Elastic Persistent Memory for Virtualized Environments

Bhavesh Singh
143059003

June 27, 2017

Guided by
Prof. Purushottam Kulkarni

Outline

Introduction

- Persistent memory: Non-volatile memory with access latencies between HDD and DRAM

Introduction

- Persistent memory: Non-volatile memory with access latencies between HDD and DRAM
  - Block-addressable e.g. flash memory

Introduction

- Persistent memory: Non-volatile memory with access latencies between HDD and DRAM
  - Block-addressable e.g. flash memory
  - Byte-addressable (also storage class memory) e.g. NVRAM

Performance Characteristics of Persistent Memory

	Read Latency	Write Latency	Erase Latency	Endurance
HDD	5 ms	5 ms	N/A	$> 10^{15}$
SLC Flash	25 μs	500 μs	2 ms	$10^4 - 10^5$
PCM	50 ns	500 ns	N/A	$10^8 - 10^9$
DRAM	50 ns	50 ns	N/A	$> 10^{15}$
STT-RAM	10 ns	50 ns	N/A	$> 10^{15}$
ReRAM	10 ns	50 ns	N/A	10^{11}

Table: Comparison of metrics of different memory types [9]

Uses of Persistent Memory

- As replacement of slower memory

Uses of Persistent Memory

- As replacement of slower memory
  - File systems for flash memory [7]

Uses of Persistent Memory

- As replacement of slower memory
  - File systems for flash memory [7]
  - Replacement of DRAM

Uses of Persistent Memory

- As replacement of slower memory
  - File systems for flash memory [7]
  - Replacement of DRAM
- As another tier in memory hierarchy

Uses of Persistent Memory

- As replacement of slower memory
  - File systems for flash memory [7]
  - Replacement of DRAM
- As another tier in memory hierarchy
  - Block cache [4, 6]

Uses of Persistent Memory

- As replacement of slower memory
  - File systems for flash memory [7]
  - Replacement of DRAM
- As another tier in memory hierarchy
  - Block cache [4, 6]
  - Data structures in NVRAM [5]

Problem Statement

Two questions in the context of persistent memory

Problem Statement

Two questions in the context of persistent memory

- Is there a case for virtualizing persistent memory?

Problem Statement

Two questions in the context of persistent memory

- Is there a case for virtualizing persistent memory?
- How can we efficiently overcommit/multiplex this resource?

Motivation: Why Virtualize Persistent Memory?

Three aspects to the answer

Motivation: Why Virtualize Persistent Memory?

Three aspects to the answer

- Make the existing use-cases work in a virtualized environment

Motivation: Why Virtualize Persistent Memory?

Three aspects to the answer

- Make the existing use-cases work in a virtualized environment
  - Specific use-cases not needing virtualization e.g. hypervisor-managed block cache

Motivation: Why Virtualize Persistent Memory?

Three aspects to the answer

- Make the existing use-cases work in a virtualized environment
  - Specific use-cases not needing virtualization e.g. hypervisor-managed block cache
  - But not all use-cases can be mapped like this

Motivation: Why Virtualize Persistent Memory?

Three aspects to the answer

- Make the existing use-cases work in a virtualized environment
  - Specific use-cases not needing virtualization e.g. hypervisor-managed block cache
  - But not all use-cases can be mapped like this
- Higher cost per gigabyte for persistent memory

Motivation: Why Virtualize Persistent Memory?

Three aspects to the answer

- Make the existing use-cases work in a virtualized environment
  - Specific use-cases not needing virtualization e.g. hypervisor-managed block cache
  - But not all use-cases can be mapped like this
- Higher cost per gigabyte for persistent memory
- Limited number of write-erase cycles (flash memory)

Motivation: Why Virtualize Persistent Memory?

Three aspects to the answer

- Make the existing use-cases work in a virtualized environment
  - Specific use-cases not needing virtualization e.g. hypervisor-managed block cache
  - But not all use-cases can be mapped like this
- Higher cost per gigabyte for persistent memory
- Limited number of write-erase cycles (flash memory)

Multiplexed usage would be better than static partitioning of SSDs.

Related Work: Hasn't Anyone Tried This?

- Liang et al. [8] propose virtualization of NVRAM applying memory ballooning techniques

Related Work: Hasn't Anyone Tried This?

- Liang et al. [8] propose virtualization of NVRAM applying memory ballooning techniques
- Amazon EC2 provides Instance Store [3], static partitions in VMs based on SSDs attached to the same physical machine

Related Work: Hasn't Anyone Tried This?

- Liang et al. [8] propose virtualization of NVRAM applying memory ballooning techniques
- Amazon EC2 provides Instance Store [3], static partitions in VMs based on SSDs attached to the same physical machine
  - Provide strictly ephemeral semantics

Memory Ballooning

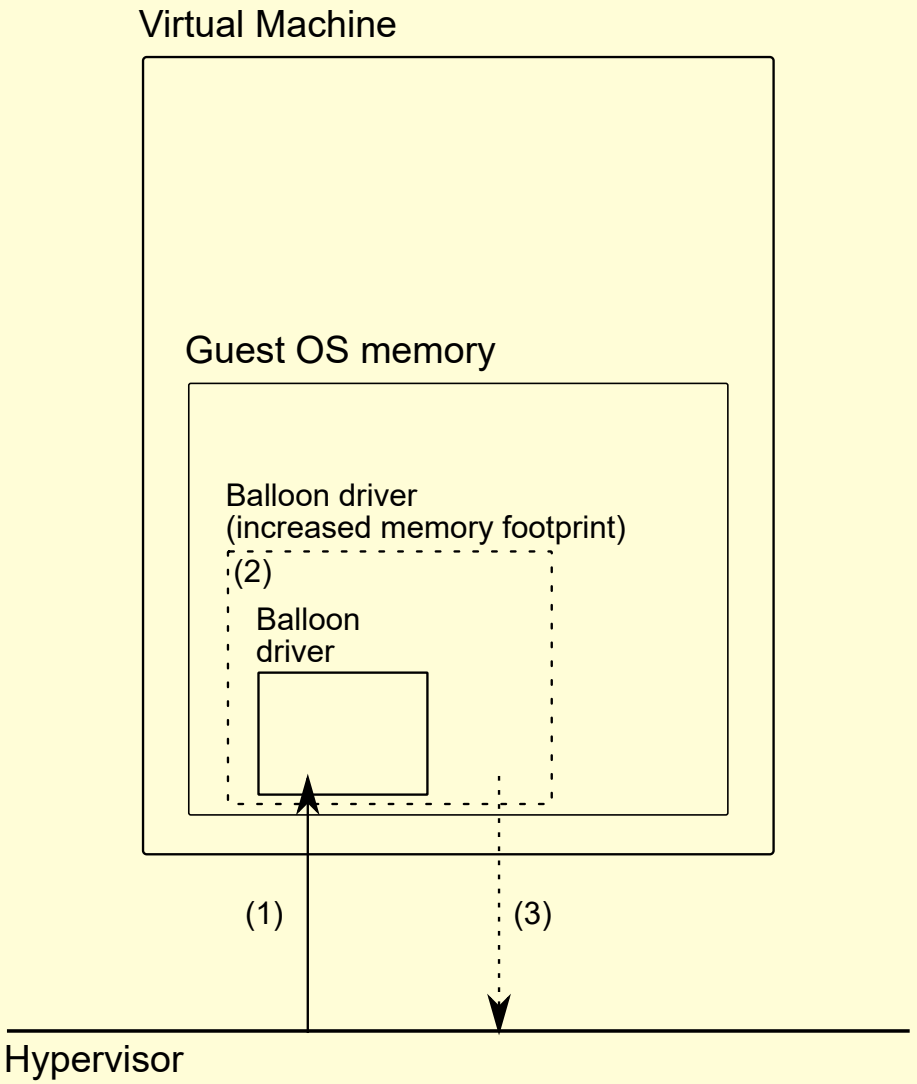

Figure: Memory Ballooning

Outline

Design

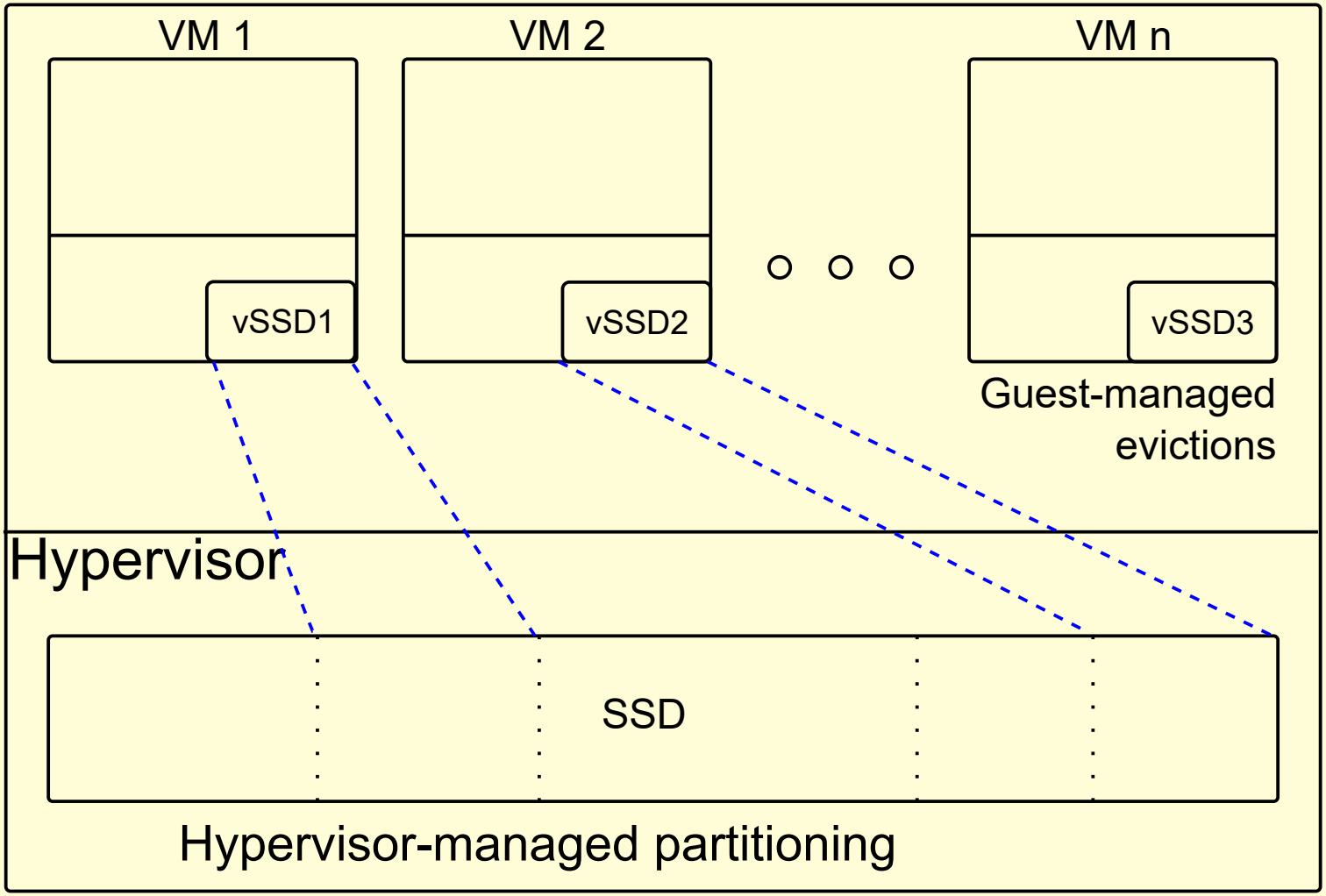

Figure: Our Proposed Solution

Design

We have three major components

- A virtual device, which we call vSSD; shows up as a ‘resizable’ block device in the guest

Design

We have three major components

- A virtual device, which we call vSSD; shows up as a ‘resizable’ block device in the guest
- A mechanism to resize the device dynamically from the hypervisor

Design

We have three major components

- A virtual device, which we call vSSD; shows up as a ‘resizable’ block device in the guest
- A mechanism to resize the device dynamically from the hypervisor
- An interface to ask the application using this device, which block numbers to evict in case of device shrink

Design: The Resize Interface

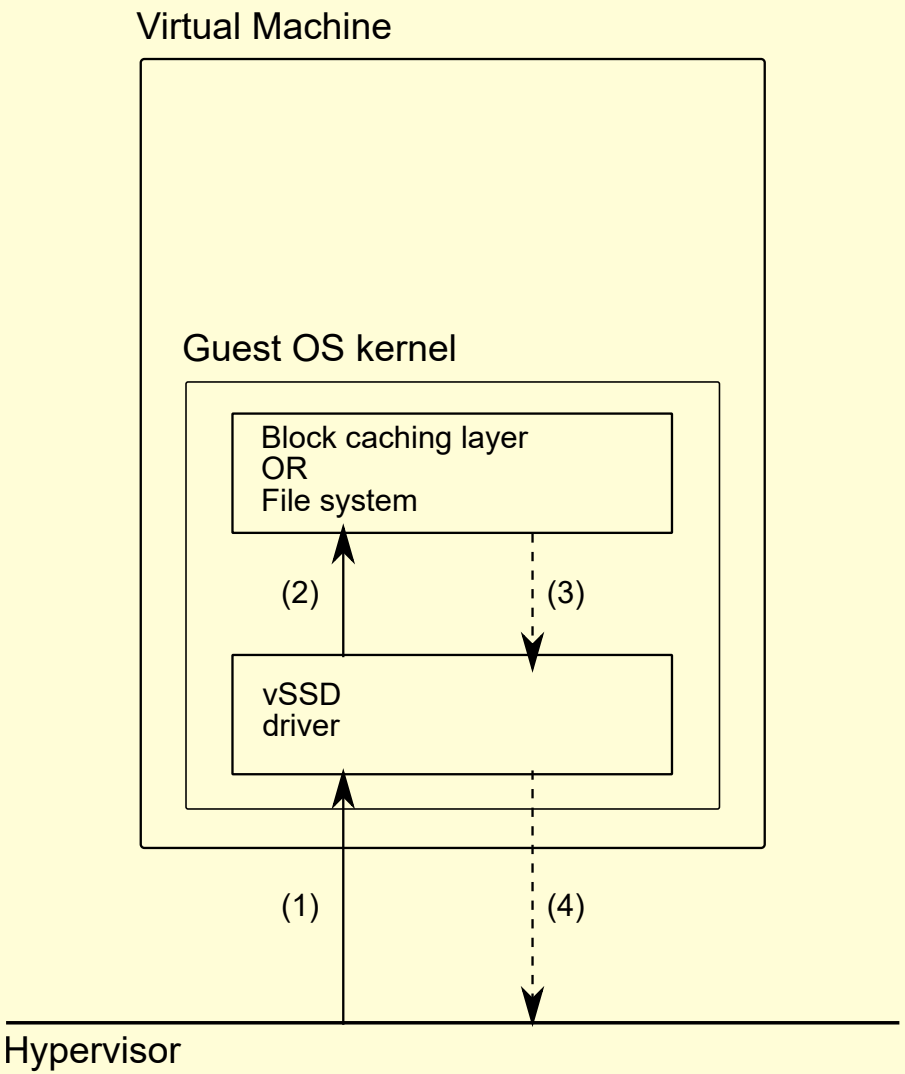

Figure: The resize interface

Resize Steps: Shrink

(1) Give me `n` blocks!

(2) Hey, which blocks should I give?

(3) Here, give $\{x_1, x_2, \ldots x_n\}$

(4) Here you go, you can take back these blocks $\{x_1, x_2, \ldots x_n\}$

Both the hypervisor and the guest maintain a list of valid block numbers for the device.

The steps for 'grow'ing the size are also similar.

Implementation: Background

- QEMU (Quick EMUlator) is a full system emulator for the Linux platform

Implementation: Background

- QEMU (Quick EMUlator) is a full system emulator for the Linux platform
- Uses `kvm` (kernel virtual machine) to leverage hardware virtualization support and provide full virtualization in Linux

Implementation: Background

- QEMU (Quick EMUlator) is a full system emulator for the Linux platform
- Uses `kvm` (kernel virtual machine) to leverage hardware virtualization support and provide full virtualization in Linux
- Virtio is a paravirtualized device driver model for common drivers like disk, network etc.

Implementation: Background

- QEMU (Quick EMUlator) is a full system emulator for the Linux platform
- Uses `kvm` (kernel virtual machine) to leverage hardware virtualization support and provide full virtualization in Linux
- Virtio is a paravirtualized device driver model for common drivers like disk, network etc.
- Provides better I/O throughput than fully emulated I/O.

Implementation: Components

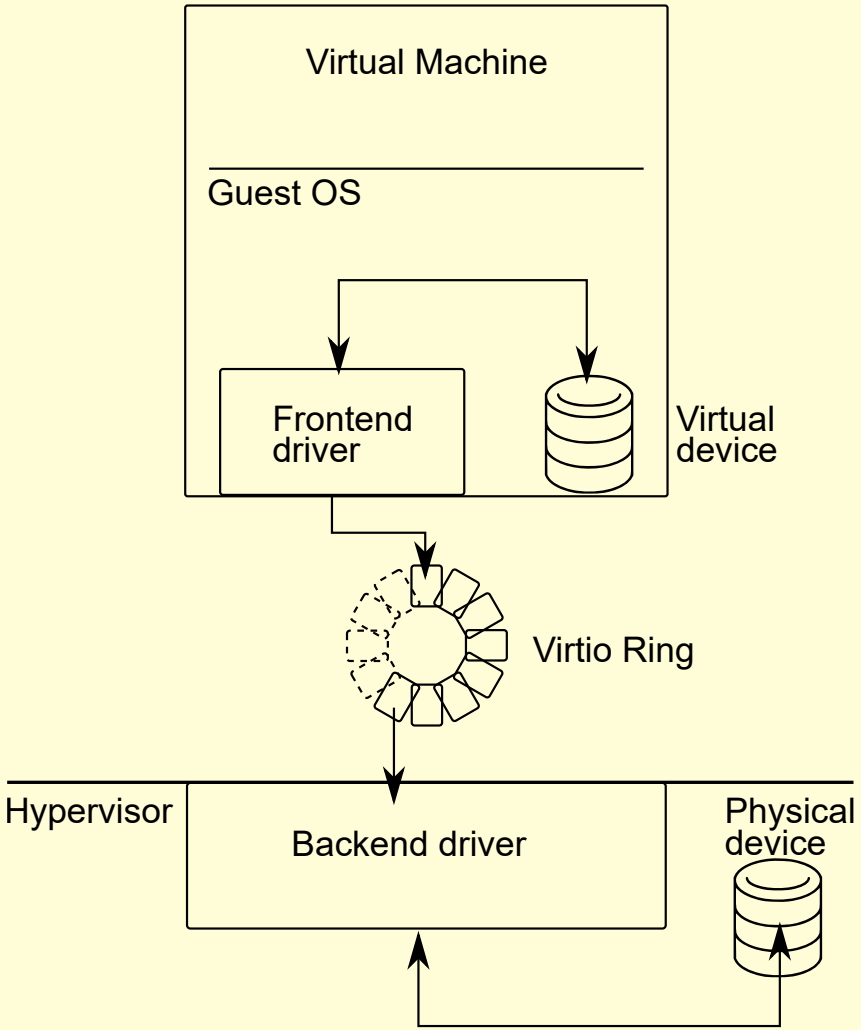

Figure: Virtio split drivers and Virtio ring; icons taken from [1] and [2]

Implementation: Components

- A backend device driver for our new ‘resizable’ disk in QEMU 2.8.0

Implementation: Components

- A backend device driver for our new ‘resizable’ disk in QEMU 2.8.0
- A frontend device driver in the Linux 4.9.14 that registers this device as a block device

Implementation: Components

- A backend device driver for our new ‘resizable’ disk in QEMU 2.8.0
- A frontend device driver in the Linux 4.9.14 that registers this device as a block device
- A proc file interface using which a userspace process can give hints about which blocks to evict

Implementation: Components

- A backend device driver for our new ‘resizable’ disk in QEMU 2.8.0
- A frontend device driver in the Linux 4.9.14 that registers this device as a block device
- A proc file interface using which a userspace process can give hints about which blocks to evict
- An interface to the user to give the resize command to the hypervisor for a VM’s vSSD device.

Outline

Experimental Evaluation

Setup

- A single VM
- 2 GB RAM
- 40 GB HDD (image file)
- 2 CPUs
- vSSD 1GB (for correctness experiments); 32 GB (for measuring virtualization overhead)

Experiment: Correctness

Question

Does vSSD resize (both shrinking and growing) happen correctly?

Experiment: Correctness

Question

Does vSSD resize (both shrinking and growing) happen correctly?

Parameter

Resize command with number of blocks to be evicted or given back

Experiment: Correctness

Question

Does vSSD resize (both shrinking and growing) happen correctly?

Parameter

Resize command with number of blocks to be evicted or given back

Metric

Number of read errors [a] we get in the userspace while reading the blocks sequentially on the vSSD device using direct I/O.

[a] The frontend driver does not allow reads on blocks evicted during a resize command.

Experiment: Correctness

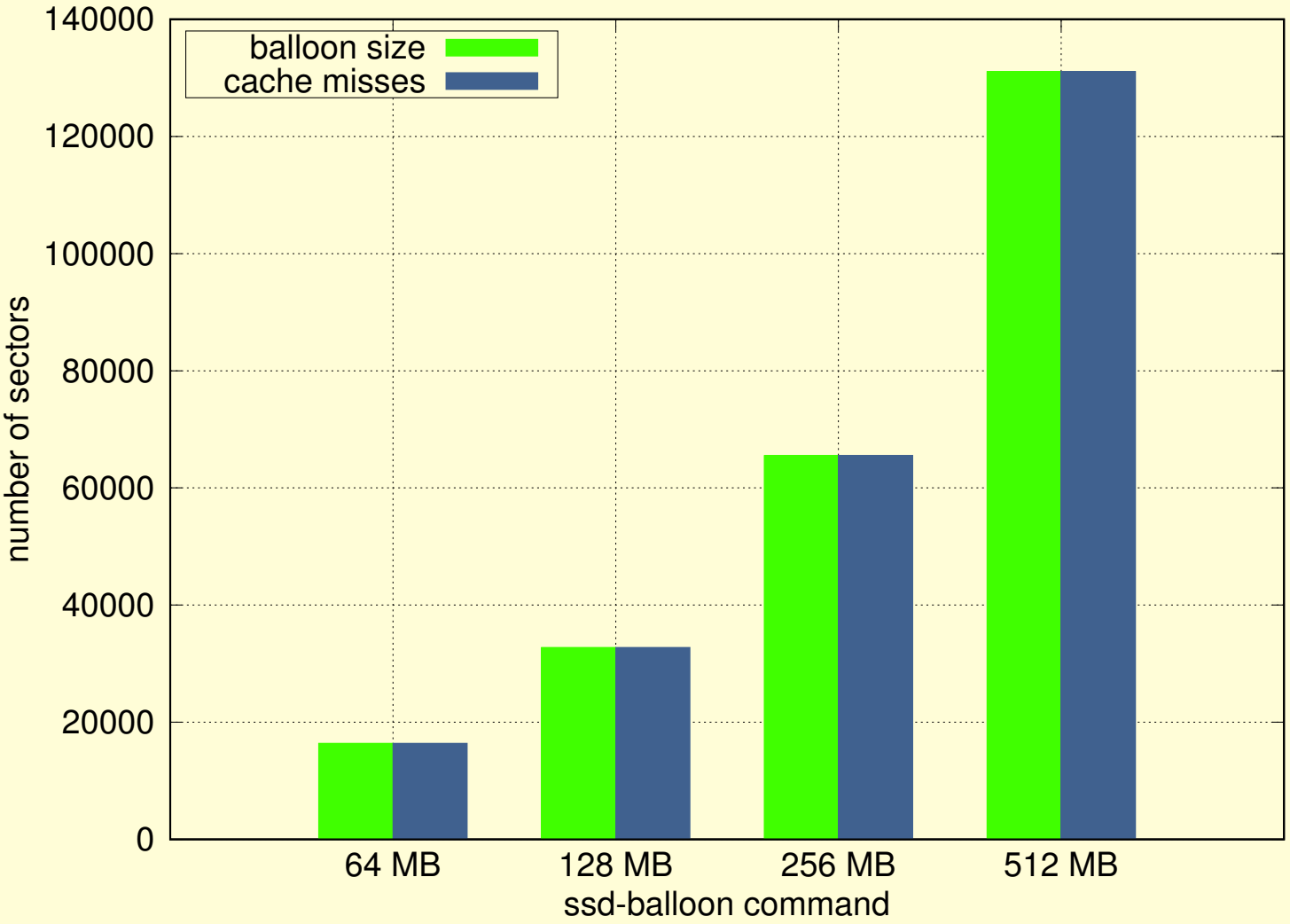

Figure: Balloon sizes for `ssd-balloon` command

Experiment: Case for Guest-managed Evictions

Hypothesis

Guest-managed evictions give better performance than hypervisor-managed evictions.

Experiment: Case for Guest-managed Evictions

Setup

- A userspace application reads blocks randomly generated according to Gaussian distribution (μ = 20000, σ = 8192)
- The disk is shrunk by 256 MB
- For guest-managed evictions, a differently seeded random number generator generates block numbers according to same Gaussian distribution; we evict the unaccessed blocks
- For hypervisor-managed evictions, blocks are generated according to uniform distribution

Experiment: Case for Guest-managed Evictions

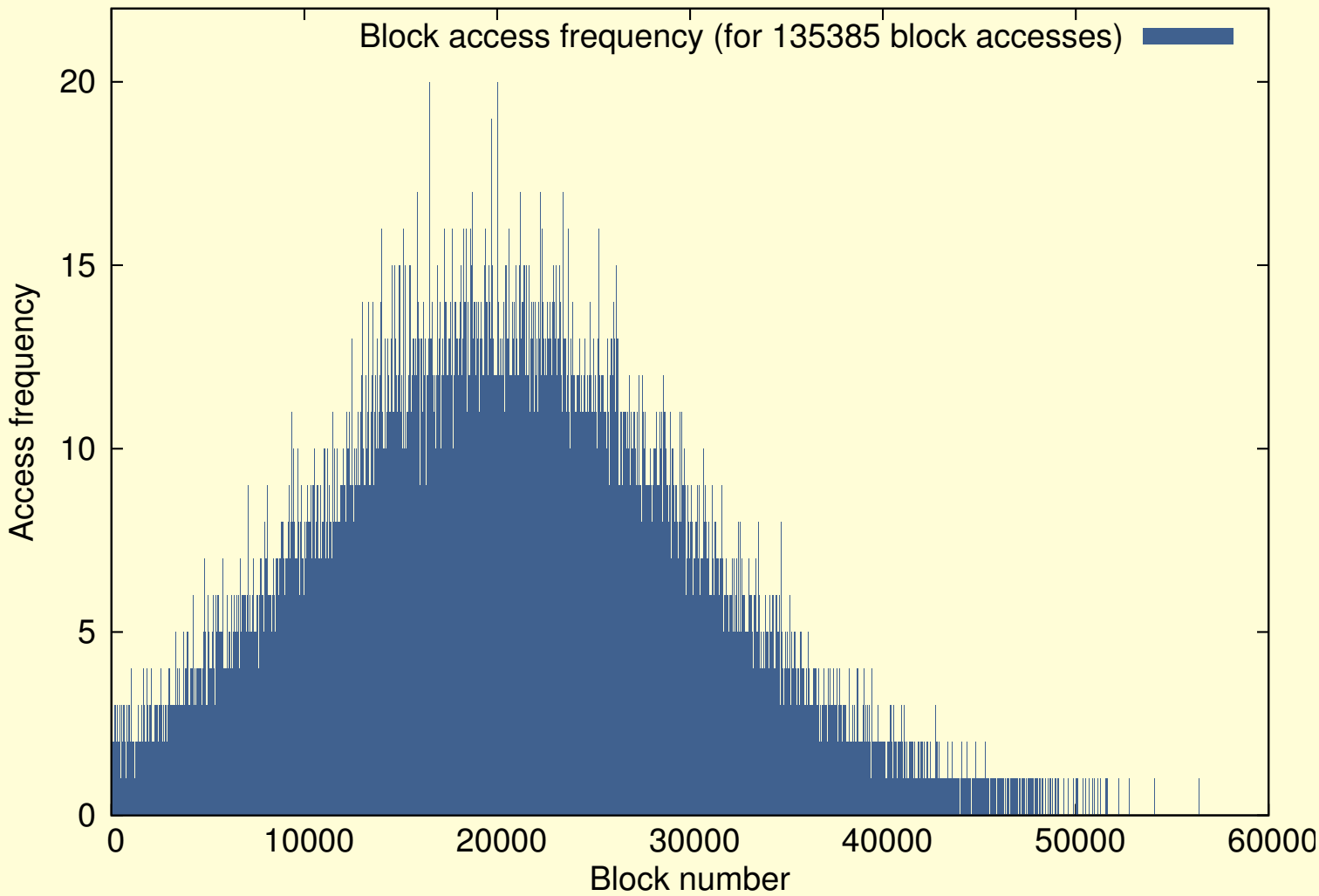

Figure: Block access pattern for the userspace application

Experiment: Case for Guest-managed Evictions

Parameter

The block numbers to be evicted, generated according to different distributions

Experiment: Case for Guest-managed Evictions

Parameter

The block numbers to be evicted, generated according to different distributions

Metric

The number of cache misses (failed block reads) on the vSSD device

Experiment: Case for Guest-managed Evictions

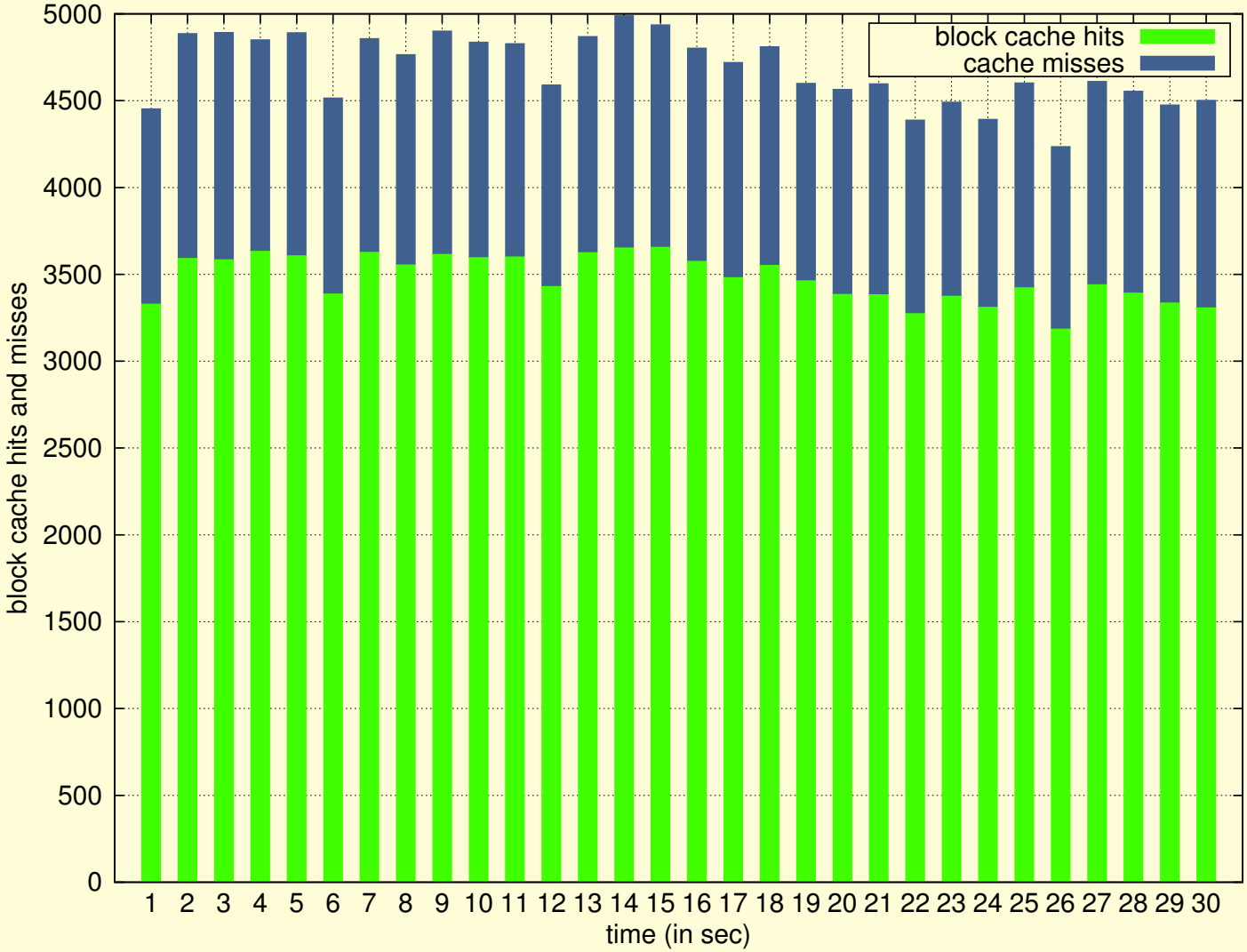

Figure: Block cache hits and misses with hypervisor-managed evictions

Experiment: Case for Guest-managed Evictions

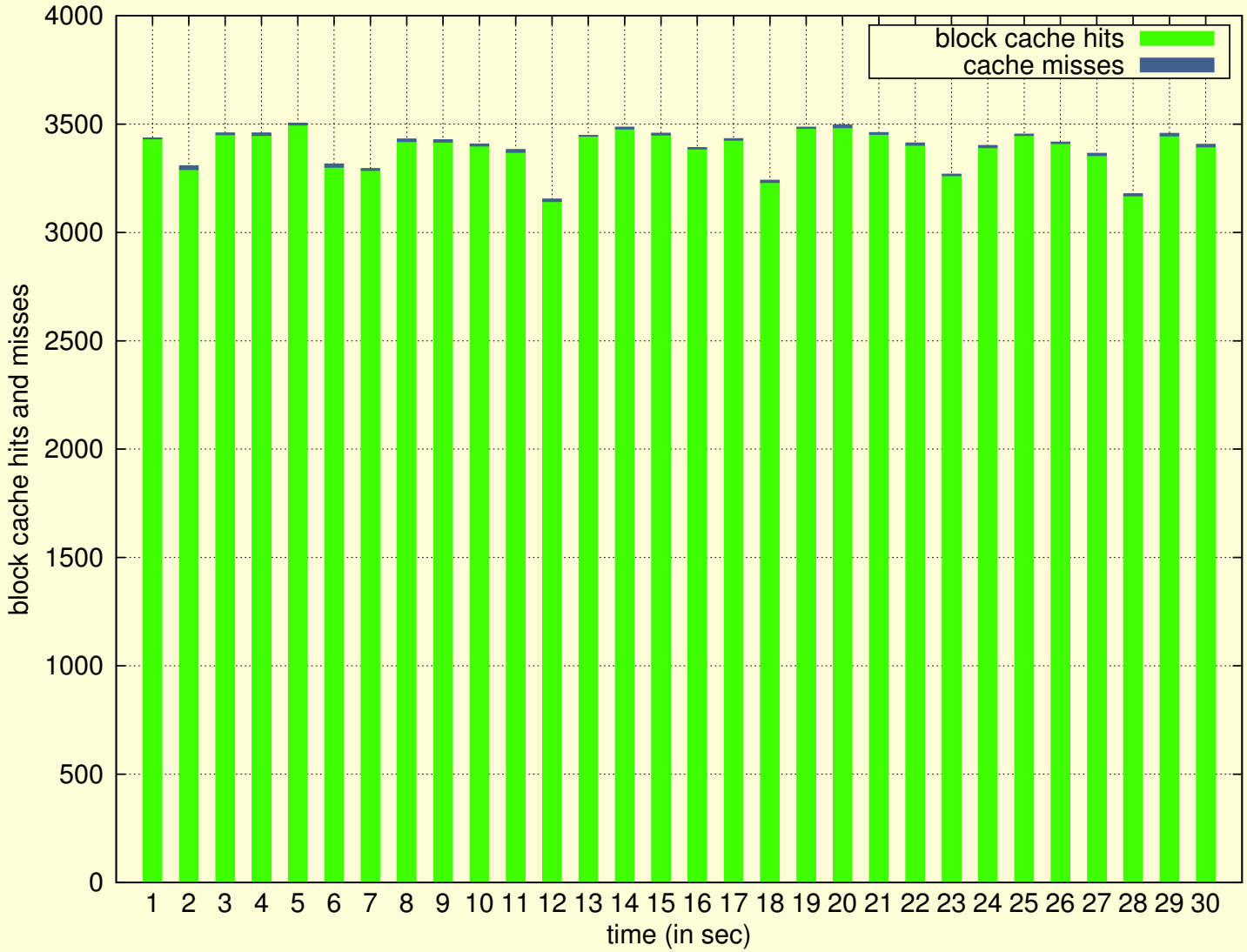

Figure: Block cache hits and misses with guest-managed evictions

Experiment: Case for Guest-managed Evictions

Observation

Guest-managed evictions give much better performance than hypervisor-managed evictions for this highly orchestrated experiment.

Experiment: Virtualization Overhead

Question

What is the time taken to resize (shrink and grow) the vSSD device?

Experiment: Virtualization Overhead

Question

What is the time taken to resize (shrink and grow) the vSSD device?

Parameter

Resize command with number of blocks to be evicted or given back

Experiment: Virtualization Overhead

Question

What is the time taken to resize (shrink and grow) the vSSD device?

Parameter

Resize command with number of blocks to be evicted or given back

Metric

The time taken to shrink to the given size and grow back to full size again

Experiment: Virtualization Overhead

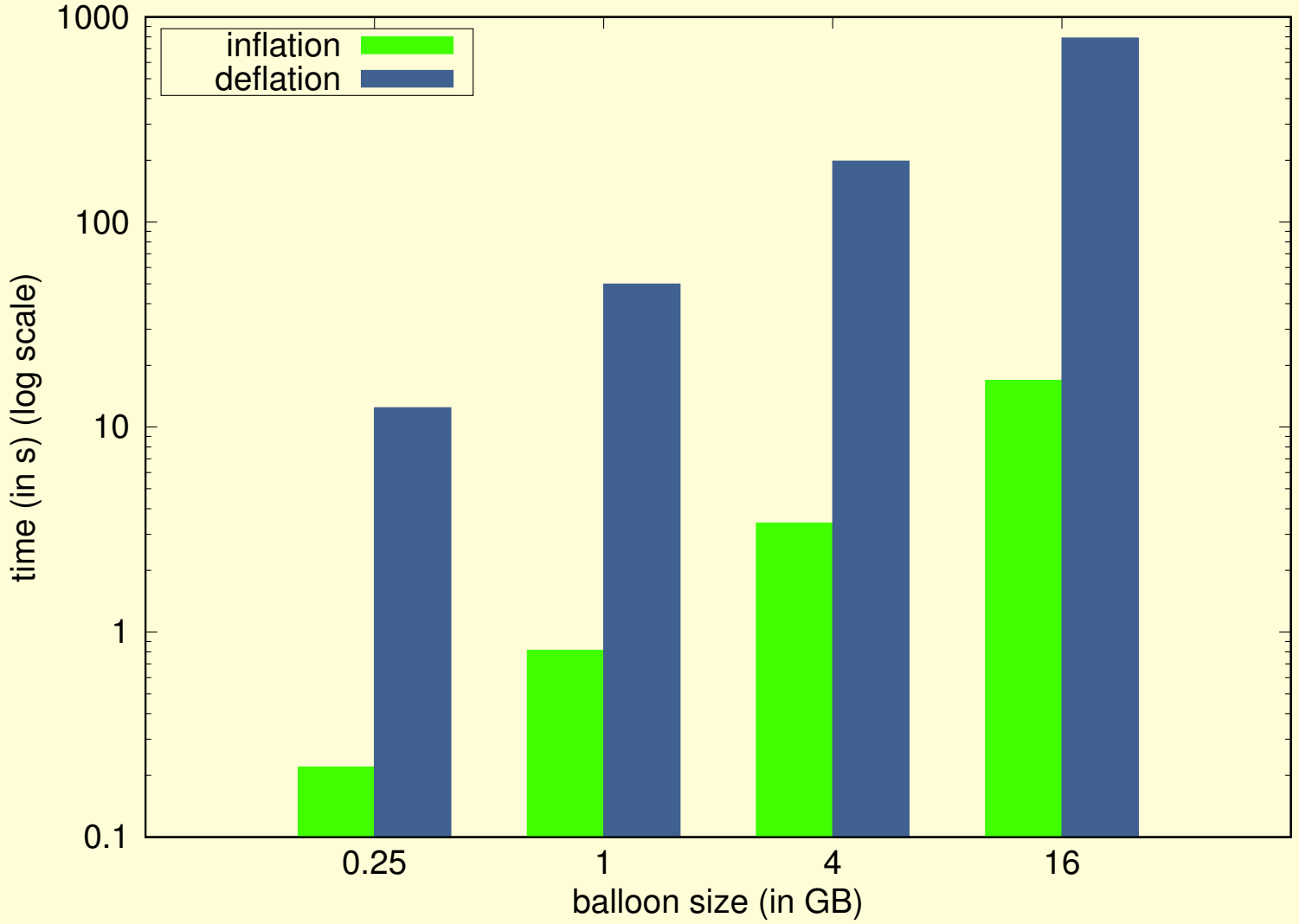

Figure: Time taken for dynamic resizing

Experiment: Virtualization Overhead

Observations

- The time taken to grow is much greater than the time to shrink the vSSD device.

Experiment: Virtualization Overhead

Observations

- The time taken to grow is much greater than the time to shrink the vSSD device.
- This is because of an extra step involved in the growing phase.

Experiment: Virtualization Overhead

Observations

- The time taken to grow is much greater than the time to shrink the vSSD device.
- This is because of an extra step involved in the growing phase.
- The current implementation exchanges only 4096 sectors at a time. Thus multiple iterations are involved to get the required size.

Outline

Conclusion and Next Steps

- We have created a basic setup and defined a new type of dynamically resizable device.

Conclusion and Next Steps

- We have created a basic setup and defined a new type of dynamically resizable device.
- This setup works only for one VM for now. Need to extend this to multiple VMs sharing the underlying SSD; thus share state across VMs.

Conclusion and Next Steps

- We have created a basic setup and defined a new type of dynamically resizable device.
- This setup works only for one VM for now. Need to extend this to multiple VMs sharing the underlying SSD; thus share state across VMs.
- Unresolved issue of growing phase taking unacceptably long time

Conclusion and Next Steps

- We have created a basic setup and defined a new type of dynamically resizable device.
- This setup works only for one VM for now. Need to extend this to multiple VMs sharing the underlying SSD; thus share state across VMs.
- Unresolved issue of growing phase taking unacceptably long time
- Unresolved issue of hypervisor caching the accesses to SSD despite doing direct I/O

Conclusion and Next Steps

- We have created a basic setup and defined a new type of dynamically resizable device.
- This setup works only for one VM for now. Need to extend this to multiple VMs sharing the underlying SSD; thus share state across VMs.
- Unresolved issue of growing phase taking unacceptably long time
- Unresolved issue of hypervisor caching the accesses to SSD despite doing direct I/O
- Integrate the solution with real-life applications like dm-cache, the block caching layer in Linux kernel

Thank you!

Thank you for listening patiently!

Feedback/Comments/Questions?

Thank you!

. . . and thank you Puru, for everything!

Thank you!

. . . and thank you Puru, for everything!

~~Sir Puru, Puru Sir, Sir, *Sir*, . . .~~

References I

Circular buffer image.

https://gnuradio.org/wp-content/uploads/2017/01/ring-buffers_web.svg.

[Online; Last accessed: 22-June-2017].

Disk image.

http://cdn.onlinewebfonts.com/svg/img_504033.svg.

[Online; Last accessed: 22-June-2017].

AMAZON.

Amazon ec2 instance store.

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html.

[Online; Last accessed: 22-June-2017].

References II

Byan, S., Lentini, J., Madan, A., and Pabón, L.

Mercury: Host-side flash caching for the data center.

In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), 2012* (apr 2012), pp. 1–12.

Dulloor, S. R., Roy, A., Zhao, Z., Sundaram, N., Satish, N., Sankaran, R., Jackson, J., and Schwan, K.

Data Tiering in Heterogeneous Memory Systems.

In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, ACM, pp. 15:1—-15:16.

Koller, R., Marmol, L., Rangaswami, R., Sundararaman, S., Talagala, N., and Zhao, M.

Write Policies for Host-side Flash Caches.

In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 13)* (San Jose, CA, 2013), USENIX, pp. 45–58.

References III

Lee, C., Sim, D., Hwang, J., and Cho, S.

F2FS: A New File System for Flash Storage.

In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 273–286.

Liang, L., Chen, R., Chen, H., Xia, Y., Park, K., Zang, B., and Guan, H.

A case for virtualizing persistent memory.

In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (New York, NY, USA, 2016), SoCC '16, ACM, pp. 126–140.

Mittal, S., and Vetter, J. S.

A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems.

IEEE Transactions on Parallel and Distributed Systems 27, 5 (may 2016), 1537–1550.