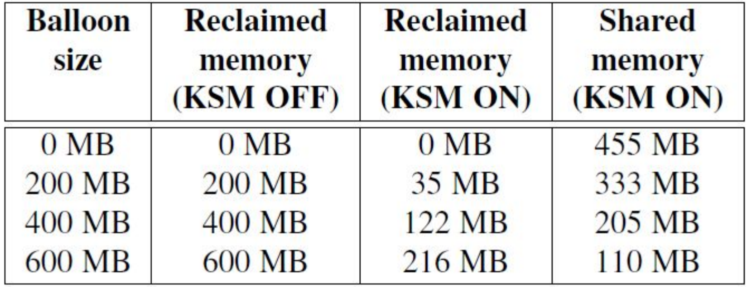
multiple/redundant copies of content in memory

- page/disk caches in VM and hypervisor
- multiple VMs with same OS/applications

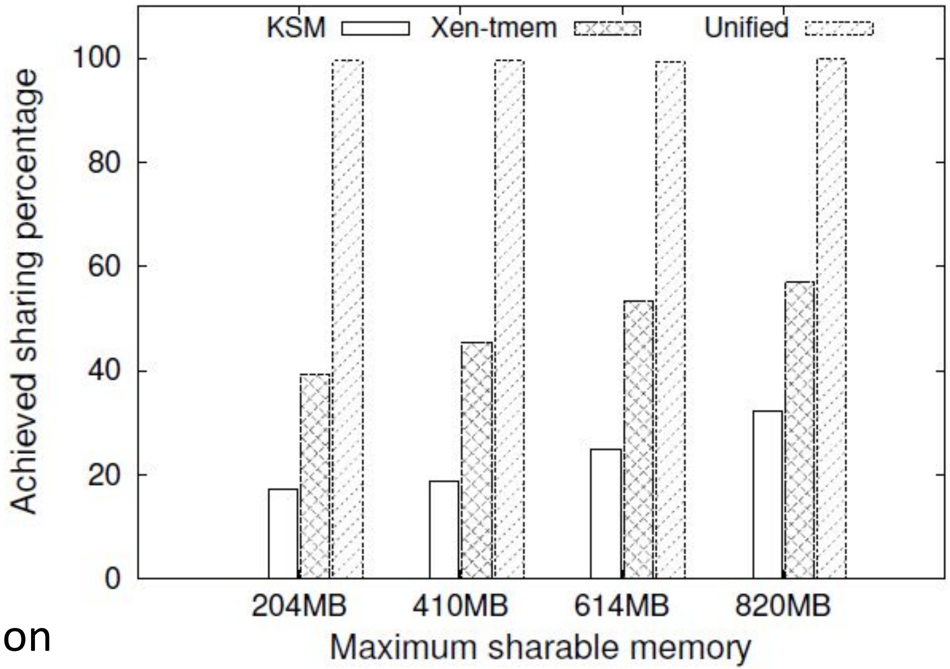
conflicting management mechanisms

- ballooning vs. sharing
- shared pages if ballooned have no effect
    - infact reduce sharing and decrease memory efficiency

examples of conflicting memory mgmt. actions



Balloon size	Reclaimed memory (KSM OFF)	Reclaimed memory (KSM ON)	Shared memory (KSM ON)
0 MB	0 MB	0 MB	455 MB
200 MB	200 MB	35 MB	333 MB
400 MB	400 MB	122 MB	205 MB
600 MB	600 MB	216 MB	110 MB

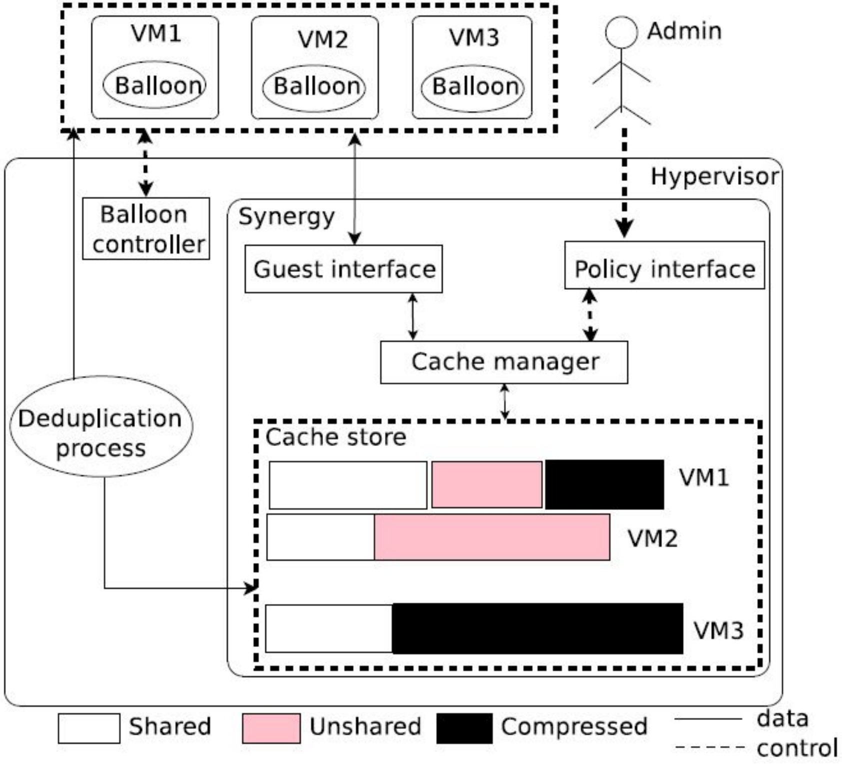


shared pages on reclamation allocate a new page!

no mechanism for system-wide deduplication

Synergy: A Hypervisor Managed Holistic Caching System

TCC 2016



(exclusive) hypervisor caching

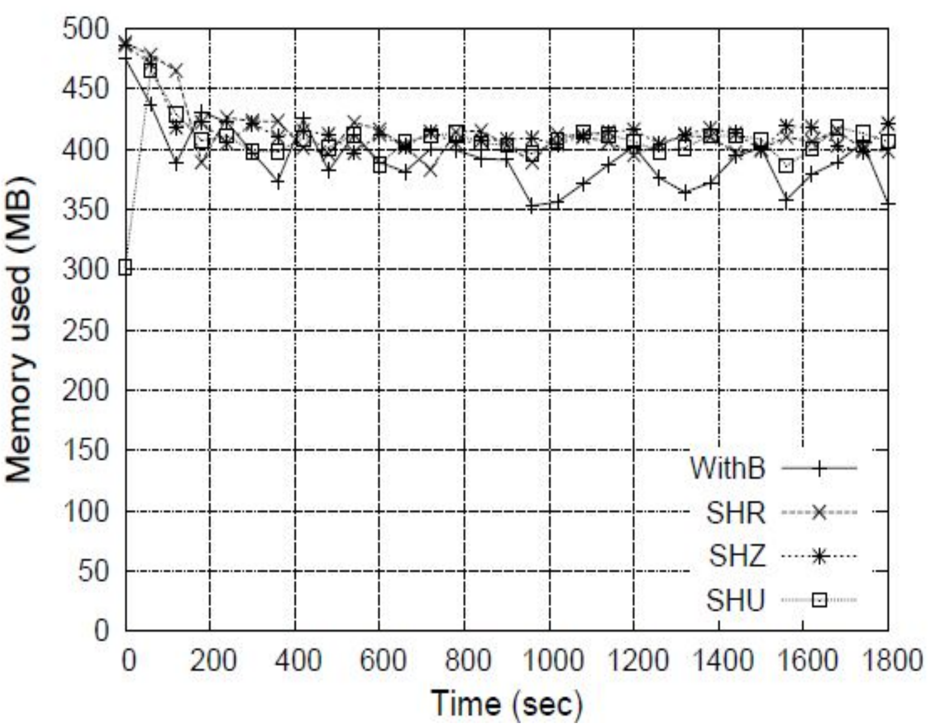
+

KSM (same page merging)

for

- retaining shared pages on ballooning
- system-wide deduplication of **all** memory
- system-wide memory provisioning

system-wide dedup with Synergy



balloon inflation/deflation across different VMs

memory utilization between 350 MB to 450 MB

Synergy resharing allows system-wide utilization to stay ~400 MB

Memory and IO efficiency related

Singleton: System-wide Page Cache Deduplication in Virtual Environments *HPDC 2012*

Share-o-meter: An empirical analysis of KSM based memory sharing in virtualized systems *HiPC 2013*

Comparative analysis of page cache provisioning in virtualized environments *MASCOTS 2014*

DRIVE: Using implicit caching hints to achieve disk I/O reduction in virtualized environments HiPC 2014

Per-VM page cache partitioning for cloud computing platforms Comsnets 2016

new virtualization mechanisms

dynamic reconfiguration of network endpoints

Vagabond: Dynamic network endpoint reconfiguration in virtualized environments
SoCC 2014

elastic SSD devices for IO caching

SymFlex: Elastic, Persistent and Symbiotic IO Caching in Virtualization Environments
(under submission)

record-replay framework

InSight: A Framework for Application Diagnosis using Virtual Machine Record and Replay

capacity planning and provisioning

understanding/modeling the VM migration mechanism

Resource Availability Based Performance Benchmarking of Virtual Machine Migrations (ICPE 2013)

Towards a comprehensive performance model of virtual machine live migration (SoCC 2015)

On Selecting the Right Optimizations for Virtual Machine Migration (VEE 2016)

provisioning and placement heuristics

Affinity-aware modeling of CPU usage with communicating virtual machines (JSS 2013, IEEE Cloud 2011)

Risk Aware Provisioning and Resource Aggregation based Consolidation of Virtual Machines (IEEE CLOUD 2012)

Dynamic Resource Management Using Virtual Machine Migrations (IEEE Communications Magazine, September 2012)

benchmarking tool

VirtPerf: A Capacity Planning Tool for Virtualized Environments (IEEE CLOUD 2011)

OS & hypervisor intersection

VM introspection based file system metadata and disk IO prefetching optimizations

Stepahead: Rethinking filesystem namespace translations (APSys 2016)

Prewarming of metadata caches of distributed file systems in virtualization environments (on-going)

acceleration-as-a-service **(on-going)**

GPU multiplexing mechanisms

Empirical analysis of hardware-assisted GPU virtualization (HiPC 2019)

managing GPU memory to increasing size of trainable neural networks

Dynamic Memory Management for GPU-based training of Deep Neural Networks (IPDPS 2019)

offload hypervisor management tasks to GPU

Catalyst: GPU-assisted rapid memory deduplication in virtualization environments (VEE 2017)

FaaSter: Fast FaaS using heterogeneous GPUs

(HiPC 2021)

Optimizing Goodput of Real-time Serverless Functions using Dynamic Slicing with vGPUs (IC2E 2021)

Serverless computing/FaaS **(on-going)**

FaaS — function as a service

new abstraction from service provisioning

further decouples service usage from provisioning/management etc.

multiplexing, scheduling

integration with GPUs

smartnic offload

data pipelines for FaaS workflows

serverless workflow application development infrastructure

tools, prototypes, solutions …

design-build-experiment-repeat

We are hiring!

puru@cse.iitb.ac.in

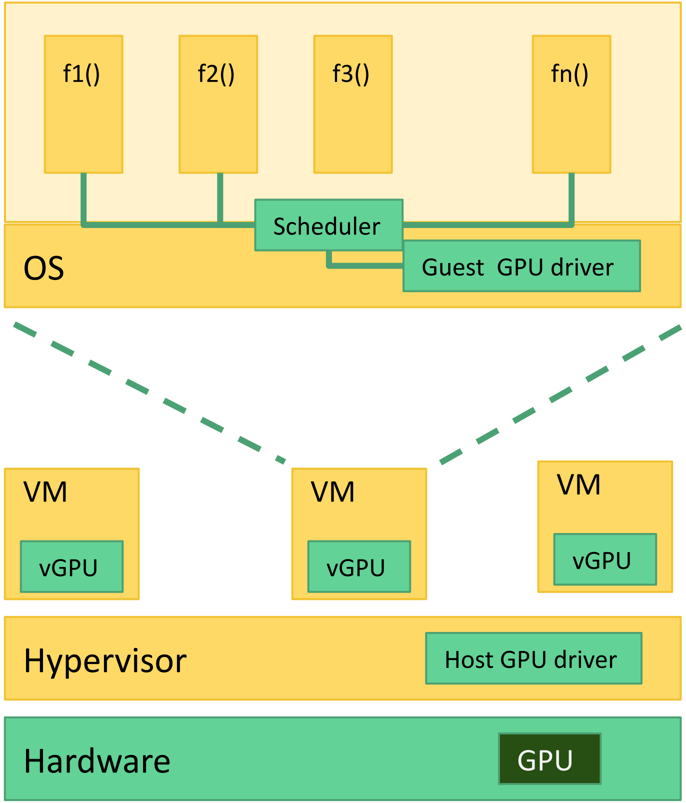
https://www.cse.iitb.ac.in/~puru

Optimizing Goodput of Real-time Serverless Functions using Dynamic Slicing with vGPUs

Chandra Prakash, Anshuj Garg, Umesh Bellur, Purushottam Kulkarni

IEEE International Conference on Cloud Engineering (IC2E 2021)

FaaS meets GPU



FaaS --- Function as a service

GPUs are candidates for parallelizing work and meet function execution deadlines

- ML training using GPUs
- Processing of images at scale

(editing, resizing, transcoding,classification)

Hosting setup

- VMs execute functions in containers
- H/W assisted vGPU multiplexing

(NVIDIA Tesla series)

Problem description

In nested setups (containers in VM),

vGPU scheduler in VM supports round-robin and FCFS scheduling

vGPUs scheduled using fixed share, equal share or best-effort mechanisms

deadline agnostic!

Determine task size and scheduling order of functions to *maximize* number of functions that complete within deadline

Functions (tasks) are not arbitrarily preemptible on GPUs

vGPU capacity is based on work across VMs and is dynamic

Solution components

1. Kernel slicing and scheduling mechanism
   - Smaller task sizes for generating scheduling events

2. GPU capacity estimator
   - Capacity of GPU is a function of load offered by all VMs
   - Dynamic loads, result in dynamic available capacity

3. Slice size selection + task scheduling
   - **Offline heuristic** (modified-EDF with adaptive slice sizes)
   - Online heuristic
   - Metrics:
     i. #tasks completed before deadline
     ii. Minimizing wasted work on GPUs

work-in-progress

FaaSter: Fast FaaS using heterogeneous GPUs

Anshuj Garg, Shahrukh Hussain, Sriram Y, Riya Baviskar
Purushottam Kulkarni, Umesh Bellur

IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC 2021)

Acceleration-as-a-service

Problem description

Context

Provide a library of functions to users via the **Function-as-a-Service** model

The FaaS services relies on GPU backends for compute (image processing, training, mathematical functions etc.)

Resource assumption: ***Heterogeneous*** GPU types

Goal

Build a FaaS framework for exploiting heterogeneous GPU backends

Map and schedule function requests to appropriate GPUs to minimize job completion times and maximize GPU resource utility

FaaSter architecture

1. **Function Library**

   ⇒ Multi-API implementation of functions

2. **Dispatch mechanism/logic**

   ⇒ Multiplex an invocation to one of the many backends/hardwares

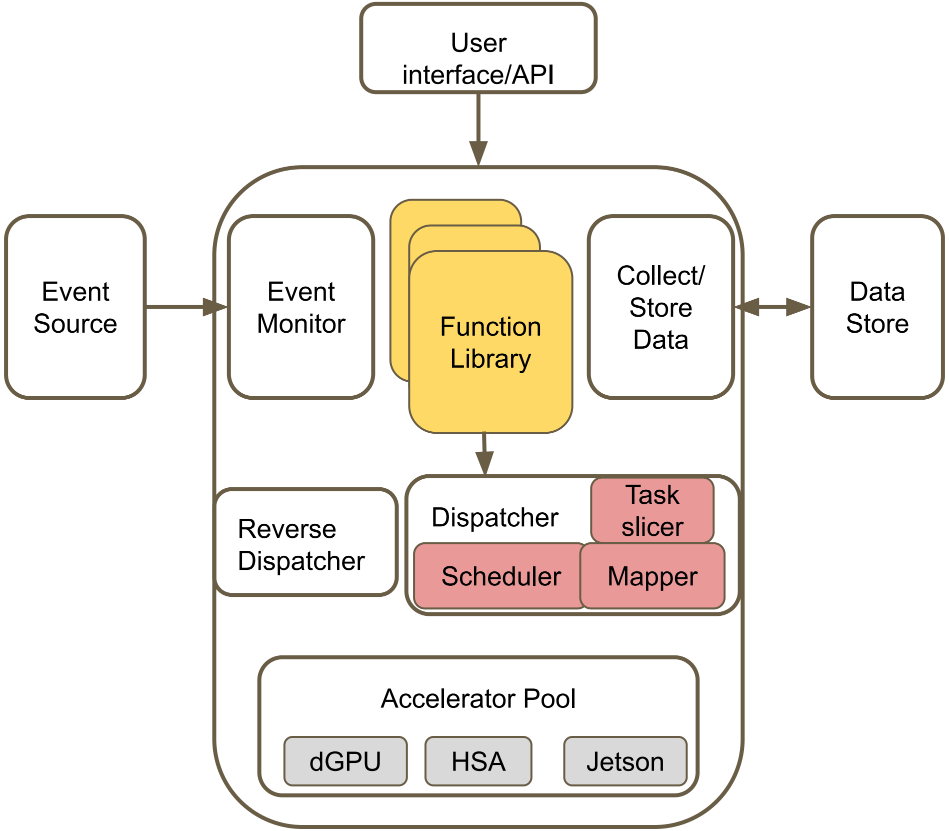
   ⇒ Decision for multiplexing

3. **Notification mechanism**

   ⇒ Events/Triggers

4. **API Usage setup**

   ⇒ how does user invoke the FaaS functionality?



FaaSter solution components

1. Function profiling across multiple GPUs
   a. At different slice and input sizes

2. Engineering the end-to-end runtime with all components

3. Design of dispatch logic for high throughput of completed tasks

   Decision dependent on

   i. current and queued up load at GPUs
   ii. function execution characteristics on GPUs
   iii. function amenability to slicing

Takeaways

Acceleration-as-a-service is a first-class service!

Several unique problems at the intersection of cloud systems and acceleration platforms

Problems across the cloud stack

management systems, OS extensions,

APIs for networked applications, building scalable applications,

acceleration hardware usage and integration …

New and demanding workloads

IoT, ML, phone and mobile computing, robotics and automation,

virtual desktops with GPUs, …

… set to to consume the acceleration services