Interim

Enhancing the Performance of Live Migration by Mitigating Redundant Memory Page Transfers in Pre-Copy Migration

Samindu R. Cooray | CloudNet Research Group

Supervisor: Dr. Dinuni K. Fernando

Table of Contents

01

Significance of Migration

Significance of Migration

- Cloud Computing (CC) offers efficient and reliable services to millions globally.
    - Eg:CC service providers include Microsoft Azure, Amazon Web Services (AWS), Alibaba Cloud, and Google Cloud Platform (GCP).
    - Ensure **uninterrupted services**
- Data centers use **virtualization** to offer services by hosting multiple virtual machines (VMs) on a single server.
- **Hardware** or **software failures** in servers, or essential **updates** in the servers can disrupt VM services.
- Live VM Migration techniques are employed moves VMs to another physical server. This approach supports **fault tolerance**, **load balancing**, **host maintenance**, and **server consolidation**.

Significance of Migration

1,000,000 Migrations Per Month

(Ruprecht, A., Jones, D., Shiraev, D., Harmon, G., Spivak, M., Krebs, M., Baker-Harvey, M. & Sanderson, T. (2018), 'Vm live migration at scale', ACM SIG-PLAN Notices 53(3), 45–56.)

02

Background

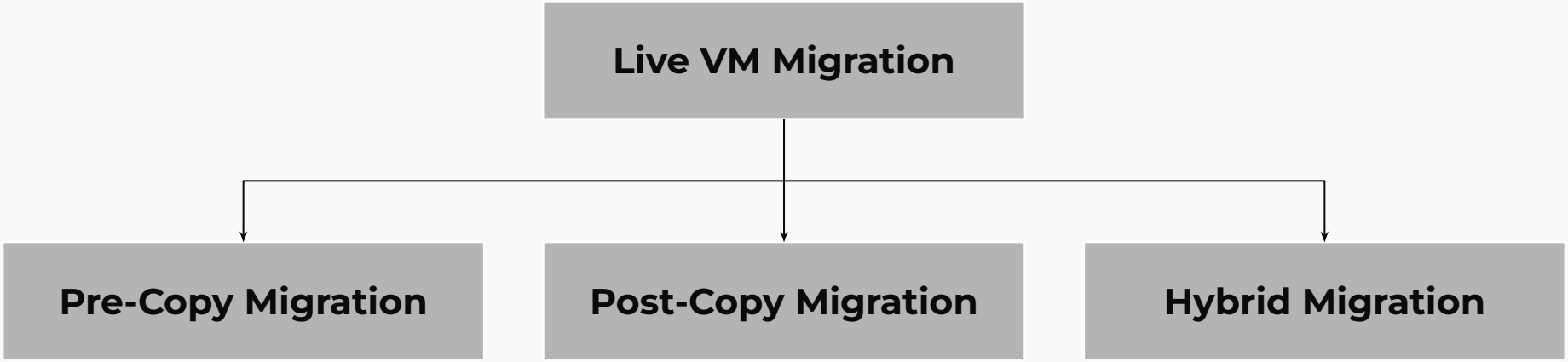
Background

Live VM Migration : Migrating a VM from a **Source Host Machine** to a **Destination Host Machine** while the **VM is in operating state**.

Goal : *Provide uninterrupted services to end-users.*

Background

Migration Techniques



Background

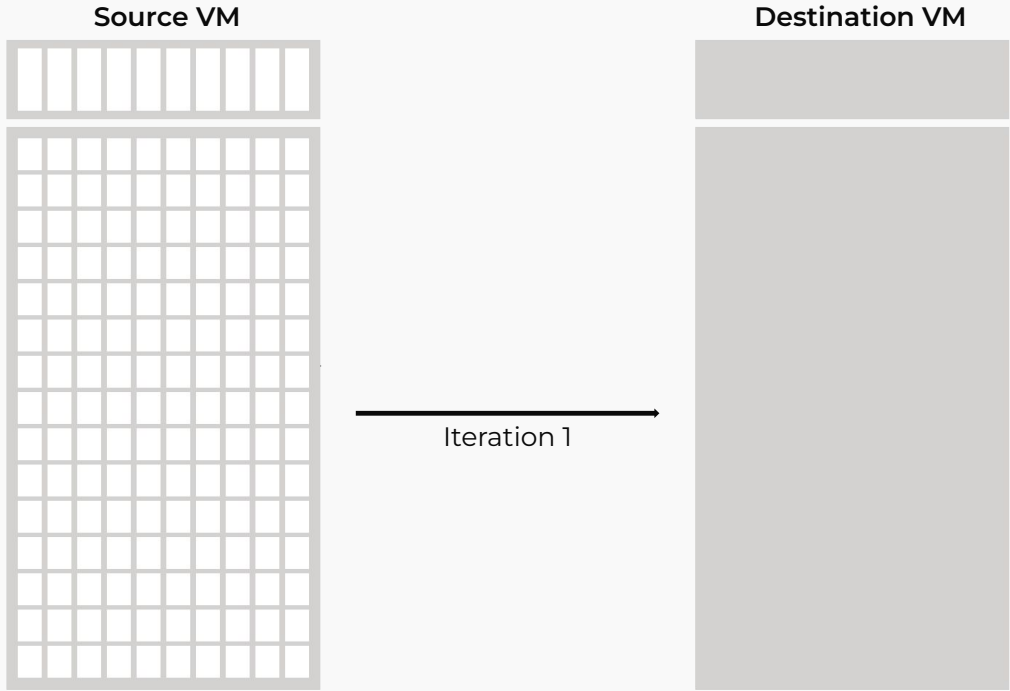
Pre-Copy Migration

Push Phase

1. Transfer all memory pages to the destination.
2. Iteratively transfer dirty pages to the destination until convergence point.

Stop-and-Copy Phase

3. At Convergence point, suspend the VM in the source and transfer the CPU and I/O state and the remaining memory pages to the destination.
4. Start VM in destination.



Background

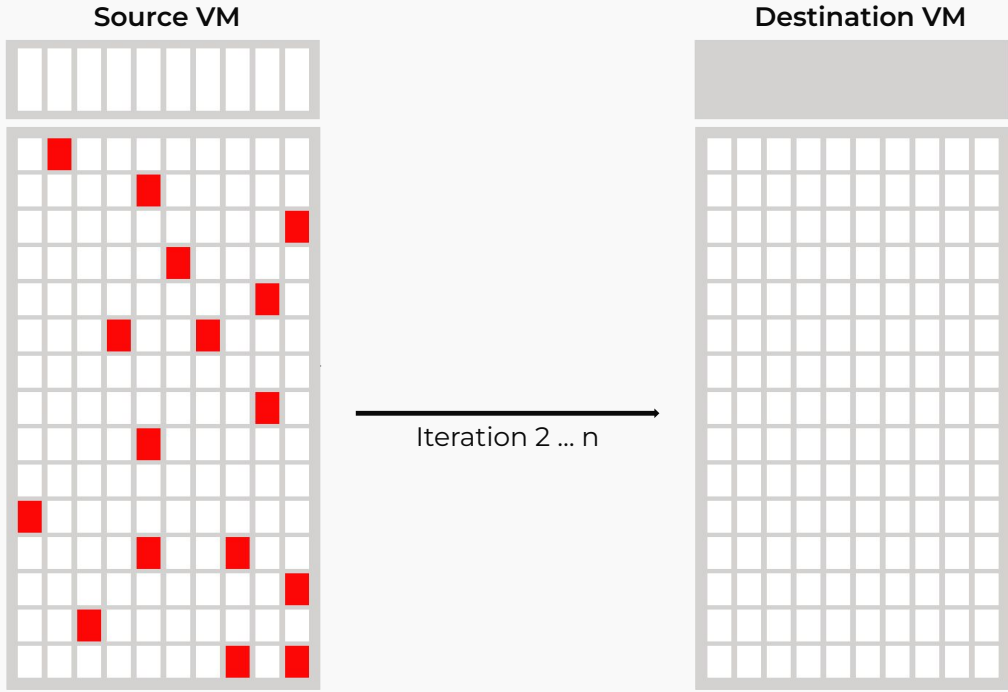
Pre-Copy Migration

Push Phase

1. Transfer all memory pages to the destination.
2. Iteratively transfer dirty pages to the destination until convergence point.

Stop-and-Copy Phase

3. At Convergence point, suspend the VM in the source and transfer the CPU and I/O state and the remaining memory pages to the destination.
4. Start VM in destination.



Background

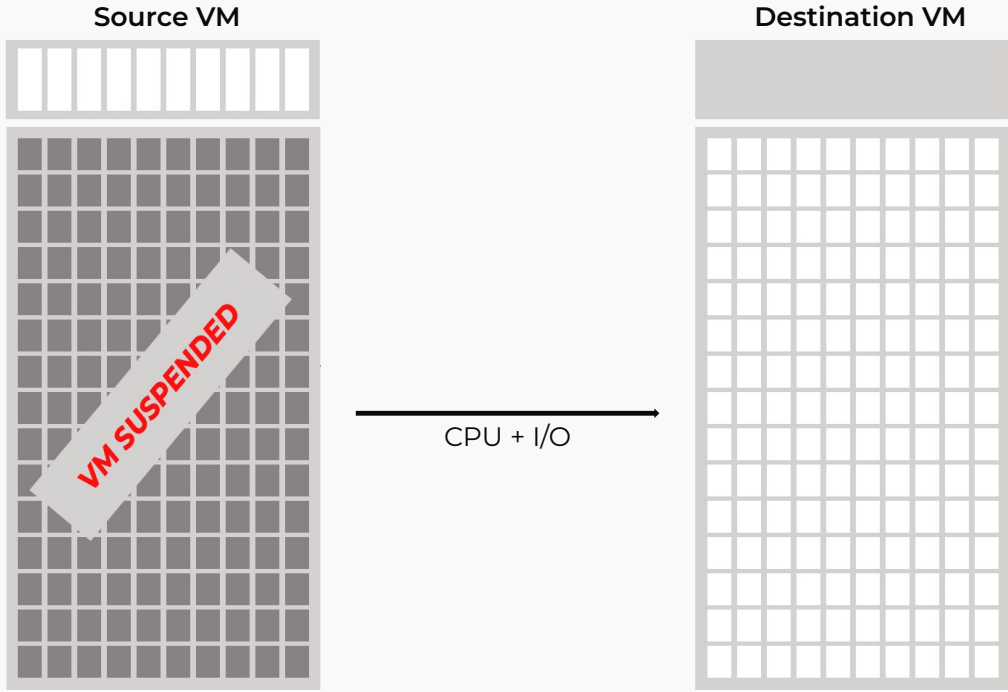
Pre-Copy Migration

Push Phase

1. Transfer all memory pages to the destination.
2. Iteratively transfer dirty pages to the destination until convergence point.

Stop-and-Copy Phase

3. At Convergence point, suspend the VM in the source and transfer the CPU and I/O state and the remaining memory pages to the destination.
4. Start VM in destination.



Background

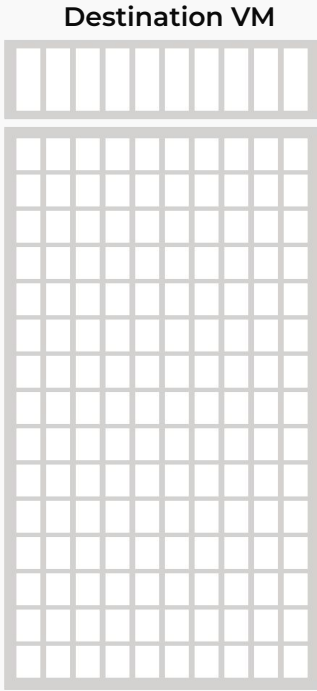
Pre-Copy Migration

Push Phase

1. Transfer all memory pages to the destination.
2. Iteratively transfer dirty pages to the destination until convergence point.

Stop-and-Copy Phase

3. At Convergence point, suspend the VM in the source and transfer the CPU and I/O state and the remaining memory pages to the destination.
4. Start VM in destination.



Background

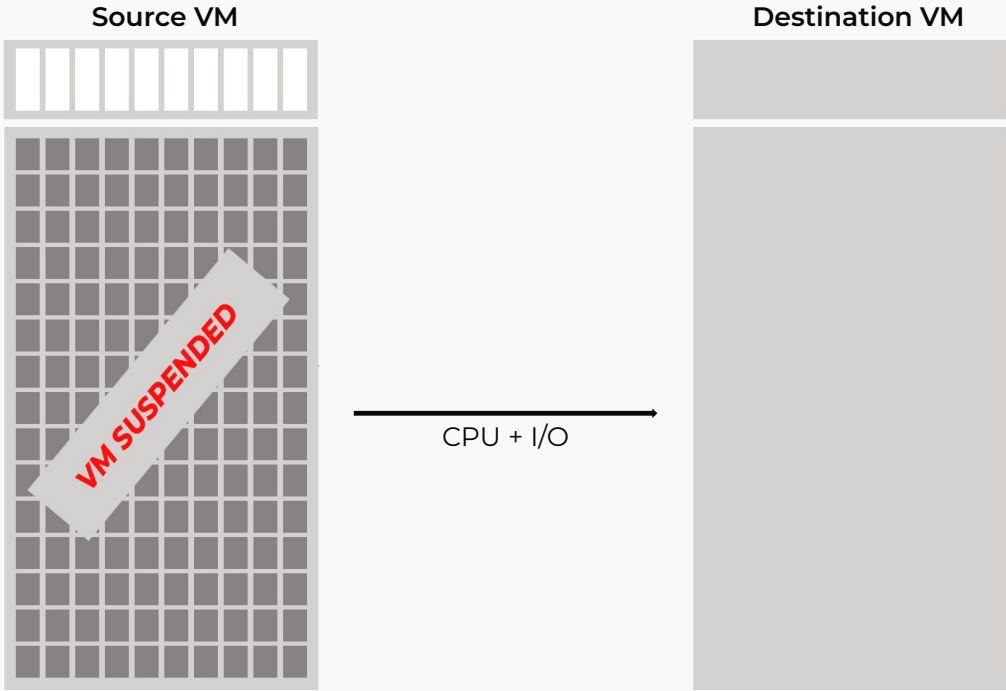
Post-Copy Migration

Stop-and-Copy Phase

1. Suspend the VM in the Source and transfer the CPU and I/O state to the destination.
2. Resume VM in destination.

Pull Phase

3. The source actively pushes pages to destination.
4. The destination fetches page faulted pages from source.



Background

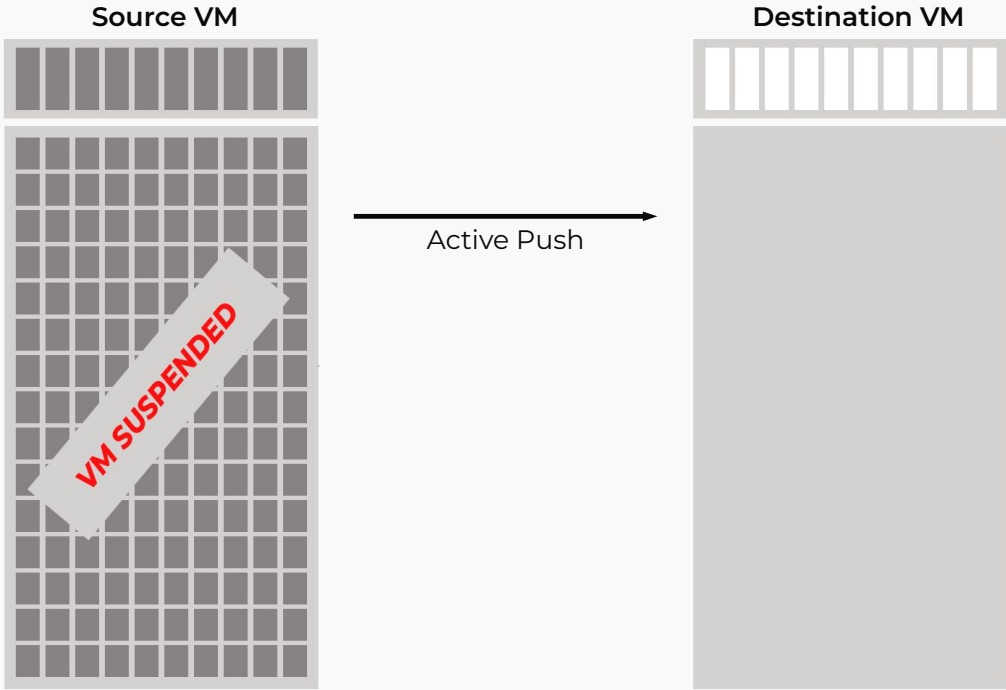
Post-Copy Migration

Stop-and-Copy Phase

1. Suspend the VM in the Source and transfer the CPU and I/O state to the destination.
2. Resume VM in destination.

Pull Phase

3. The source actively pushes pages to destination.
4. The destination fetches page faulted pages from source.



Background

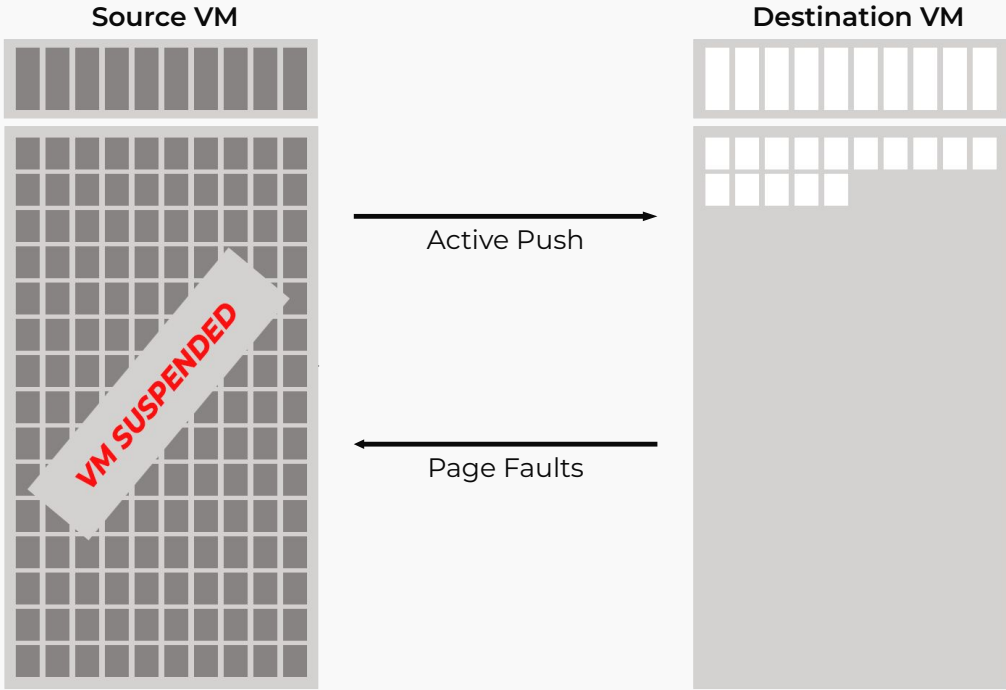
Post-Copy Migration

Stop-and-Copy Phase

1. Suspend the VM in the Source and transfer the CPU and I/O state to the destination.
2. Resume VM in destination.

Pull Phase

3. The source actively pushes pages to destination.
4. The destination fetches page faulted pages from source.



Background

Hybrid Migration

Push Phase

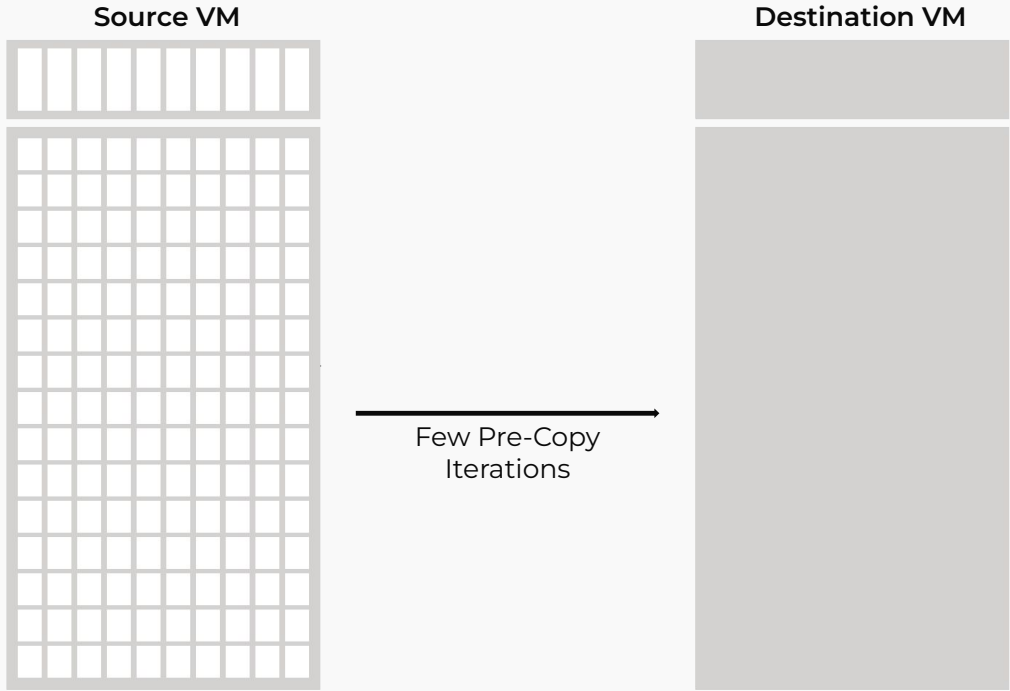
1. Transfer memory pages in few iterations

Stop-and-Copy Phase

2. Suspend the VM in the Source and transfer the CPU and I/O state to the destination.

Pull Phase

3. Resume VM in destination.
4. The destination fetches paging from the source.



Background

Hybrid Migration

Push Phase

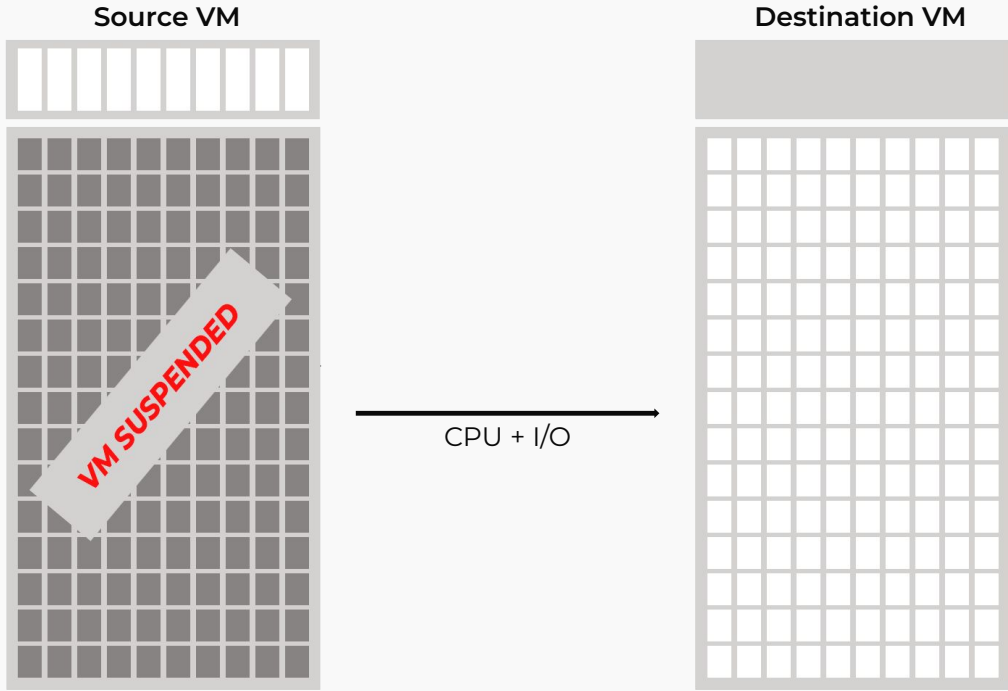
1. Transfer memory pages in few iterations

Stop-and-Copy Phase

2. Suspend the VM in the Source and transfer the CPU and I/O state to the destination.

Pull Phase

3. Resume VM in destination.
4. The destination fetches paging from the source.



Background

Hybrid Migration

Push Phase

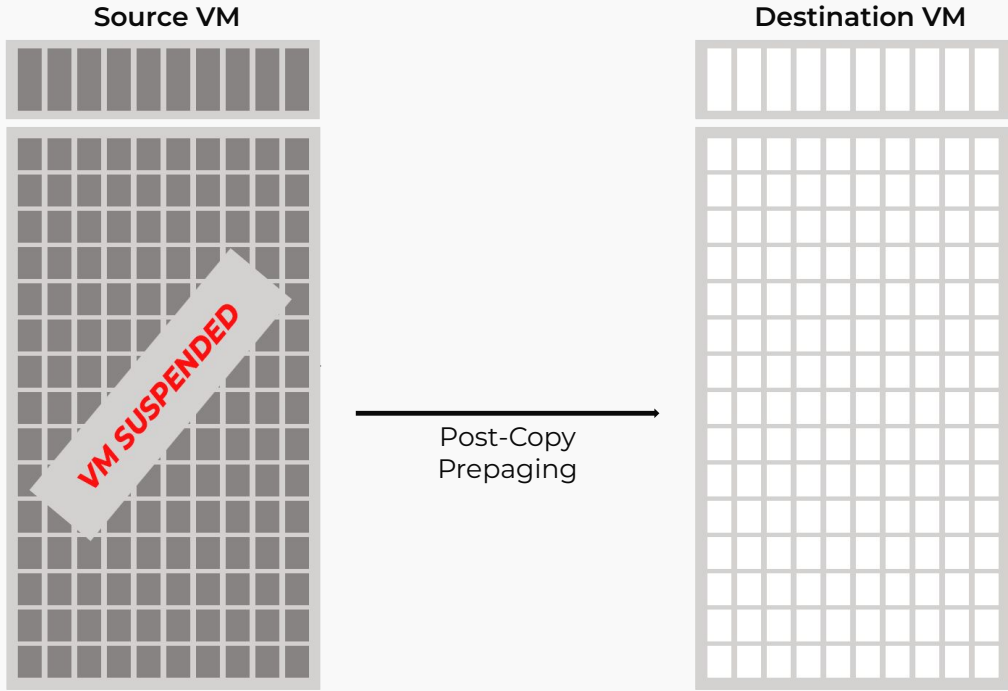
1. Transfer memory pages in few iterations

Stop-and-Copy Phase

2. Suspend the VM in the Source and transfer the CPU and I/O state to the destination.

Pull Phase

3. Resume VM in destination.
4. The destination fetches paging from the source.



03

Problem Statement

Jul et al., Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, 2005
Li et al., Future Generation Computer Systems, 2019
Elsaid et al., Distributed and Parallel Databases, 2022

Problem Statement

Challenges in Pre-Copy Migration

Primary Bottleneck in Pre-Copy : Total Migration Time.

Factors Affecting :

- Varying dirty page rates.
- Network transmission speed.

In extreme cases, rapid dirty page generation and low network bandwidth can lead to **prolonged total migration time** or even **migration failure**.

Jul et al., Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, 2005
Li et al., Future Generation Computer Systems, 2019
Elsaid et al., Distributed and Parallel Databases, 2022

Problem Statement

Challenges in Pre-Copy Migration

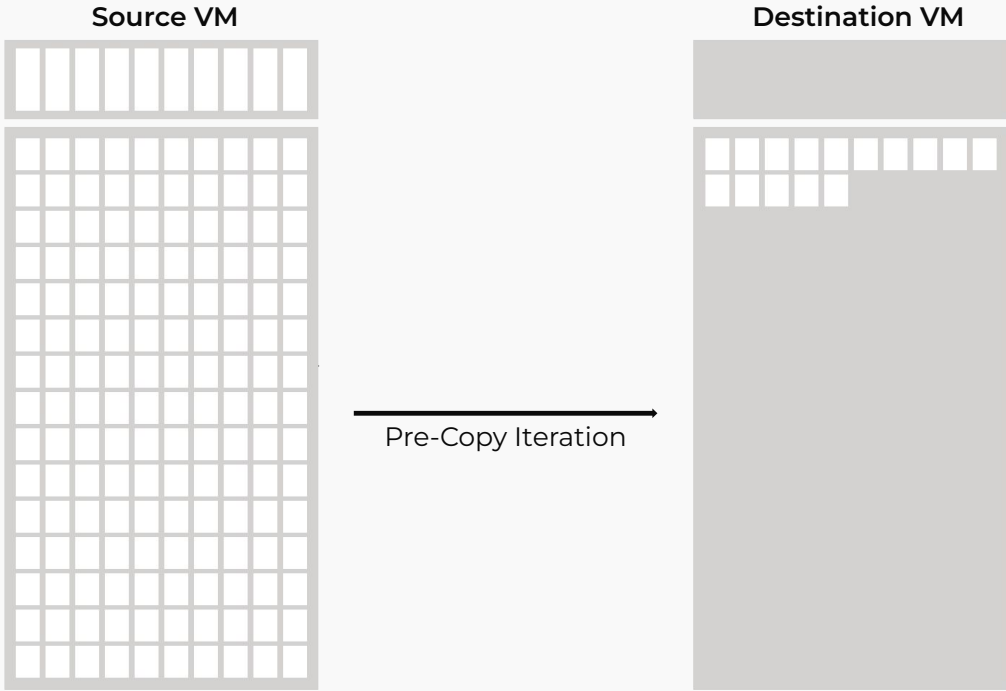
Interestingly we found that most of these **rapidly generated pages are Fake !** although this was initially found by Li et. al. in 2019 but not yet in plugged to streamline QEMU codebase!

What is a Fake Dirty Page ?

Problem Statement

Fake Dirty Page

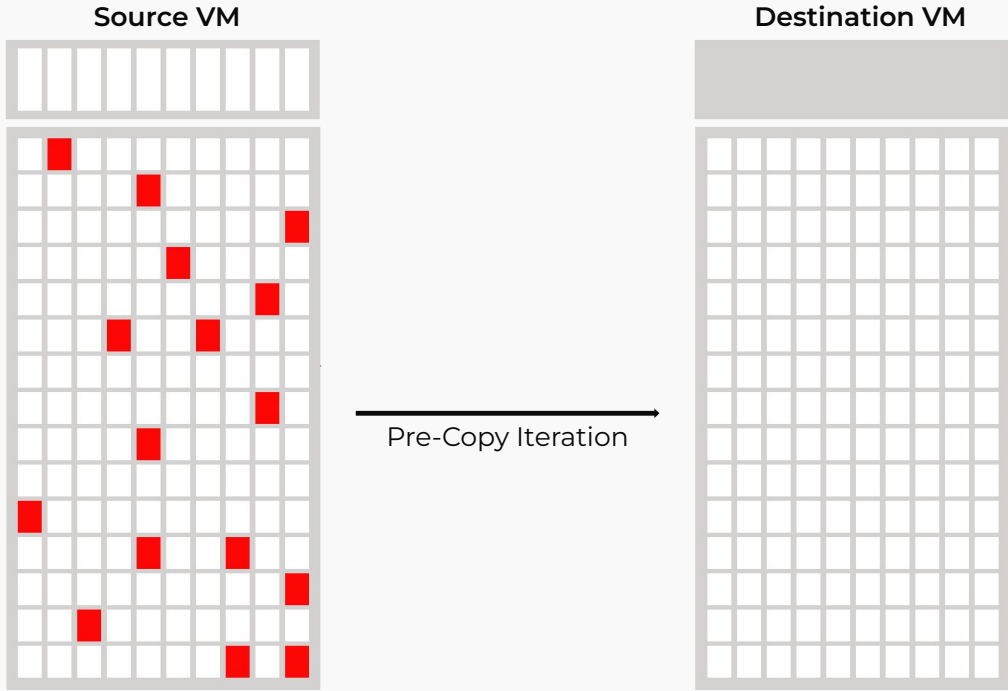
- During migration thread performs the migration the hypervisor keeps track of the pages that modified and mark that page as dirty in a bitmap called dirty bitmap.
- Then with the use of the dirty bitmap, only the pages that are dirtied in the previous iteration are considered to migrate in the current iteration .



Problem Statement

Fake Dirty Page

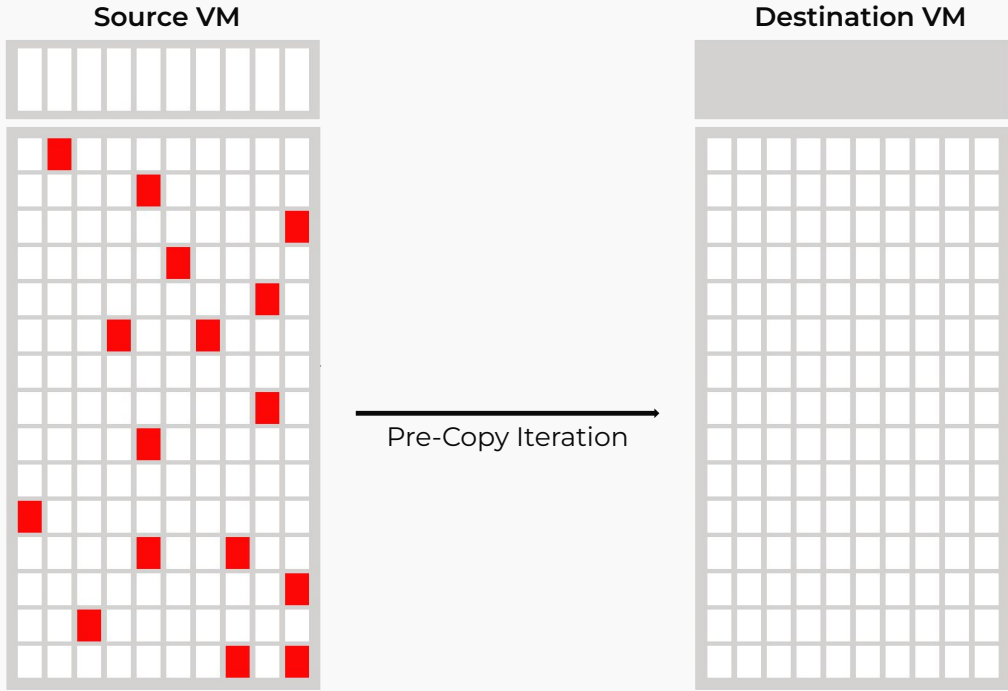
- During migration thread performs the migration the hypervisor keeps track of the pages that modified and mark that page as dirty in a bitmap called dirty bitmap.
- Then with the use of the dirty bitmap, only the pages that are dirtied in the previous iteration are considered to migrate in the current iteration .



Problem Statement

Fake Dirty Page

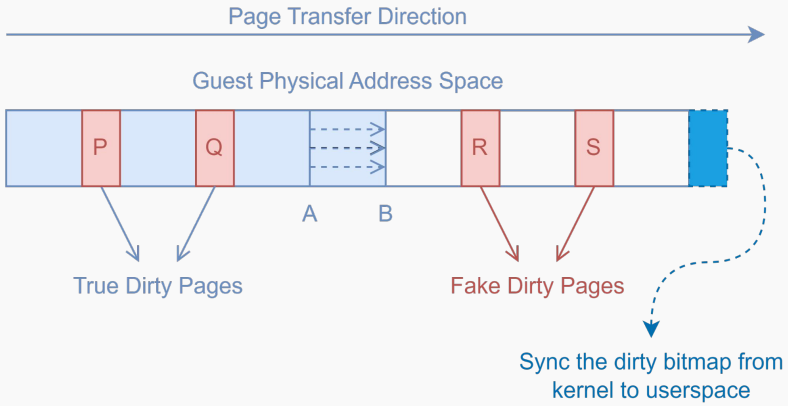
- In Reality, out of the pages marked as dirty in the previous iteration, there are some pages that **do not** have any actual **content change**. These pages are known as **Fake Dirty Pages**.
- This unnecessary duplication wastes **network bandwidth** and prolong **migration time**.



Problem Statement

Reasons for Fake Dirty Page Generation

- ***“ Write-not-dirty ”*** requests issued by *Silent Store Instructions*, *where existing value is written again to the memory address resulting no state change.*
- Defects in the ***Dirty Page Tracking Mechanism***.



How to mitigate redundant data transfers by identifying fake dirty pages?

04

Research Gap

Research Gap

- A primary concern in VM migration is minimizing the **total migration time** to mitigate service interruptions.
  - In Pre-Copy, the total migration time depends on the number of **Pre-Copy rounds** (memory transfer iterations).
  - If these Pre-Copy rounds increase without converging, the tendency of migration failure increases.
- During Pre-Copy rounds, data exceeding the **VM's RAM size** is transmitted.
  - *For example, when migrating an 8GB VM, Pre-Copy rounds may transfer memory pages totaling more than 8GB*.
  - Reducing the number of pages transferred per Pre-Copy round can decrease the total migration time.
- Fake Dirty Pages, first identified in 2016 by Nathan et al. In 2019, Li et al. further investigated and proposed a solution based on **QEMU 2.5.1** in 2019.
  - The current version of **QEMU 8.1.2** reveal that there is no default handling of Fake Dirty Pages.

Research Gap

- According to Li et al., secure hashes (SHA1) were used to generate the hash of the page required in the algorithm.
    - Migration thread is paused until the hash of a page is generated and fake dirty pages are identified.
    - This study did not investigate the applicational overhead of hash computation and fake dirty page detection.
    - Overlooked the potential benefits of using **different hashing techniques over SHA1** to reduce the application overhead.
- Existing study has not considered applying different **optimization techniques** along with the hash-based fake dirty prevention algorithm for reducing total migration time.
    - Transferring a compressed page is more time effective than of a 4096 byte page over the network.
    - Application can enhance the efficiency of VM migration by minimizing total migration time and downtime.

05

Objectives

Objectives

01 To evaluate the performance improvement of using **different hashing mechanisms** to reduce redundant memory page transfers by considering Fake Dirty Pages.

02 To evaluate the performance improvement of **combining different optimization techniques** to reduce the data transferring load in Pre-Copy migration to improve the performance for memory-intensive workloads?

06

Methodology

Methodology

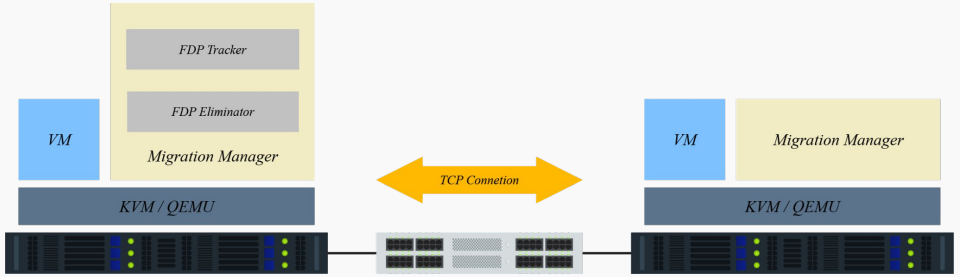
Fake Dirty Detection Mechanism

1. Initialize an array to store hash values for each RAM block.
2. Compute the hash of each page before transfer and store it in the corresponding array .
3. In subsequent iterations, compute the hash of each selected dirty page and compare the computed hash with the stored previous hash.
    a. If the hashes are similar, increment the fake dirty count variable by one.
    b. Otherwise, replace the previous hash with the current hash and transfer the page to the destination.

Methodology

Testbed Setup

- Setuped two physical servers and a NFS server interconnected with Gigabit ethernet connection.
- Two servers are setted with QEMU/KVM version 8.1.2
- A third node was selected to run network intensive / memory intensive workloads.



Prototype

- Implemented on KVM/QEMU 8.1.2
- Modifies QEMU default migration implementation
- Scripts for automated experiments and data gathering
- VMs are configured with 1 vCPU
- VM disks are accessed over the network via NFS Server

Methodology

Selecting Suitable Workloads

Workloads	Intensive Type	Description
Workingset	Memory	A benchmark that dirty pages to vary writable working set
Quicksort	CPU	A benchmark that repeatedly allocates random integers to an integer array of 1024 bytes and performs quicksort on the array.
Sysbench	CPU	A benchmark to assess the system performance of a machine planning on running a database under intensive load
Memcached	Multiple Resource	Memcached is an in-memory key-value store storing arbitrary data returned by a benchmark called Memaslap
YCSB	Multiple Resource	A Suite used to evaluate computer programs' retrieval and maintenance capabilities.

07

Preliminary Experiments

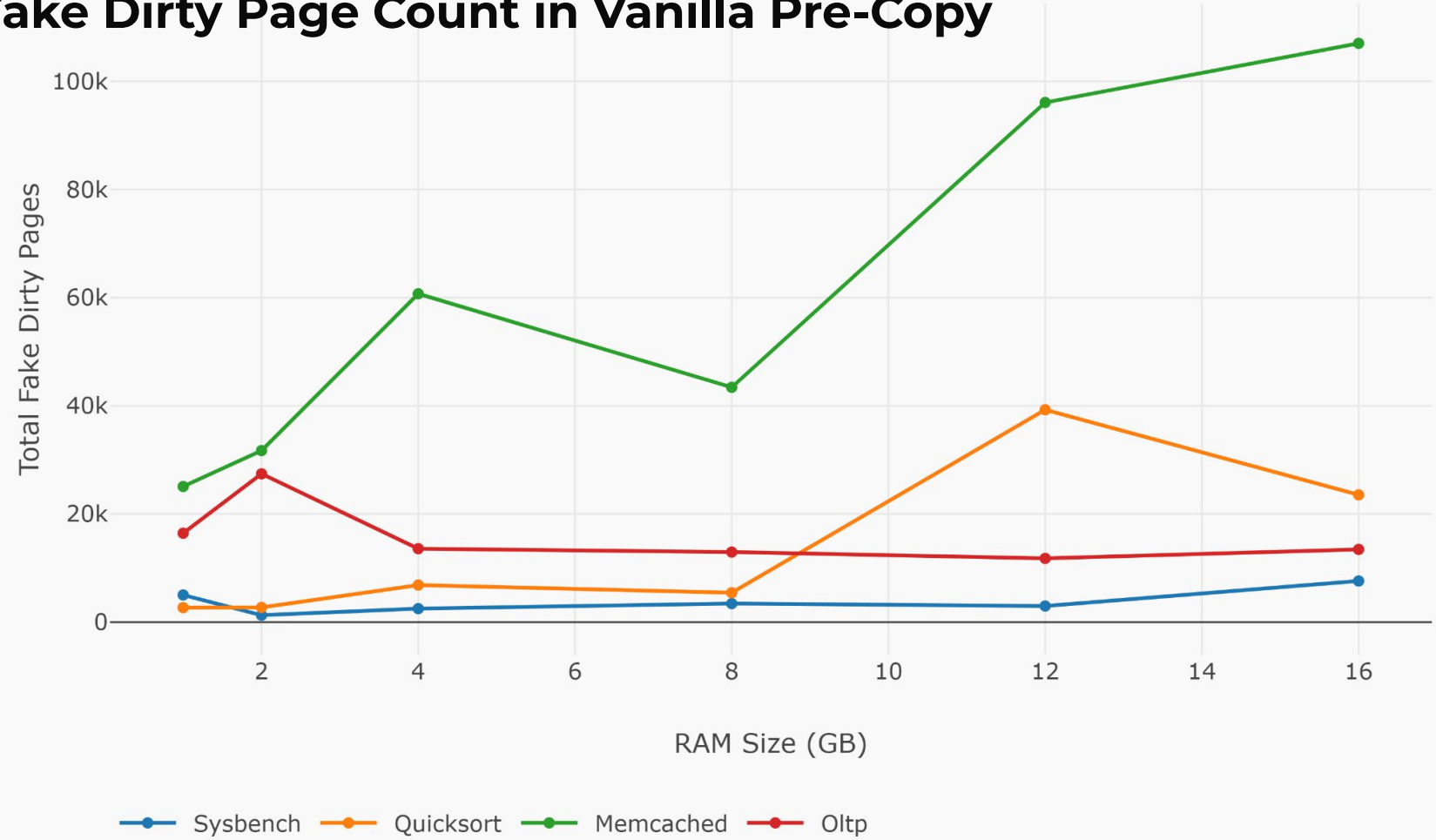
Preliminary Experiments

Fake Dirty Observation

- The initial experiment was to observe the presence of **Fake Dirty Pages** in Vanilla Pre-Copy Migration. The experiment was conducted for **6** VM memory sizes *(1GB, 2GB, 4GB, 8GB, 12GB, 16GB)*.
- Each data point is an average of 3 rounds of experiment.

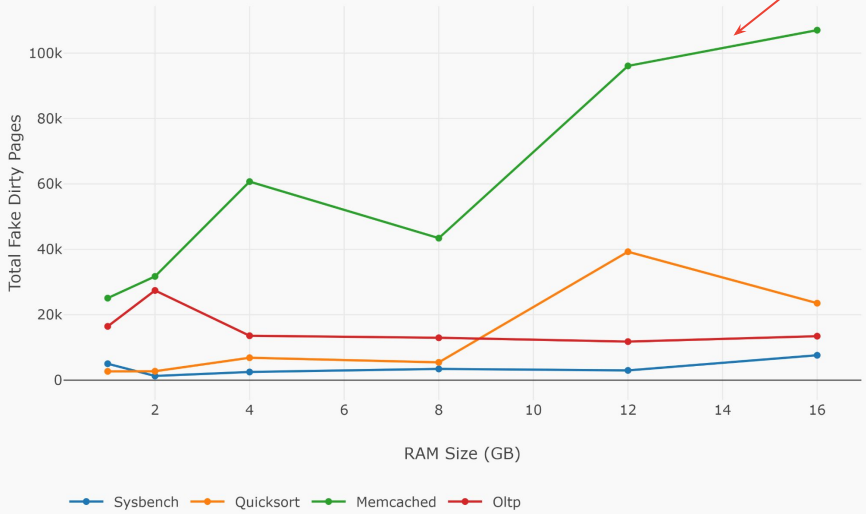
Analysis in the Next Slide

Fake Dirty Page Count in Vanilla Pre-Copy



Fake Dirty Page Count in Vanilla Pre-Copy

Fake Dirty Page Percentage Vary from 3% to 19% considering 1GB to 16GB RAM



Fake Dirty Percentage in memory-intensive workload Memcached

RAM (GB)	Fake Dirty Count	Transferred Page Count
1	33603 (9%)	380511
2	46093 (8%)	607484
4	79643 (7%)	1211109
8	58229 (3%)	2248032
12	113848 (3%)	3458888
16	138258 (3%)	4552333

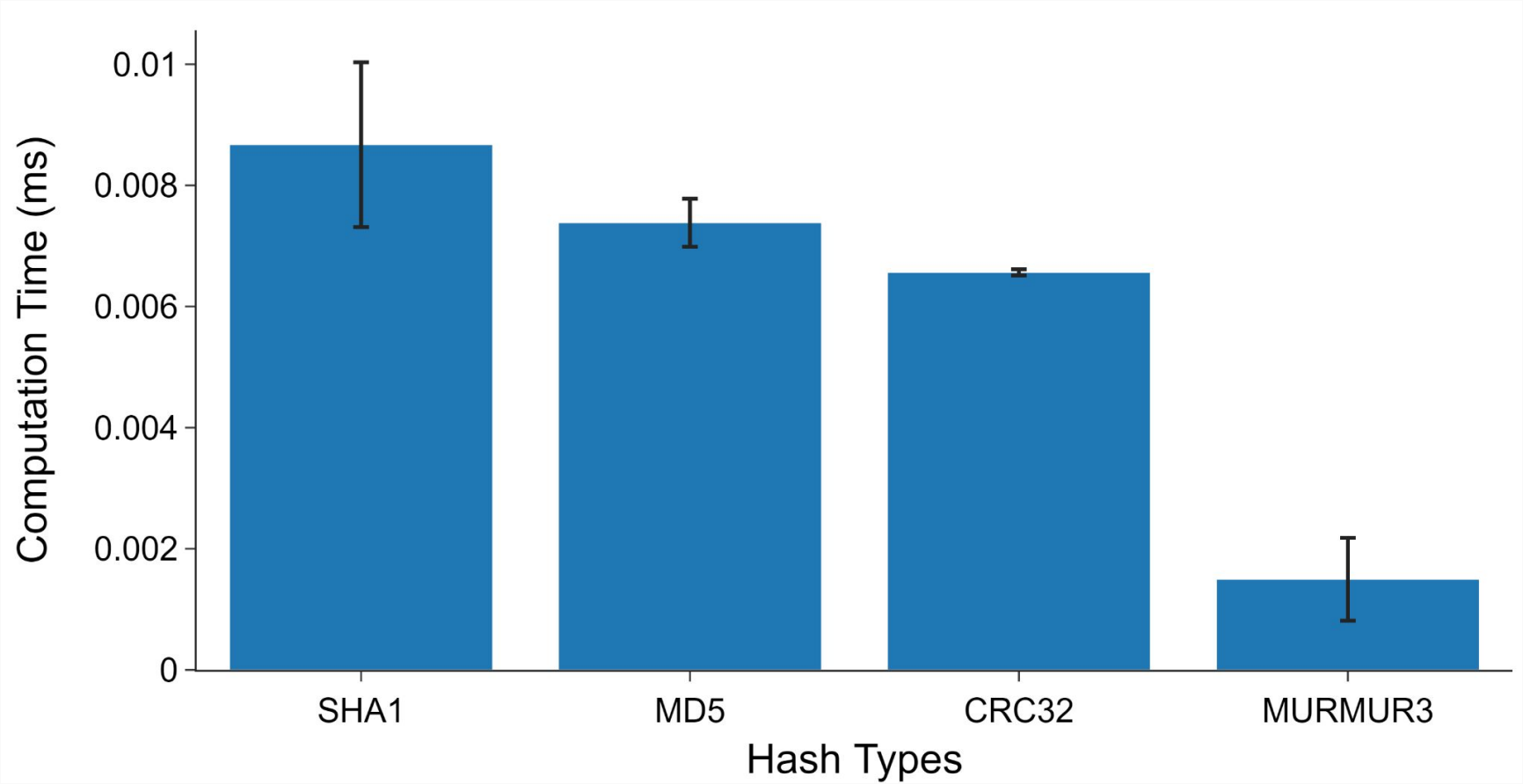
Preliminary Experiments

Hash Computation Time

- The next experiment was to observe the computation time for different hashing techniques.
- SHA1, MD5, CRC32, and Murmur3 hashing methods were used to conduct this experiment.
- In the experimental setup, a 4KB memory was allocated with random numbers and hashed using each hashing method, and the computation time was recorded.

Analysis in the Next Slide

Hash Computation Time



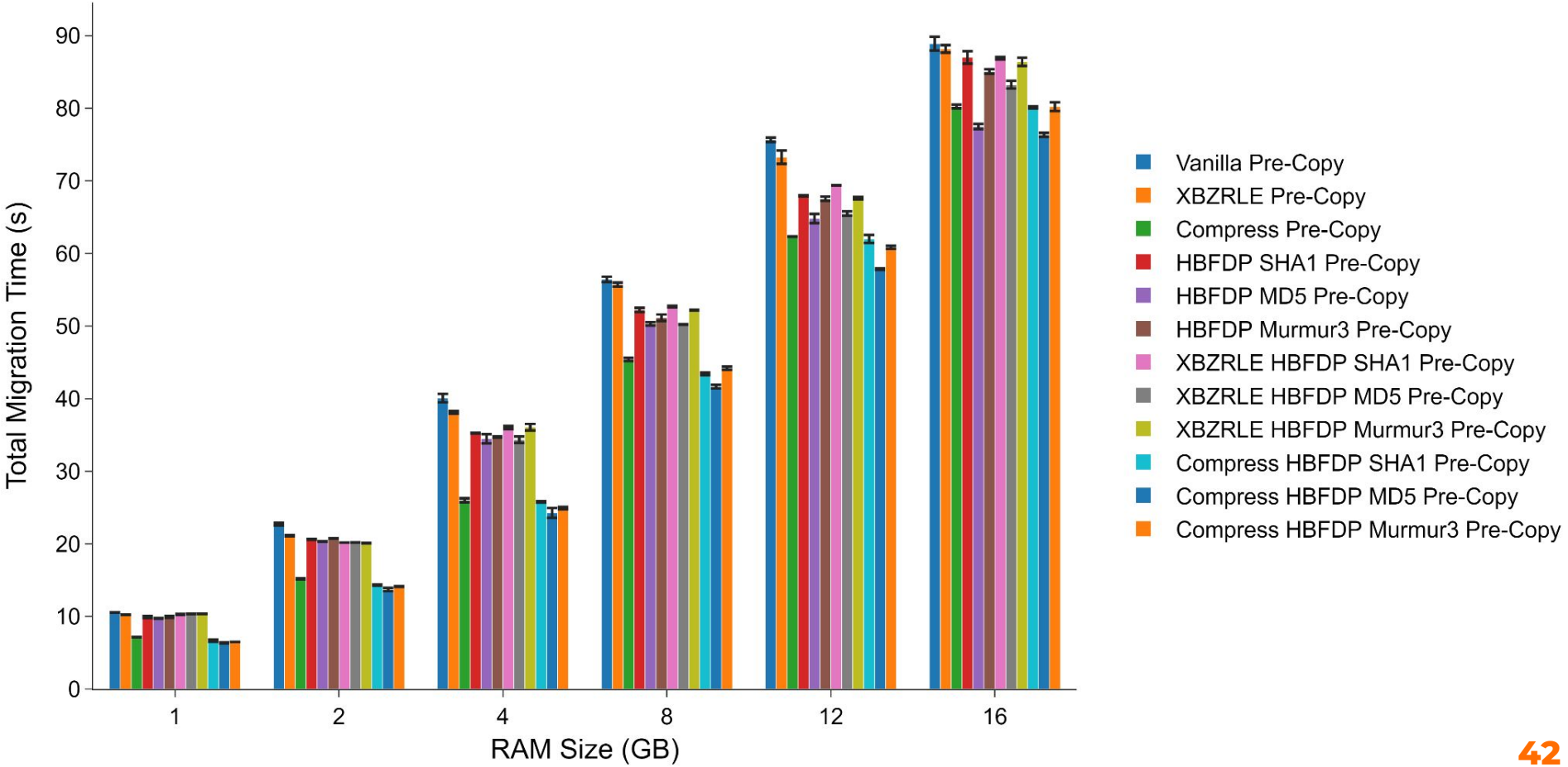
08

Evaluation

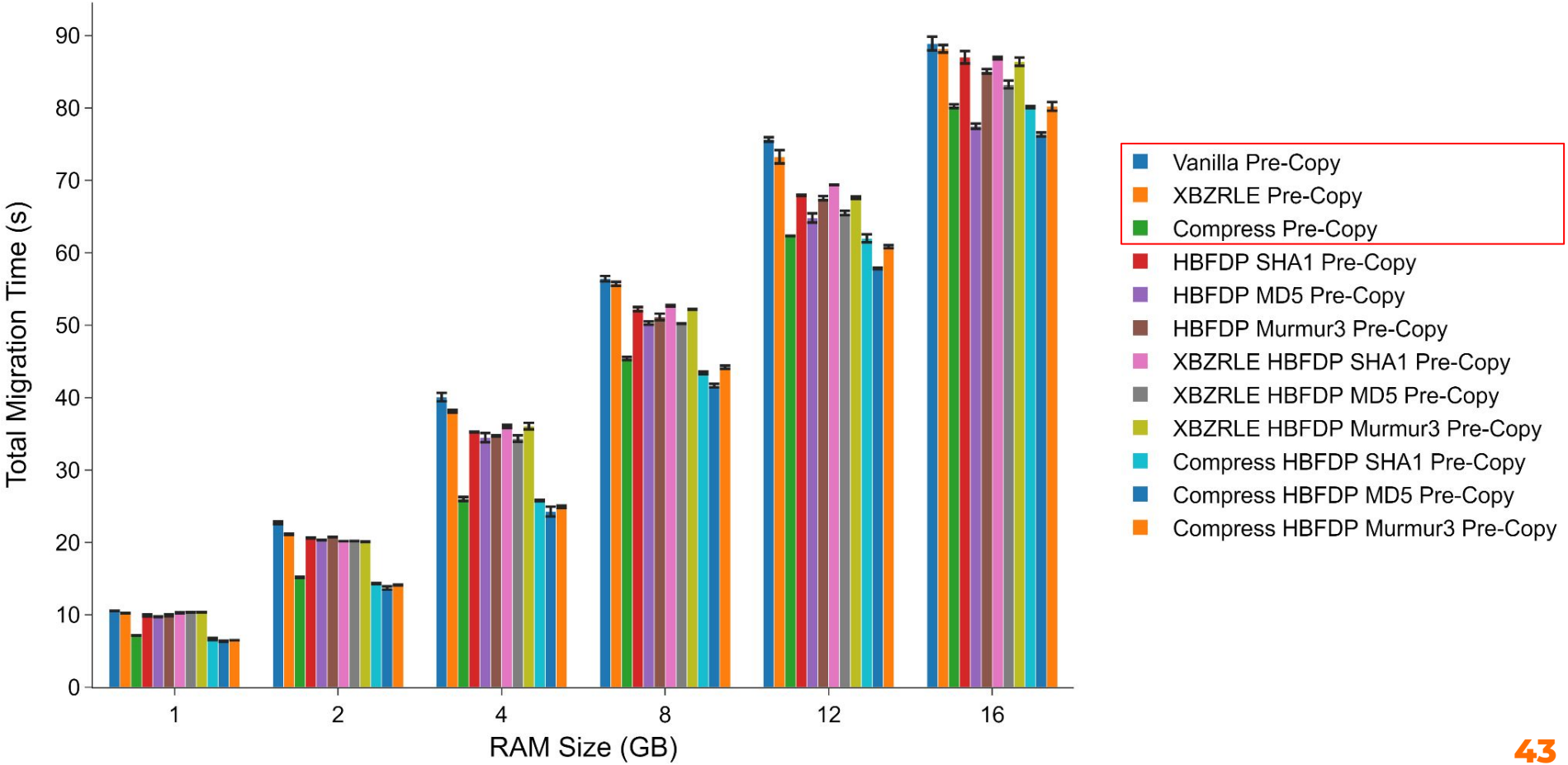
Evaluation

- The developed prototypes with and without incorporating different optimization techniques would be evaluated against
  - **Vanilla Pre-Copy**,
  - **XBZRLE enabled Pre-Copy**
  - **Compression enabled Pre-Copy**
  - **the algorithm proposed by Li et al. (2019)** (Prototype developed with SHA1 hashes).

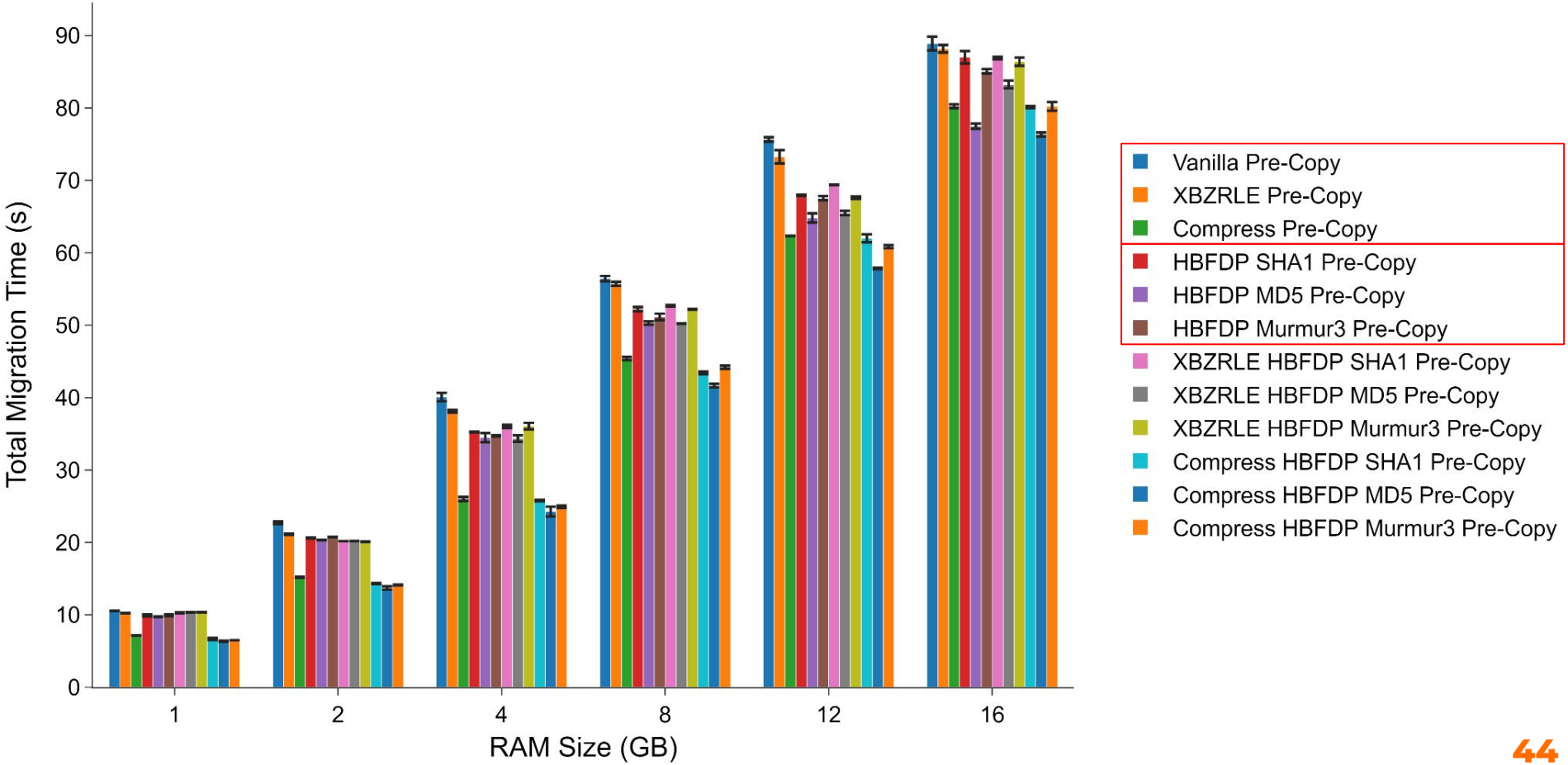
Total Migration Time for Different Precopy Migration Techniques



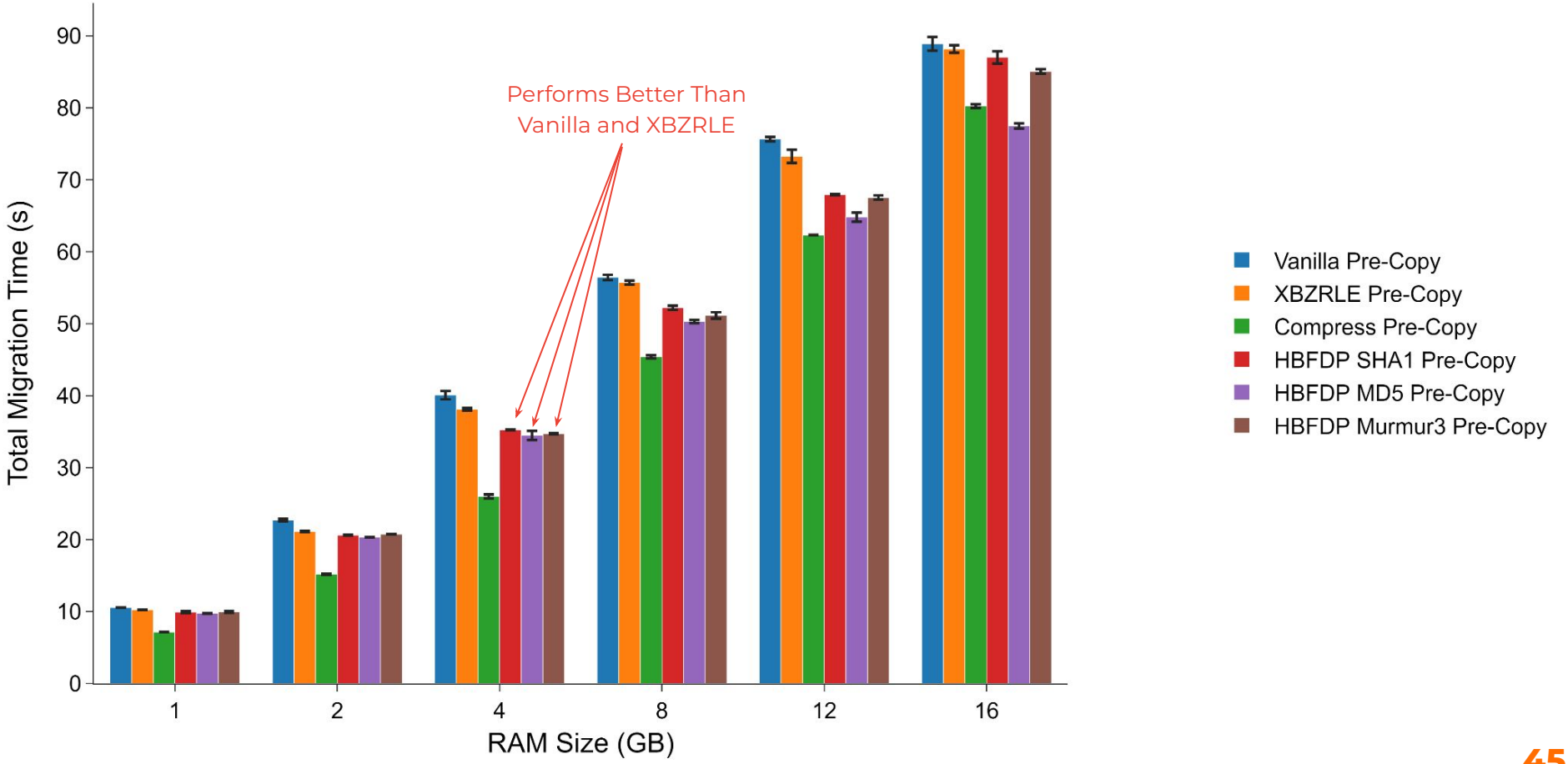
Total Migration Time for Different Precopy Migration Techniques



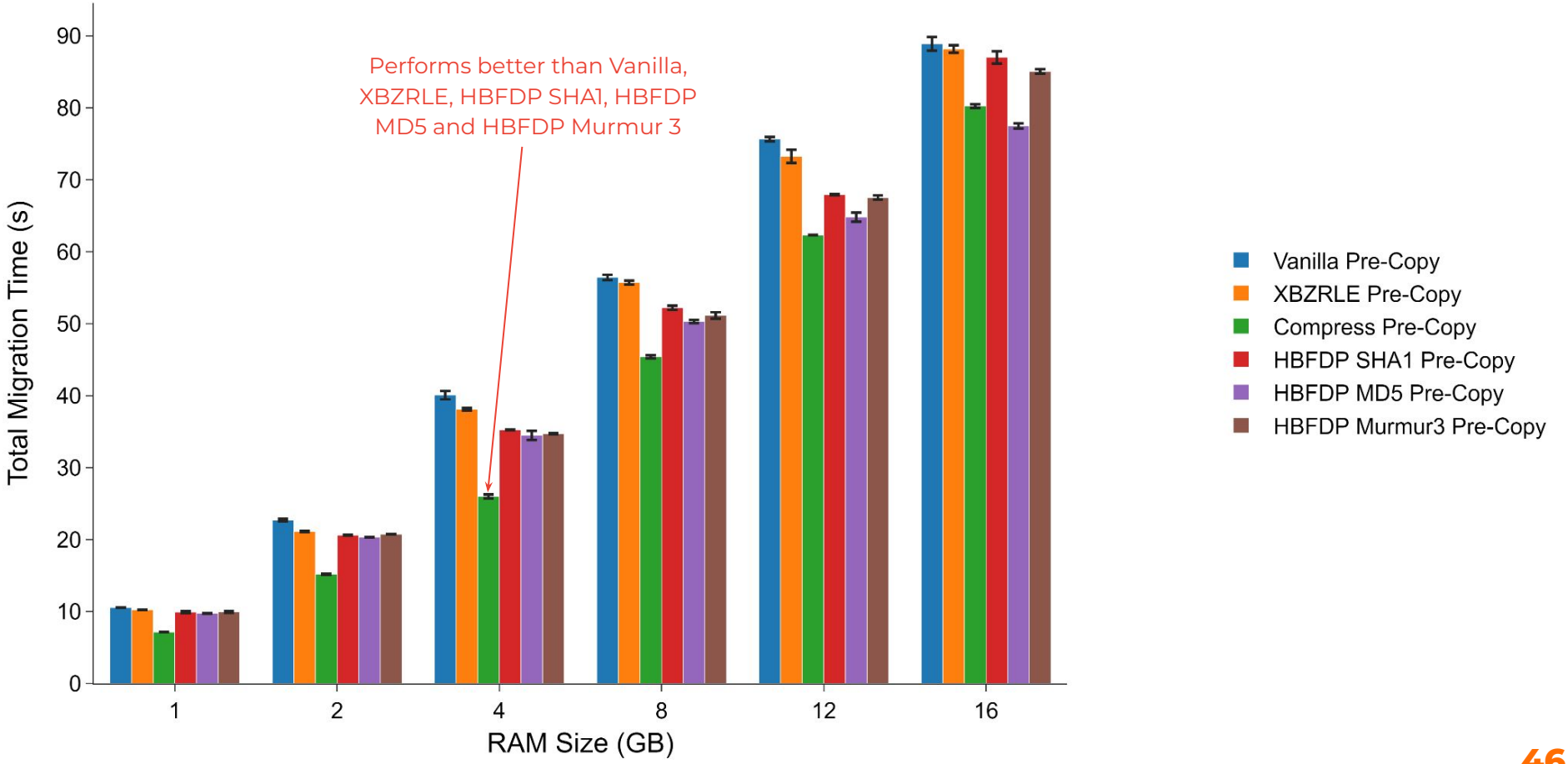
Total Migration Time for Different Precopy Migration Techniques



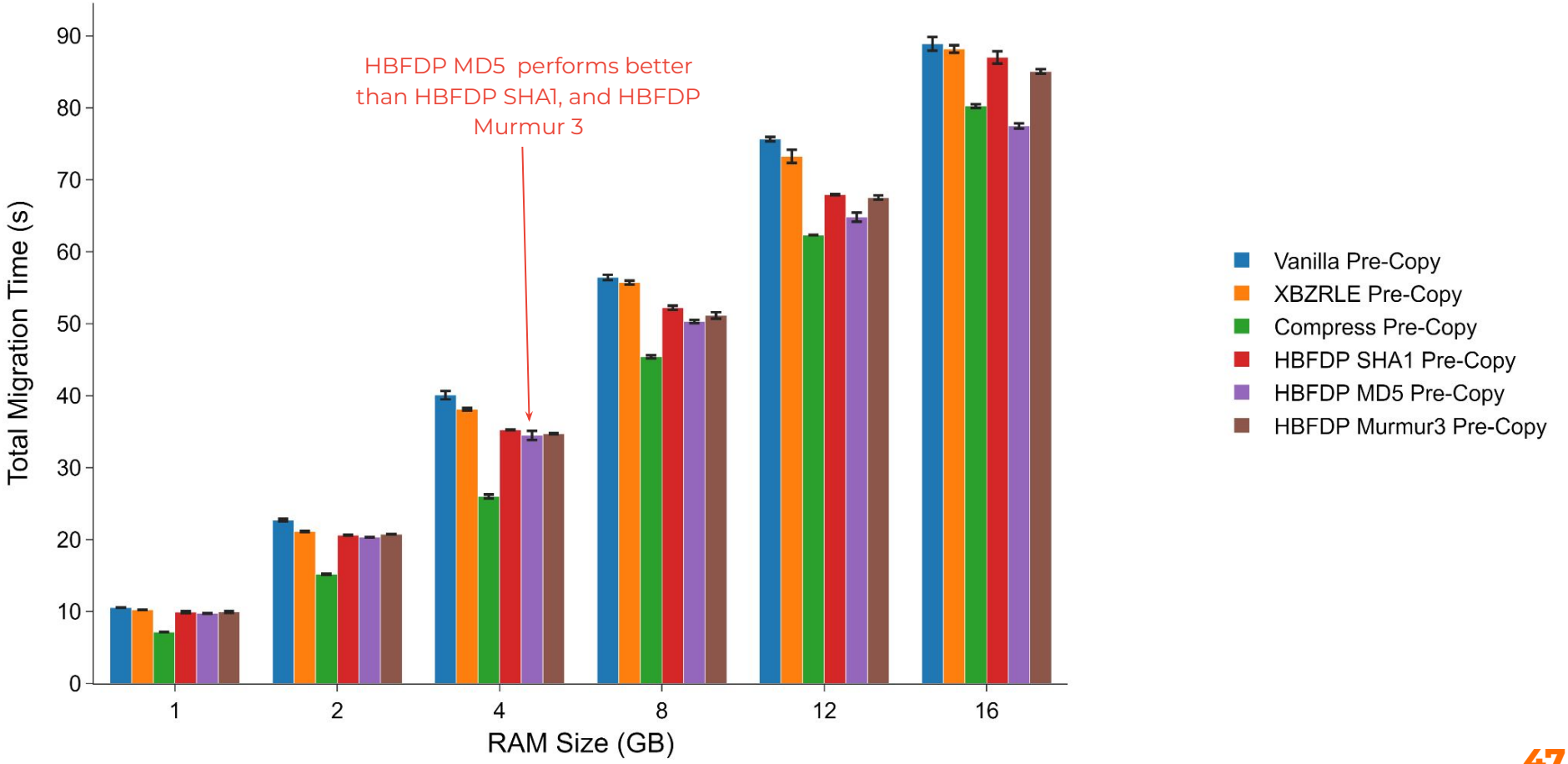
Total Migration Time for Different Precopy Migration Techniques



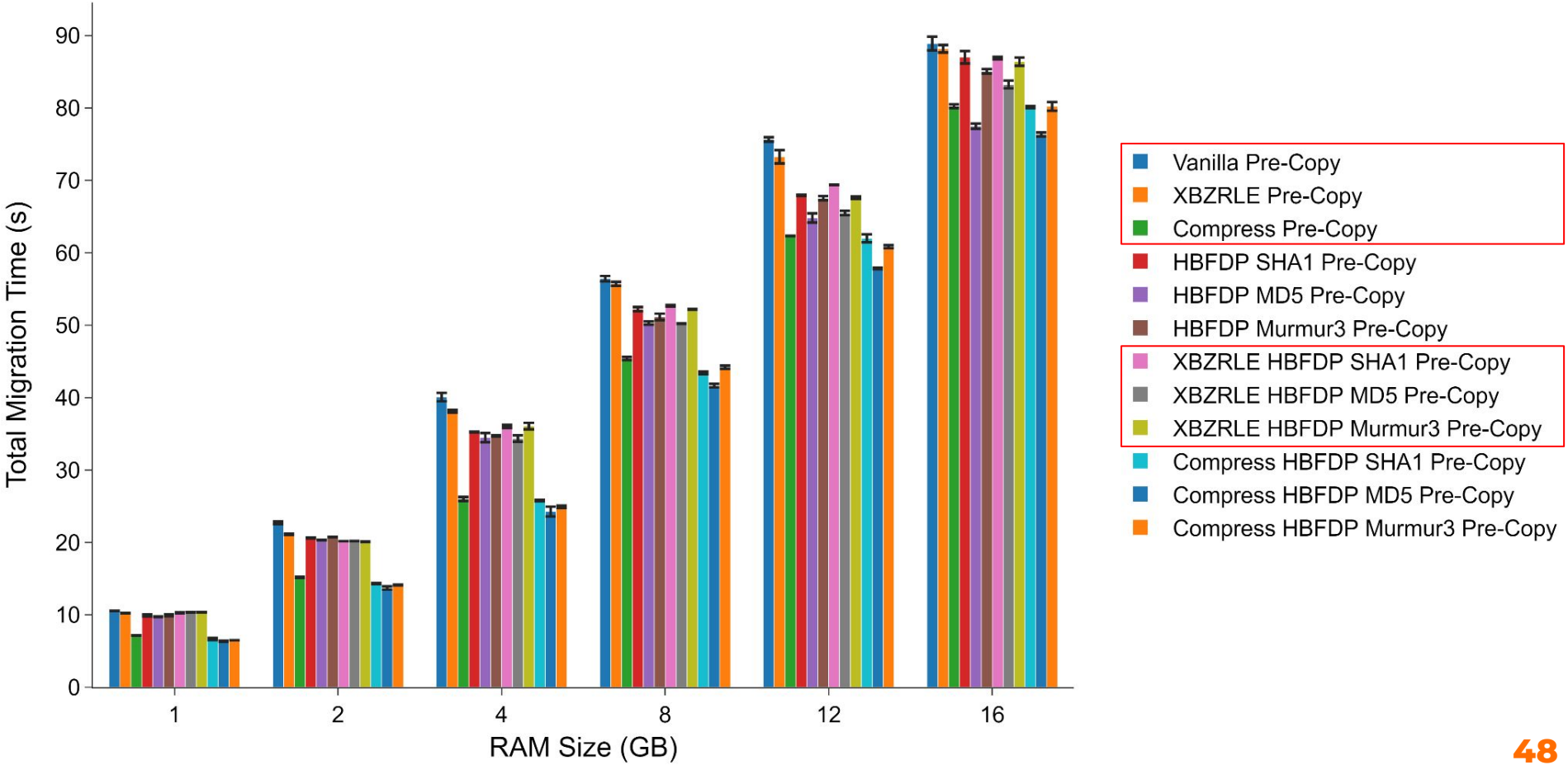
Total Migration Time for Different Precopy Migration Techniques



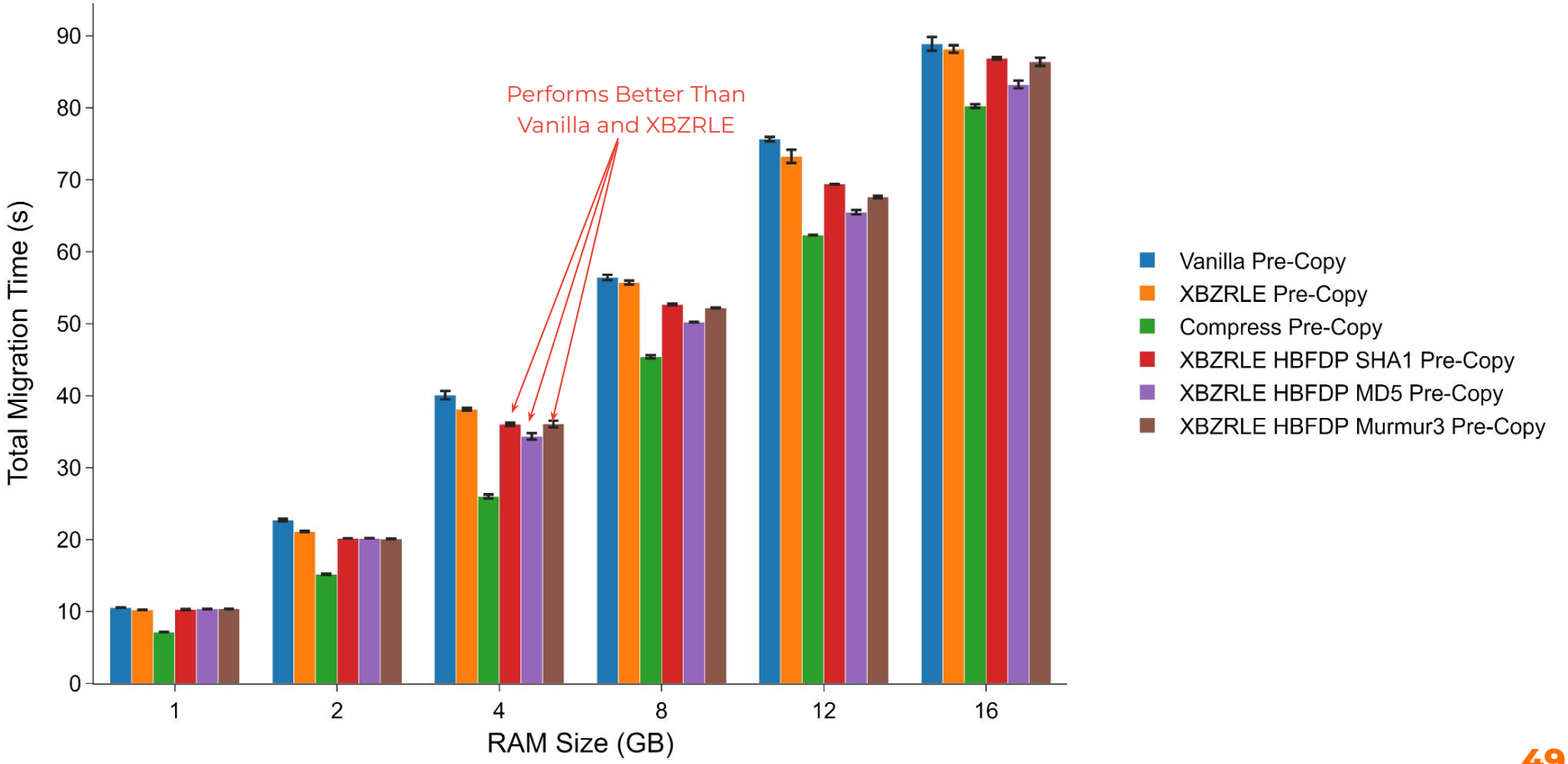
Total Migration Time for Different Precopy Migration Techniques



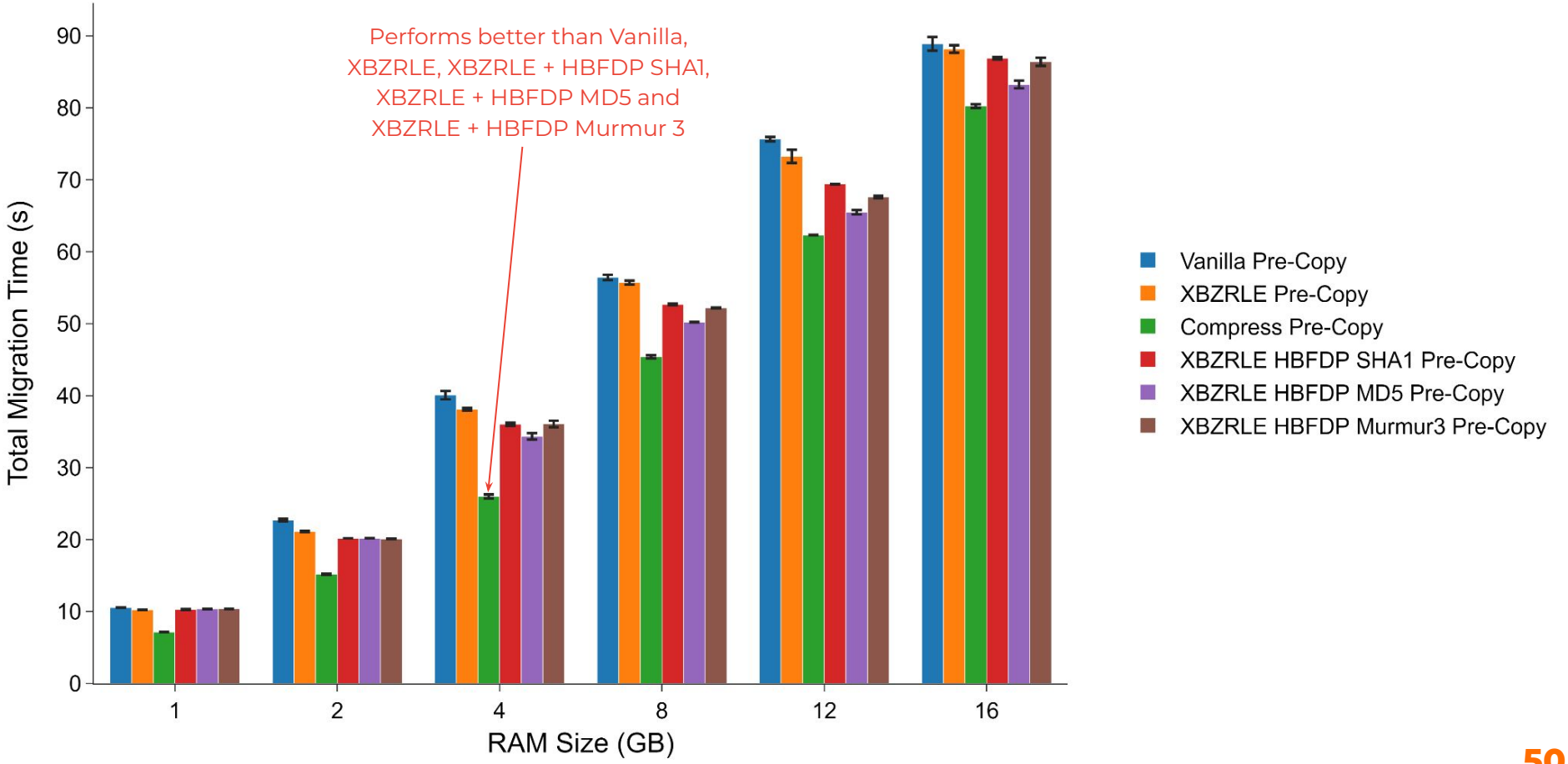
Total Migration Time for Different Precopy Migration Techniques



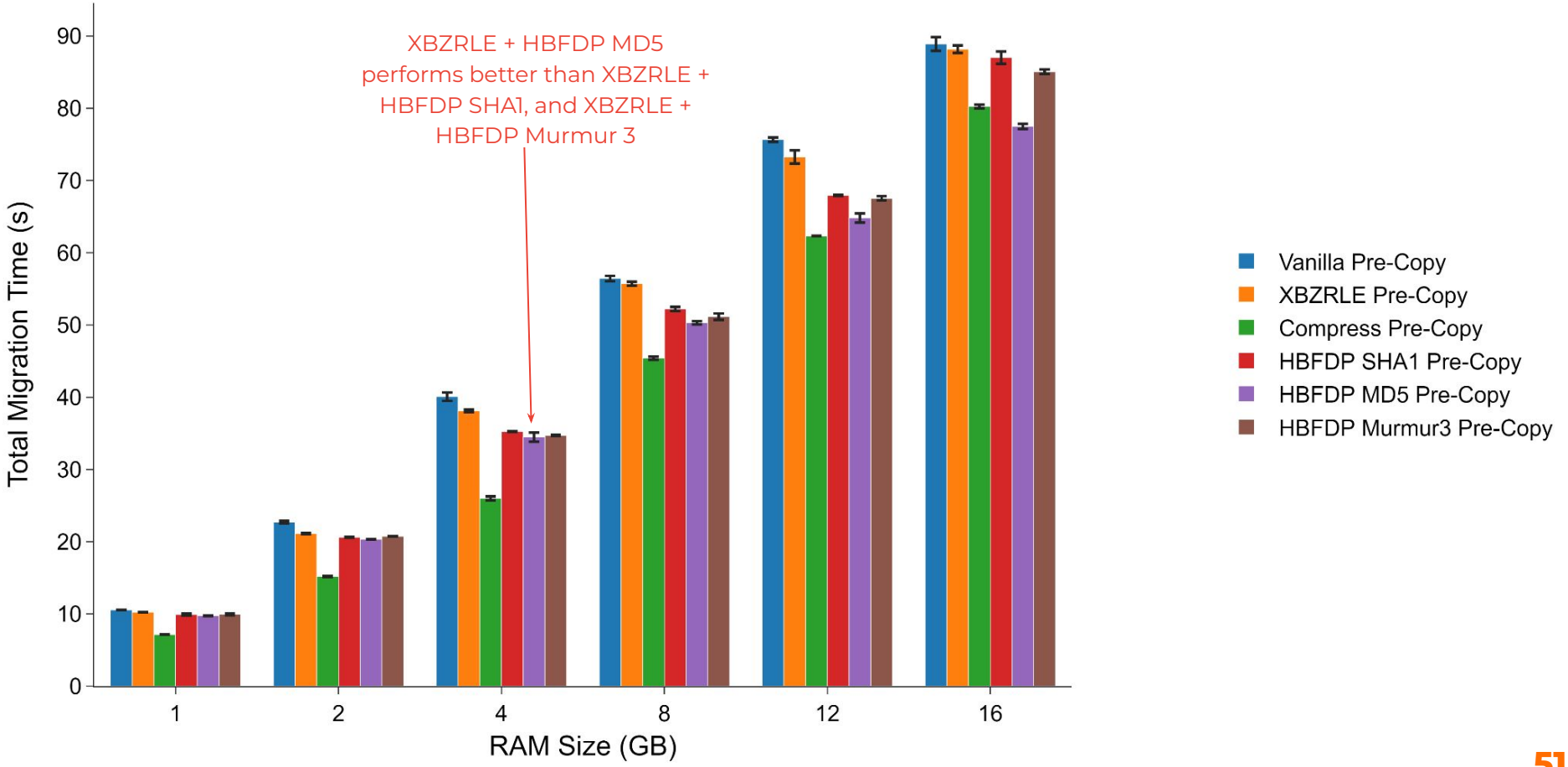
Total Migration Time for Different Precopy Migration Techniques



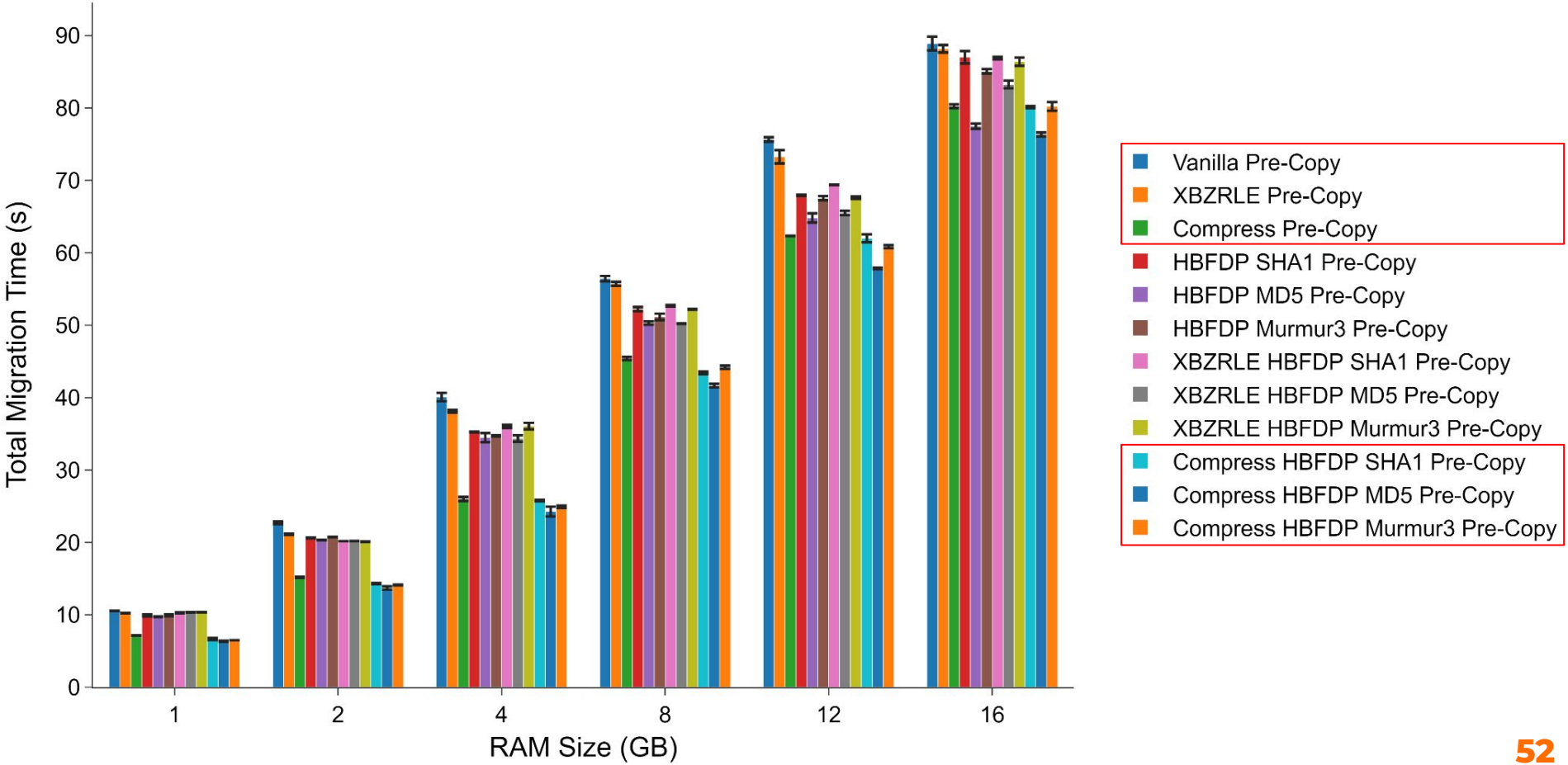
Total Migration Time for Different Precopy Migration Techniques



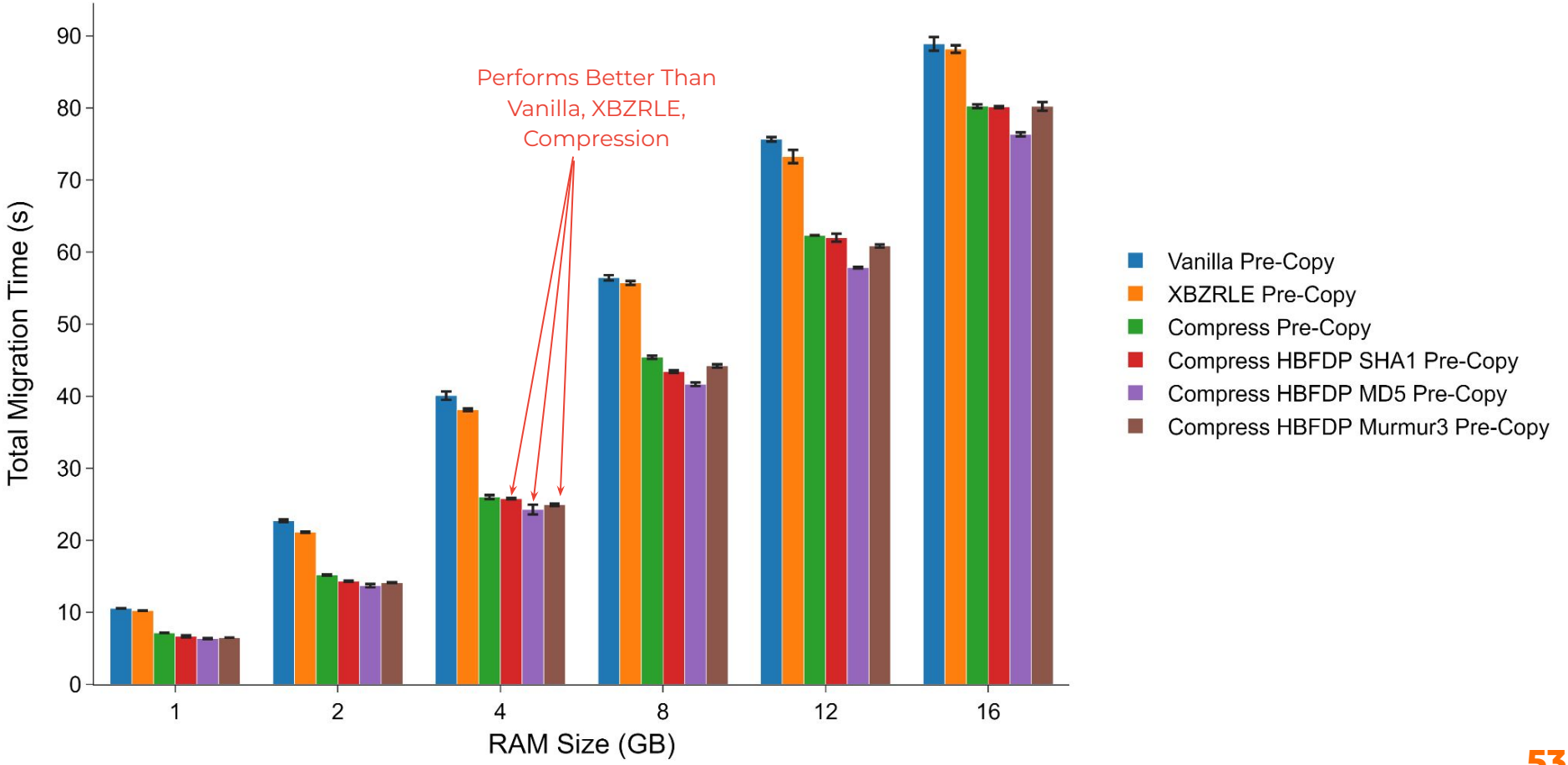
Total Migration Time for Different Precopy Migration Techniques



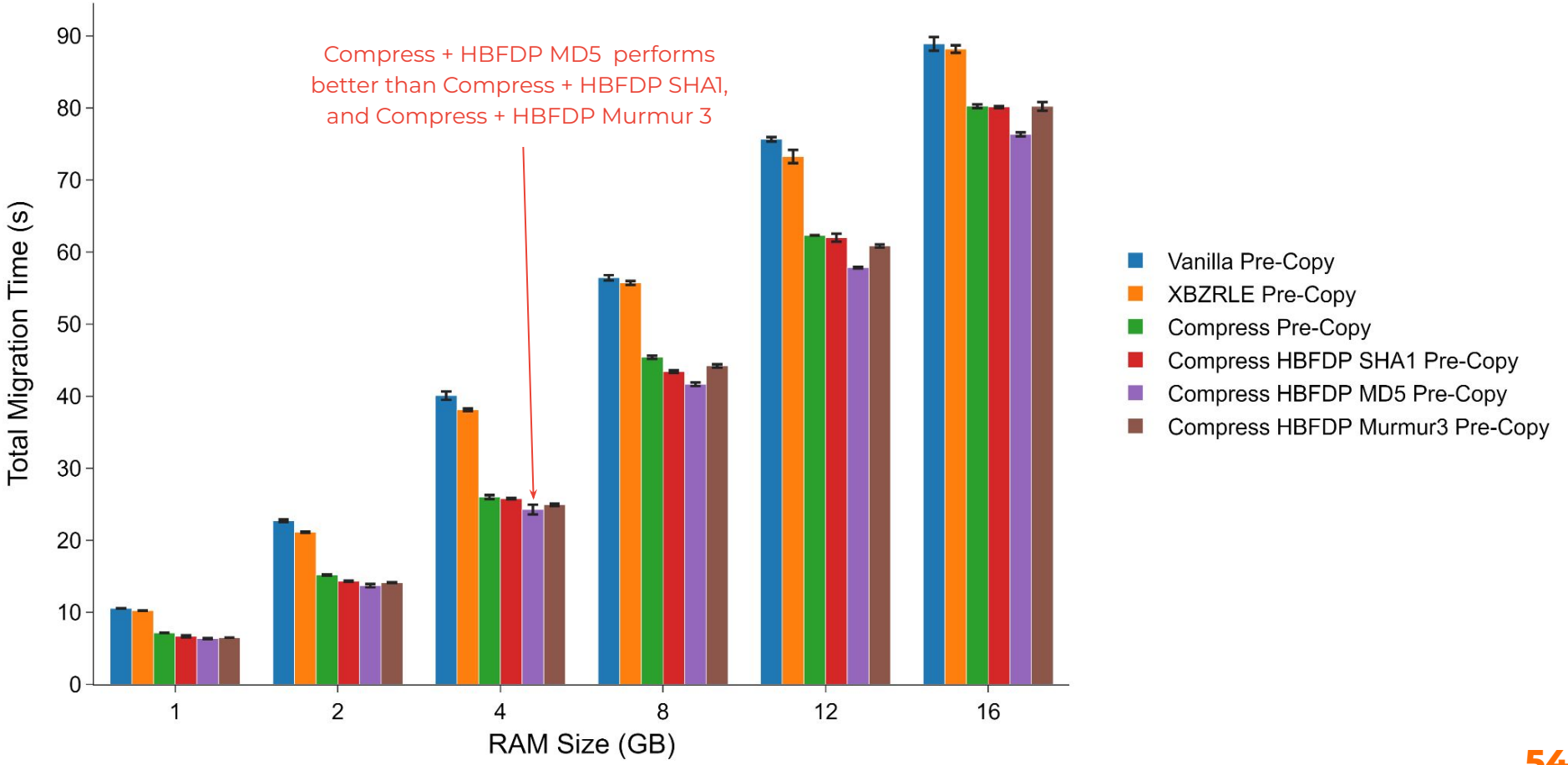
Total Migration Time for Different Precopy Migration Techniques



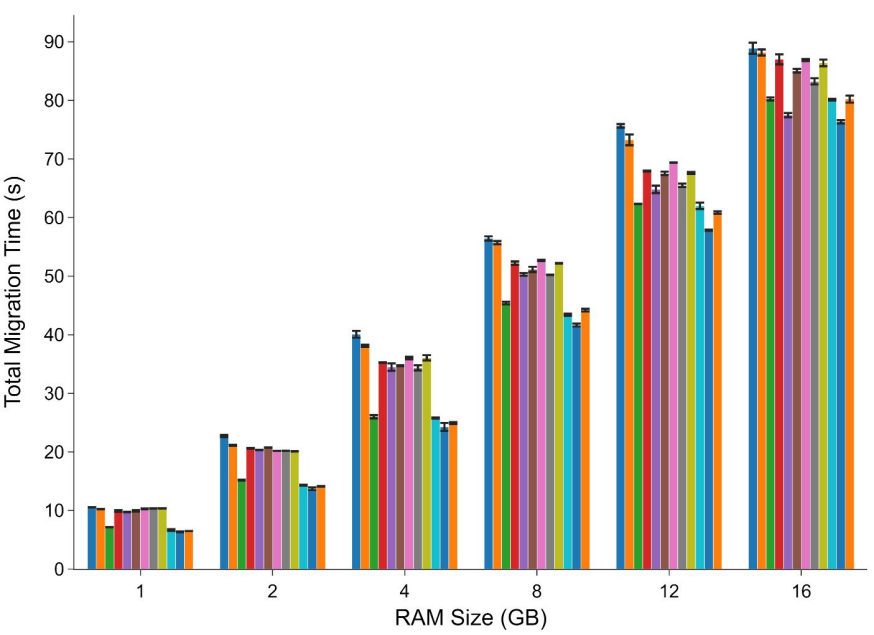
Total Migration Time for Different Precopy Migration Techniques



Total Migration Time for Different Precopy Migration Techniques



Evaluation



- **MD5 HBFDP + Compression Optimization performs** better than
  - **Vanilla Pre-Copy**,
  - **XBZRLE enabled Pre-Copy**
  - **Compression enabled Pre-Copy**
  - **the algorithm proposed by Li et al. (2019)** (Prototype developed with SHA1 hashes).

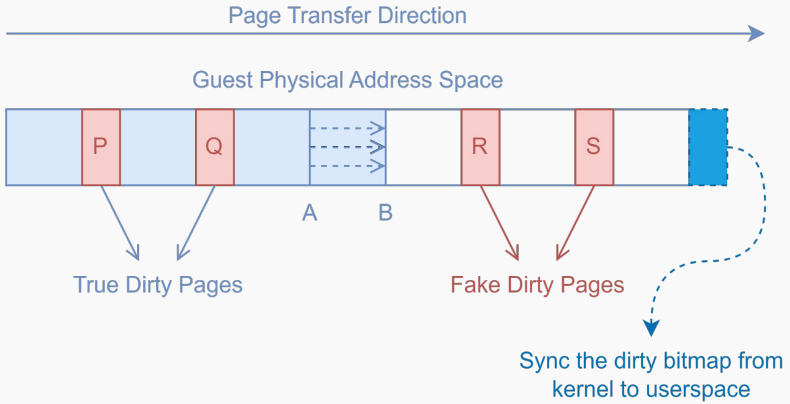
- We can see a reduction in total migration time up to 40% in **MD5 HBFDP + Compression Optimization** compared to **Vanilla Precopy**

09

Optimizations to Dirty Page Tracking

Exploration of Optimizations to Dirty Page Tracking Mechanism

- Explore optimizing the **dirty page tracking mechanism** in **Qemu** to **prevent** Fake Dirty Page transfer.



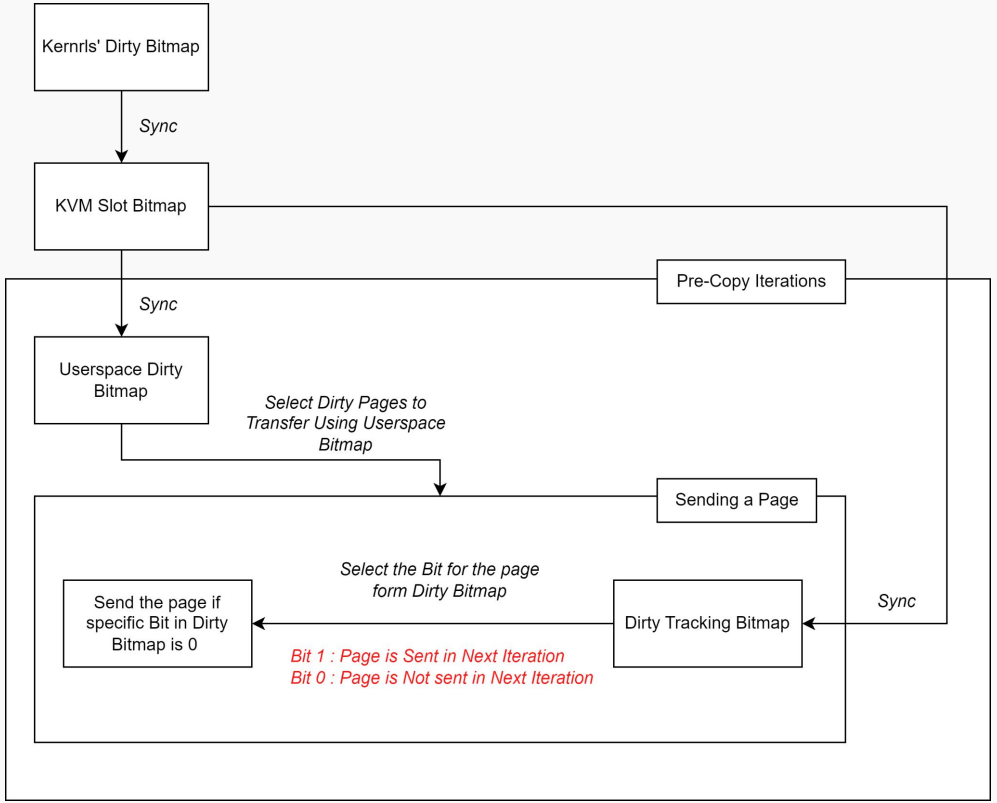
Exploration of Optimizations to Dirty Page Tracking Mechanism

- Initial approach was to find function(s) in QEMU that handles write requests issued by the Virtual Machine.
    - There is no single function that handles all guest write operations.
    - Writes happen directly from the host code generated by JIT without invoking any C code in QEMU.

- Next approach was to implement a memory listener method to capture write requests using the SIGSEGV signal issued by the operating system's kernel.
    - Locking the memory allocated to the VM and capturing segfault signals issued when VM issues a write request.
    - Ineffective since the signals the Virtual Machine issued can not be captured as a SIGSEGV signal in the QEMU source code

Exploration of Optimizations to Dirty Page Tracking Mechanism

- The current approach focuses on maintaining a **dirty page tracking bitmap** which is synced with the Kernel's dirty bitmap **before sending a page** in an iteration.
- The bitmap is used to **identify** and **not transfer** pages which are selected to transfer in the current iteration but **modified before sending** the page.



In Summary

- Live VM Migration is essential for maintaining **uninterrupted services**, achieving *fault tolerance*, *load balancing*, and *server consolidation* in cloud computing environments.
- This research focuses on enhancing pre-copy migration by addressing redundant memory page transfers, known as "**fake dirty pages**," which significantly impact total migration time.
- The study tries to mitigates these redundant transfers with the use of an optimal ashing technique, thereby improving migration efficiency by **reducing total migration time**.

Thank you !

References

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. & Warfield, A. (2003), Xen and the art of virtualization, Vol. 37.

Elsaid, M. E., Abbas, H. M. & Meinel, C. (2022), 'Virtual machines pre-copy live migration cost modeling and prediction: a survey', Distributed and Parallel Databases 40.

Hines, M. R., Deshpande, U. & Gopalan, K. (2009), Post-copy live migration of virtual machines, Vol. 43, pp. 14–26.

Jul, E., Warfield, A., Clark, C., Fraser, K., Hand, S., Hansen, J. G., Limpach, C. & Pratt, I. (2005), Live migration of virtual machines. URL: https://www.researchgate.net/publication/220831959

Li, C., Feng, D., Hua, Y. & Qin, L. (2019), 'Efficient live virtual machine migration for memory write-intensive workloads', Future Generation Computer Systems 95.

Nathan, S., Bellur, U. & Kulkarni, P. (2016), On selecting the right optimizations for virtual machine migration.

Rayaprolu, A. (2024), 'How Many Companies Use Cloud Computing in 2024? All You Need To Know'. URL: https://techjury.net/blog/how-many-companies-use-cloud-computing/

Ruprecht, A., Jones, D., Shiraev, D., Harmon, G., Spivak, M., Krebs, M., Baker-Harvey, M. & Sanderson, T. (2018), 'Vm live migration at scale', ACM SIG-PLAN Notices 53(3), 45–56

Sahni, S. & Varma, V. (2012), A hybrid approach to live migration of virtual machines.

Workloads

QuickSort

The Quicksort benchmark was configured to repeatedly create an array of size 512mb, store random numbers in each array index, and sort the array using the quicksort algorithm.

- **In-place sorting**, meaning it does not require additional memory proportional to the size of the array. Instead, it works by rearranging the elements within the original array using only a few extra variables.
- **Pivot:** For the given partition the pivot is picked randomly, then the pivot is transferred to the end of the partition. Next the partition is divided into two taking the values smaller than to the pivot to the left and values greater than the pivot to right. Then the pivot is placed in the middle. Next the quicksort is again applied to the two parts separately.

Workingset

This workload create a array for the user entered size and allocate random numbers, then it divided the array into pages of size 4KB and repeatedly allocates random numbers to some addresses by taking each page.

Workloads

Sysbench

The Sysbench benchmark was configured to calculate prime numbers up to 5,000,000 to stress the CPU.

Memcached

The Memcached workload was configured by allocating 4 threads to handle requests and 90% of the VM memory size was allocated as Memcached cache, Memaslap benchmark was configured in the a third node to send requests to the Memcached in the VM with set and get ratio of 1:0.

YCSB

The YCSB benchmark was executed in the Virtual Machine to send queries to a PostgreSQL database hosted in the NFS server with 10,000 operations per second.

Hash Collisions