Final

D-Track : Efficient Fake Dirty Page Tracker for Live VM Migration

Samindu R. Cooray | CloudNet Research Group

Supervisor: Dr. Dinuni K. Fernando

Table of Contents

01 Significance

02 Background

03 Problem Statement

04 Related Work

05 Research Gap

06 Objectives

07 Preliminary Study

08 Dirty Page Tracking Optimizations

09 D-Track

10 Implementation

11 Evaluation

12 Addressing Research Question

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

Google

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

Jul et al., Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, 2005

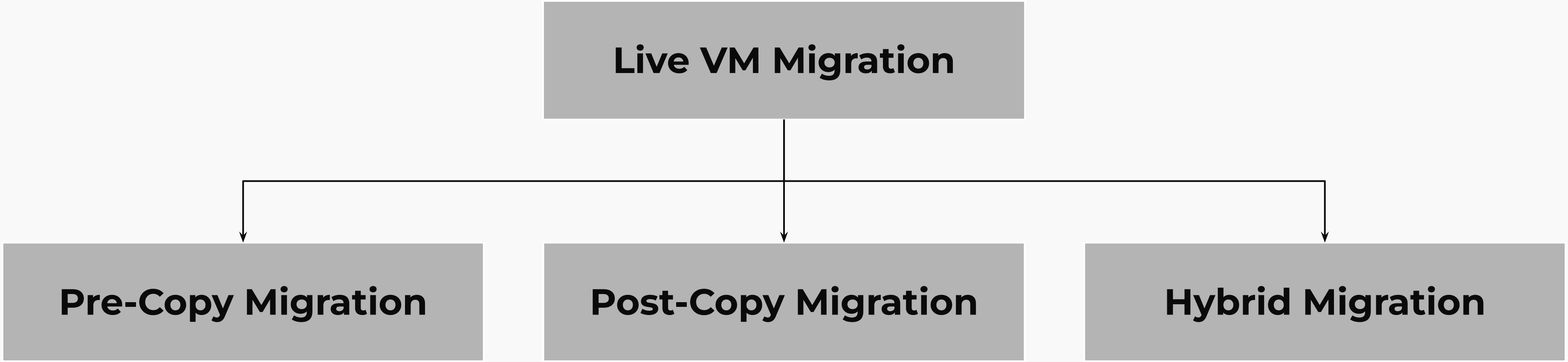
Background

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



Live VM Migration

Pre-Copy Migration

Post-Copy Migration

Hybrid Migration

Jul et al., Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, 2005

Background

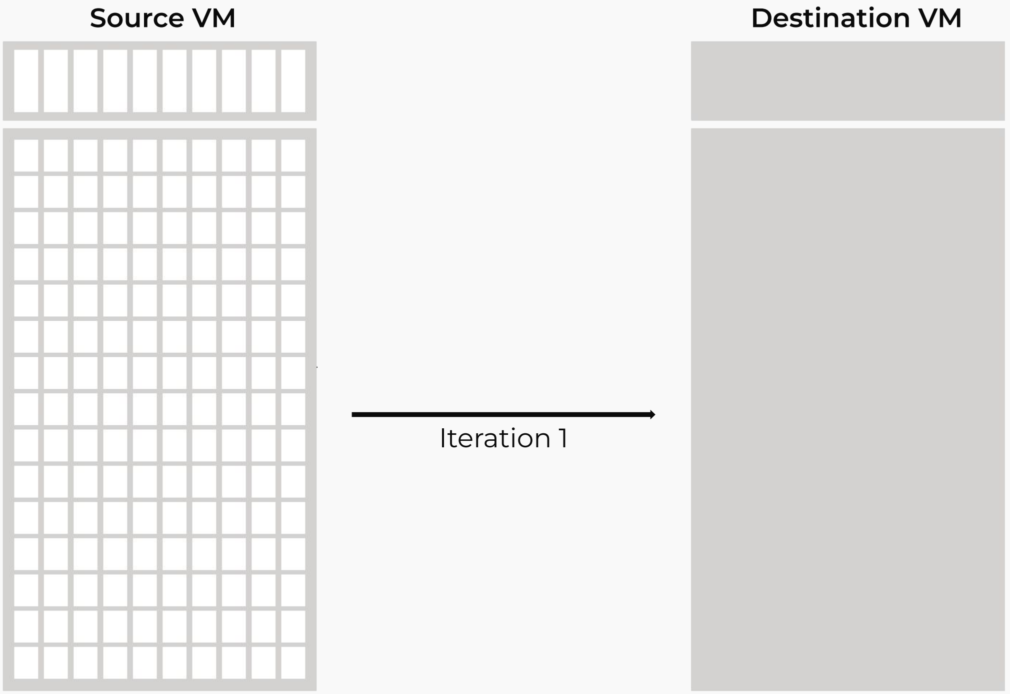
Pre-Copy Migration

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.



Jul et al., Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, 2005

Background

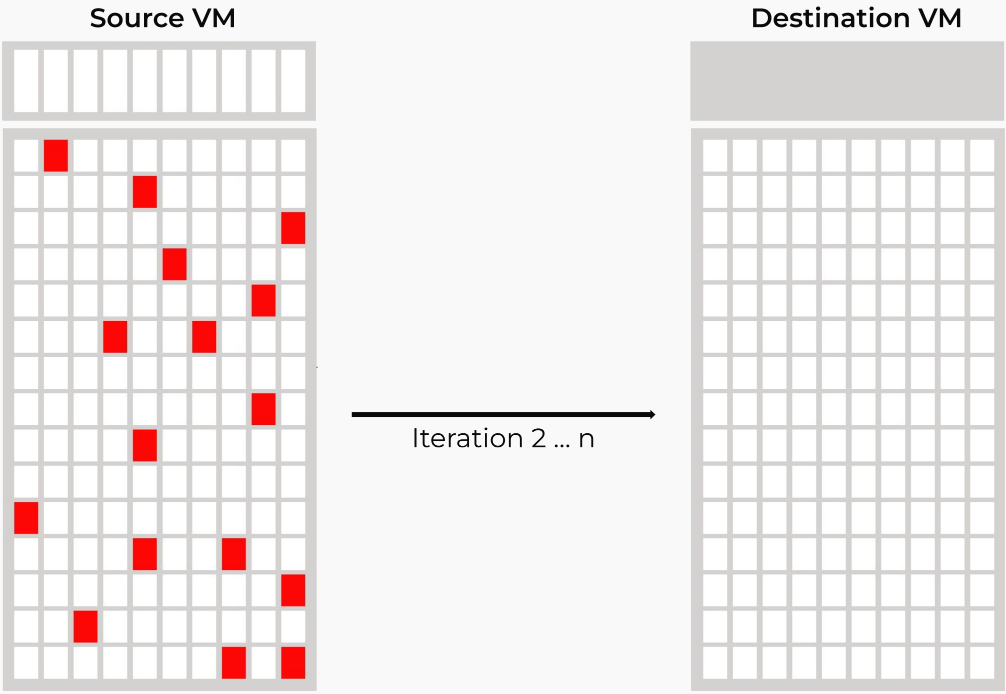
Pre-Copy Migration

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.



Jul et al., Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, 2005

Background

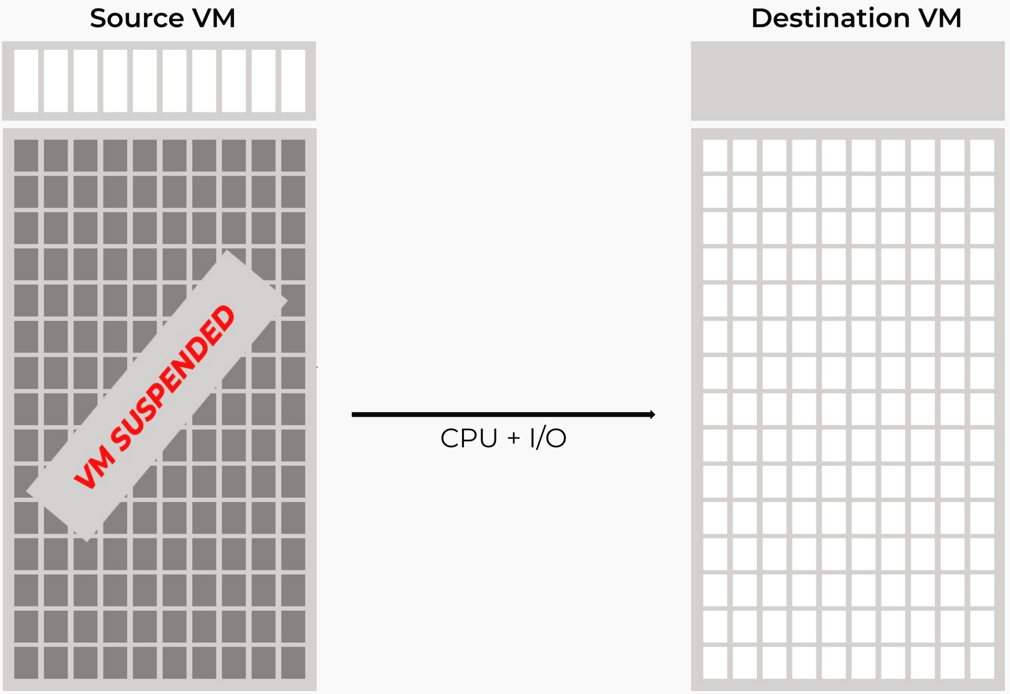
Pre-Copy Migration

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.



Jul et al., Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, 2005

Background

Pre-Copy Migration

Push Phase

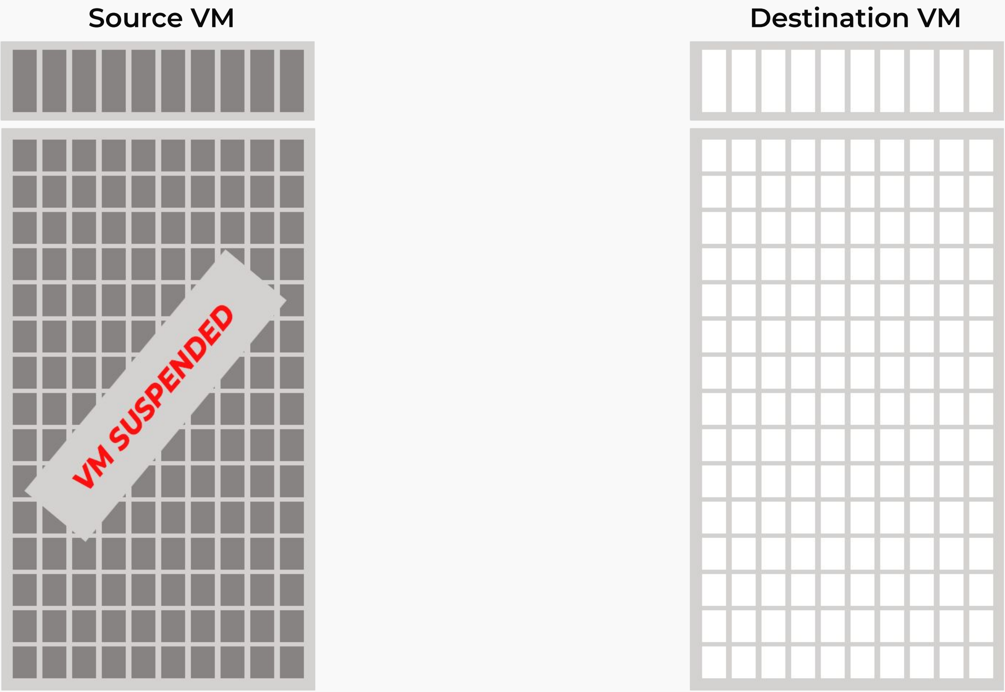
1. Transfer all memory pages to the destination.

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.



Hines et al., ACM SIGOPS Operating Systems Review, 2009

Background

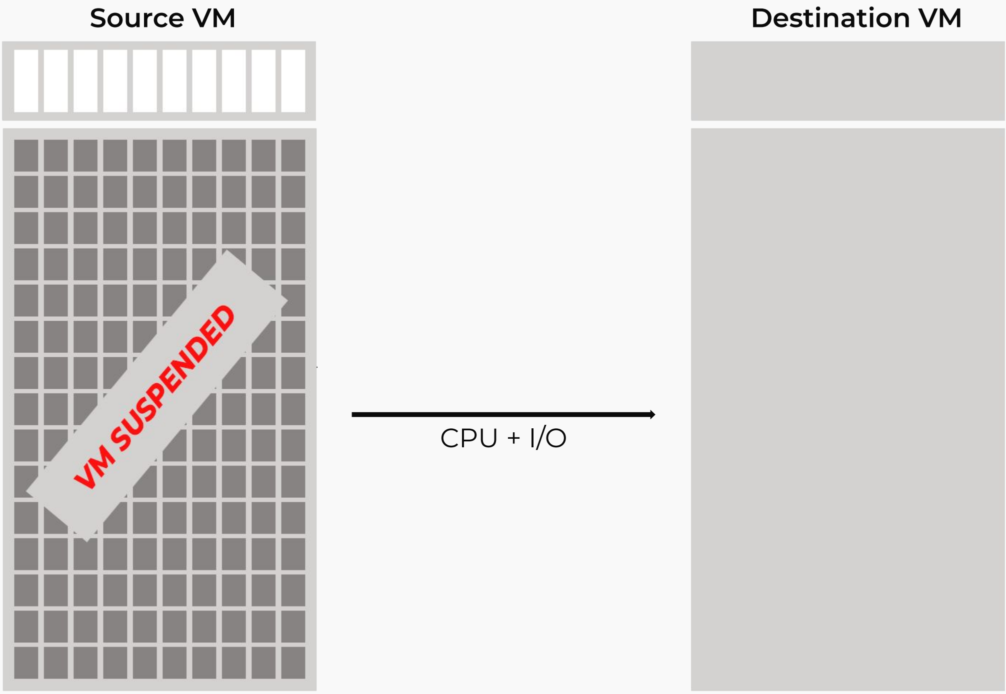
Post-Copy Migration

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.



Hines et al., ACM SIGOPS Operating Systems Review, 2009

Background

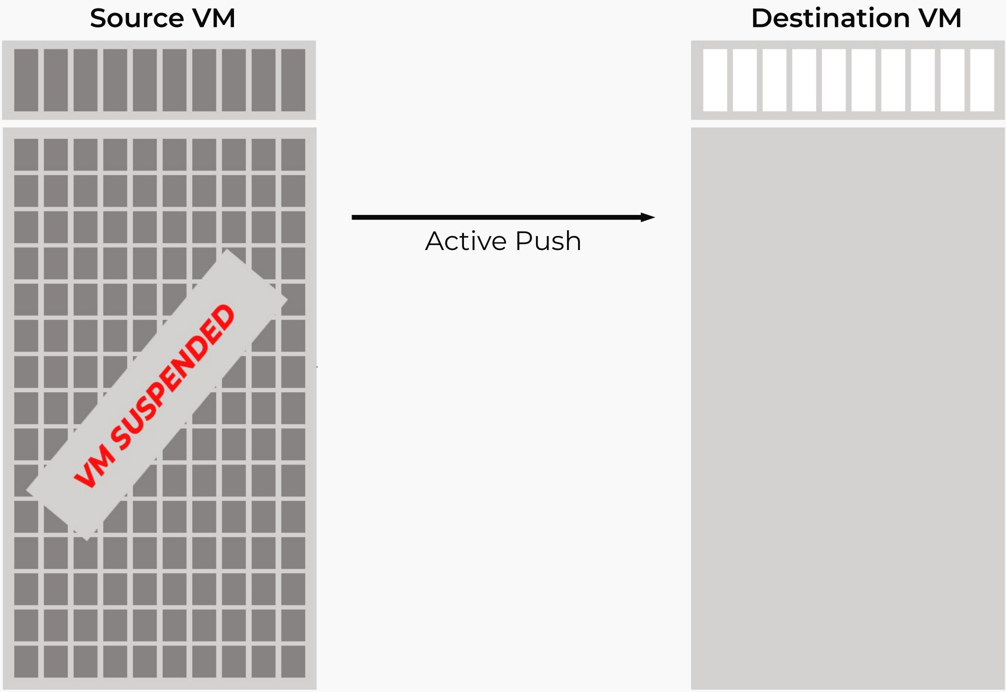
Post-Copy Migration

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.



Hines et al., ACM SIGOPS Operating Systems Review, 2009

Background

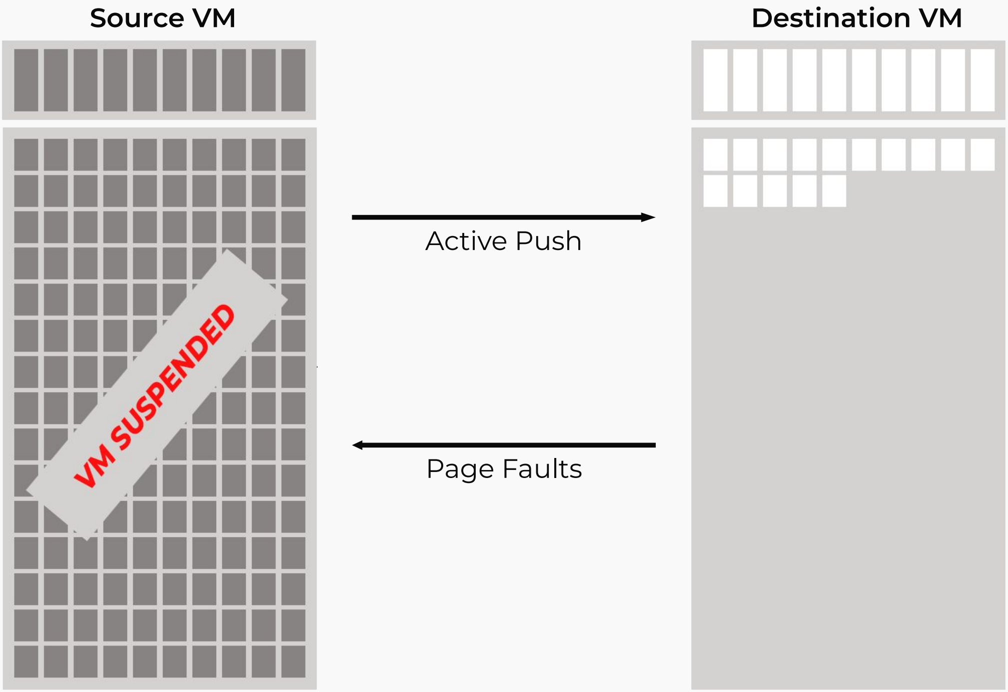
Post-Copy Migration

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.



Sahni & Varma, 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2012

Background

Hybrid Migration

Push Phase

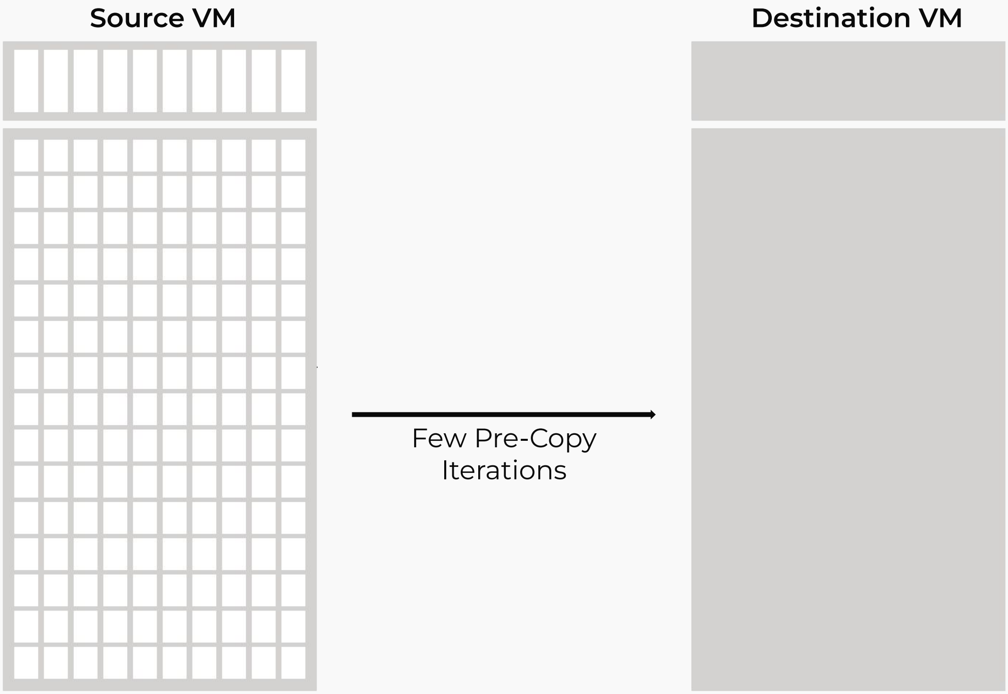
1. Transfer memory pages in few iterations

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

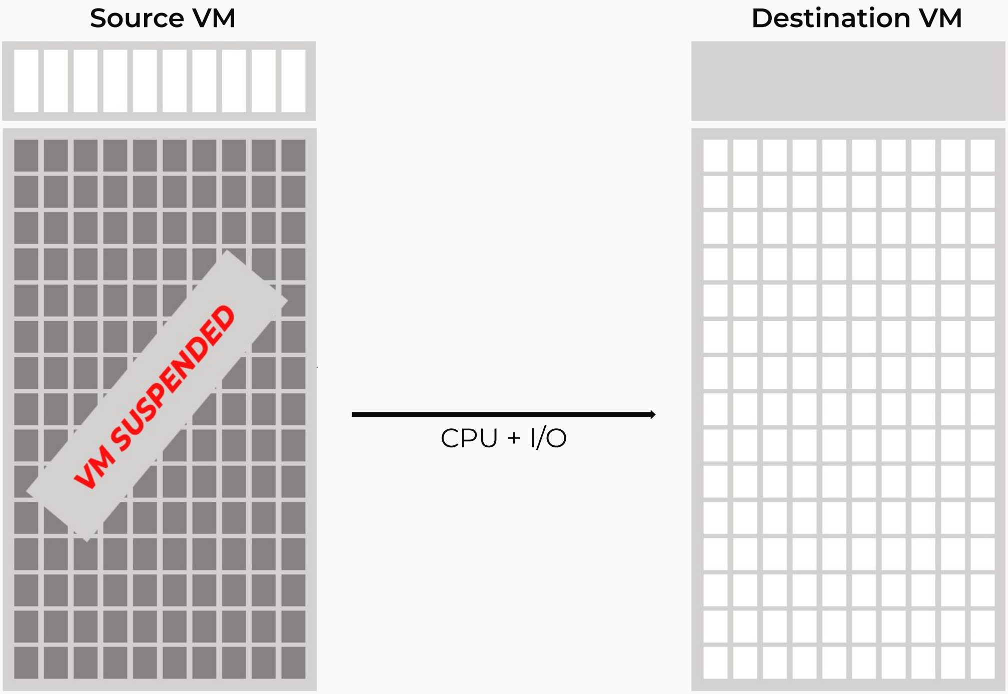
1. Transfer memory pages in few iterations

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.



Sahni & Varma, 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2012

Background

Hybrid Migration

Push Phase

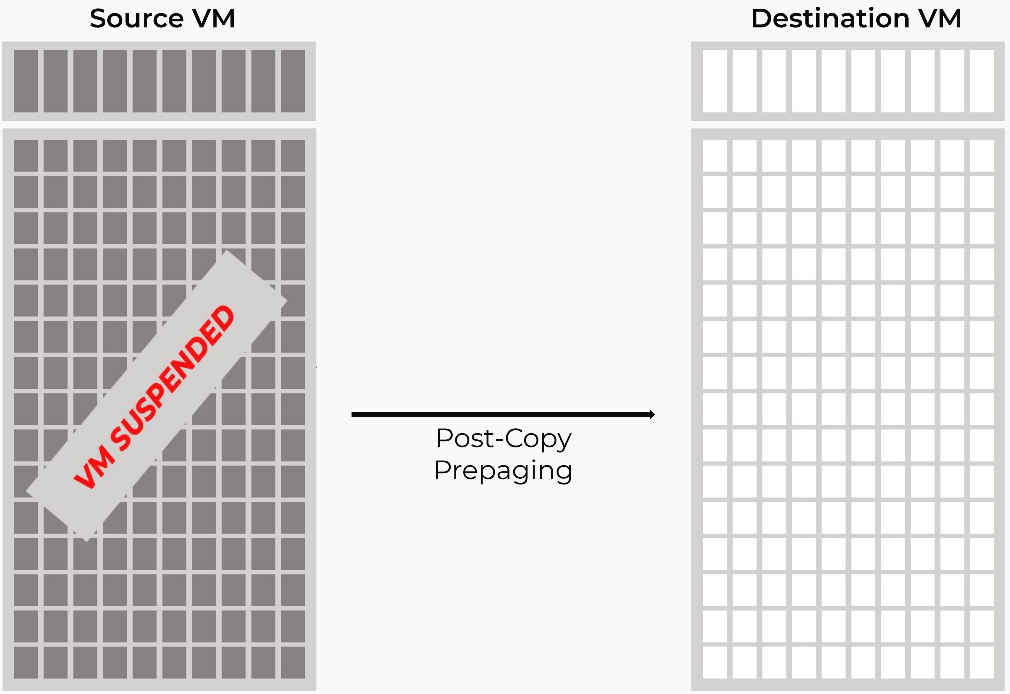
1. Transfer memory pages in few iterations

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!

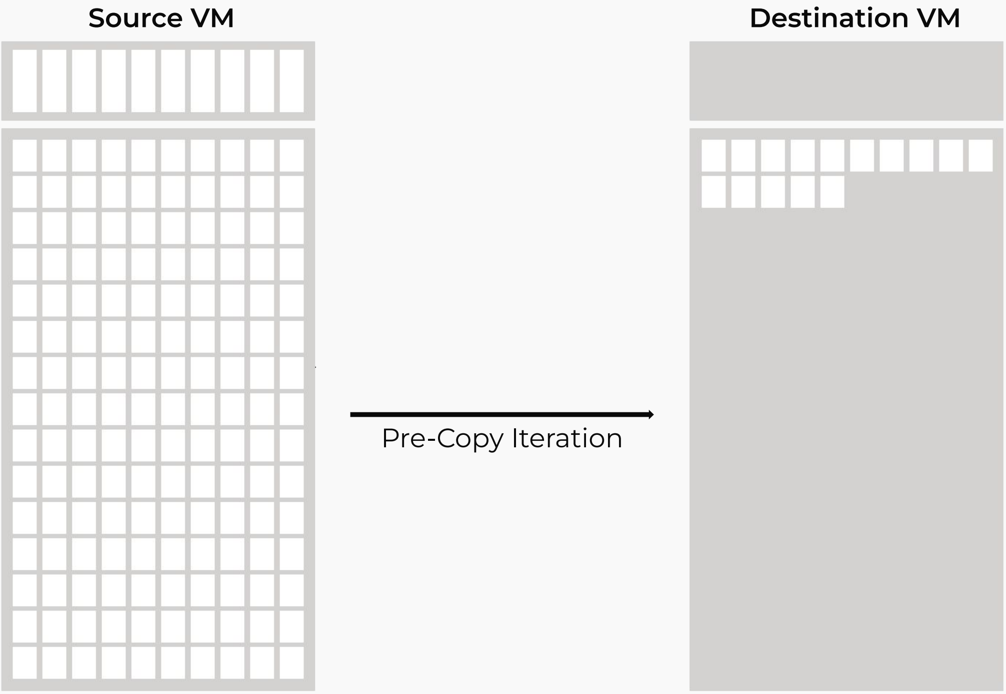
What is a Fake Dirty Page ?

Nathan et al., ACM SIGPLAN Notices, 2016
Li et al., Future Generation Computer Systems, 2019

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 .

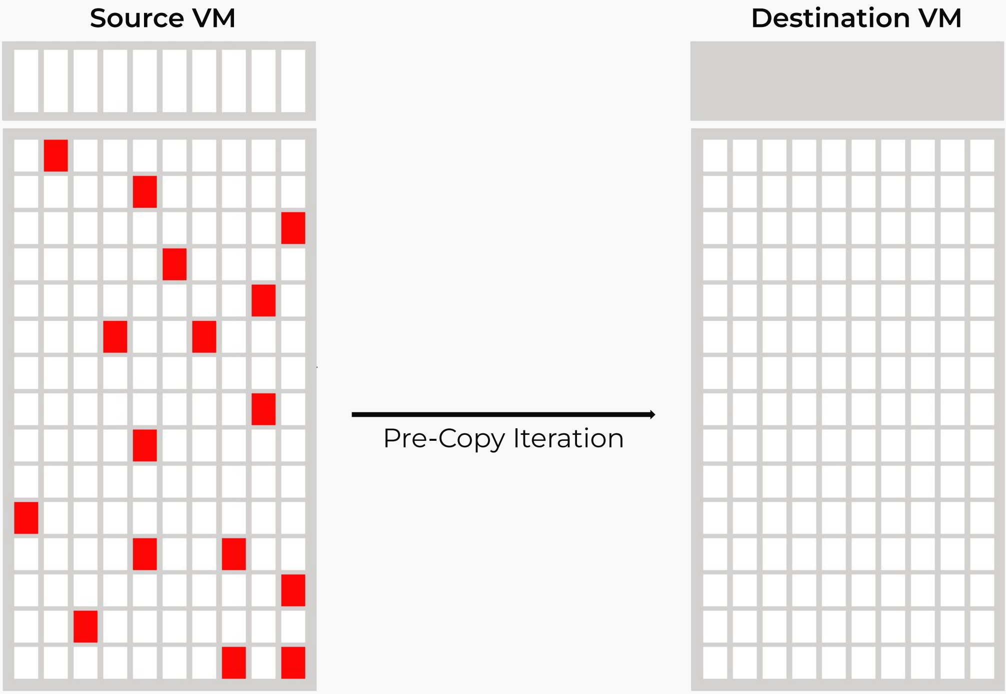


Nathan et al., ACM SIGPLAN Notices, 2016
Li et al., Future Generation Computer Systems, 2019

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 .

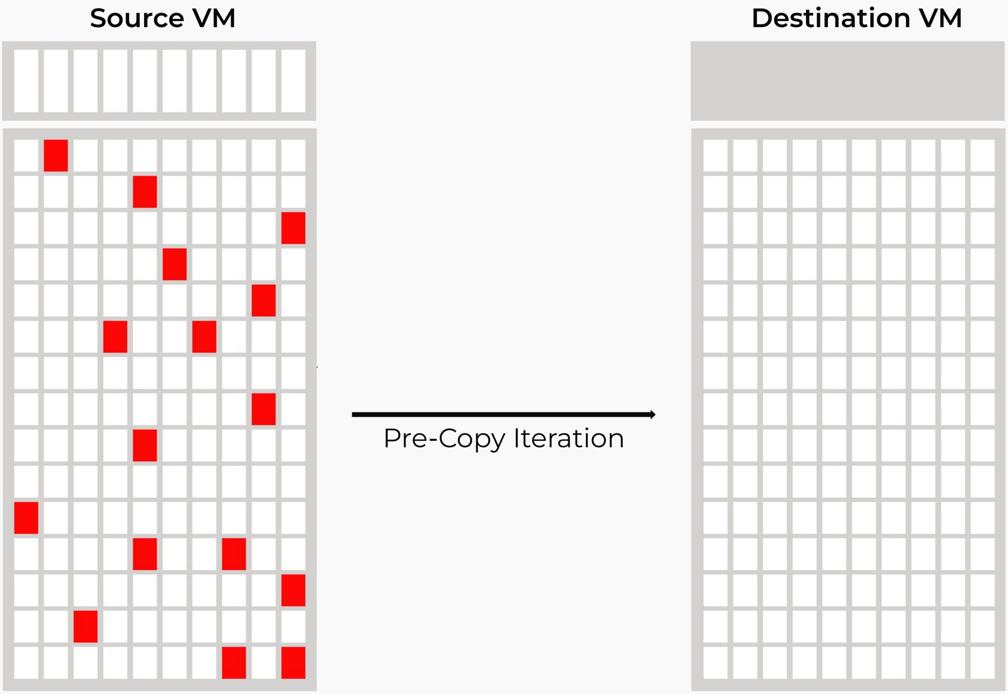


Nathan et al., ACM SIGPLAN Notices, 2016
Li et al., Future Generation Computer Systems, 2019

Problem Statement

Fake Dirty Page

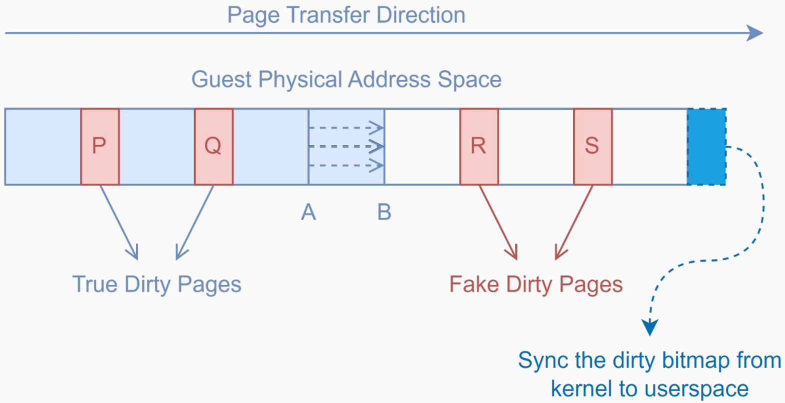
- In Reality, out of the pages marked as dirty in the previous iteration, there are some pages that **do not** have any actual **content change**. These pages are known as **Fake Dirty Pages**.
- This unnecessary duplication wastes **network bandwidth** and prolong **migration time**.



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***.

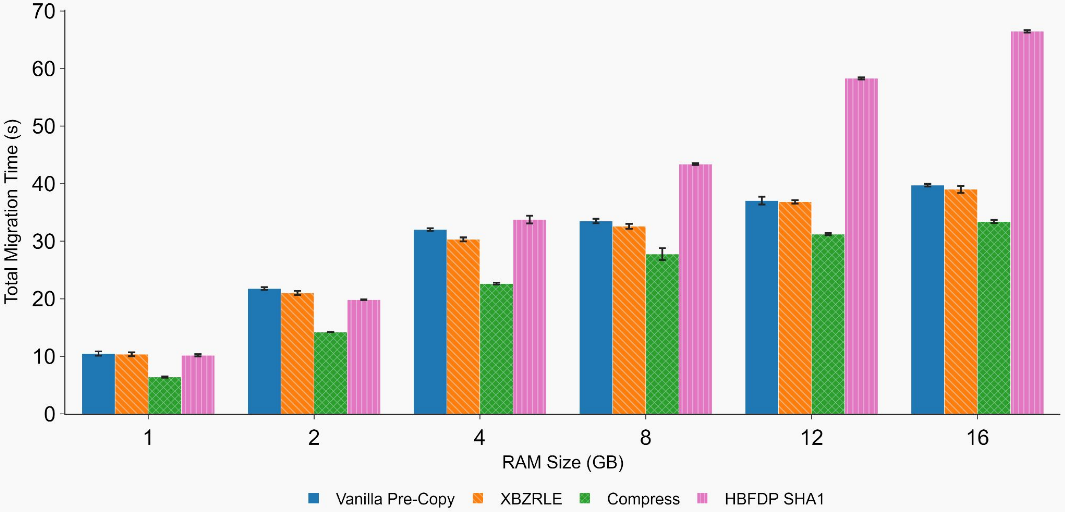

Page Transfer Direction
Guest Physical Address Space
P
Q
R
S
A
B
True Dirty Pages
Fake Dirty Pages
Sync the dirty bitmap from kernel to userspace

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

04

Related Work

Work	Approach	Key Points	Drawbacks
Li et. al. (2019)	Fake Dirty Prevention using Page Hash Comparison	Reducing Transferring Load by Mitigating Fake Dirty Page Transfer	Hash Computation Overhead on the Migration Thread
Nathan et al. (2016)	Use of Deduplication and Delta Compression Techniques	Content-based optimizations to minimize data transfer	Not completely stopping Fake Dirty Page Transfer



05

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.
- 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.

06

Objectives

Objectives

01 To evaluate the performance improvement of optimizations to **fake dirty tracking mechanism** 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?

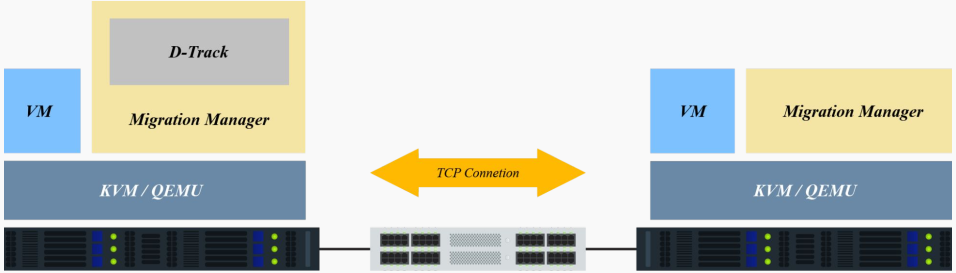
07

Preliminary Experiments

Preliminary Experiments

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

Preliminary Experiments

Selecting Suitable Workloads

Workloads	Intensive Type	Description
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	Memory	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.

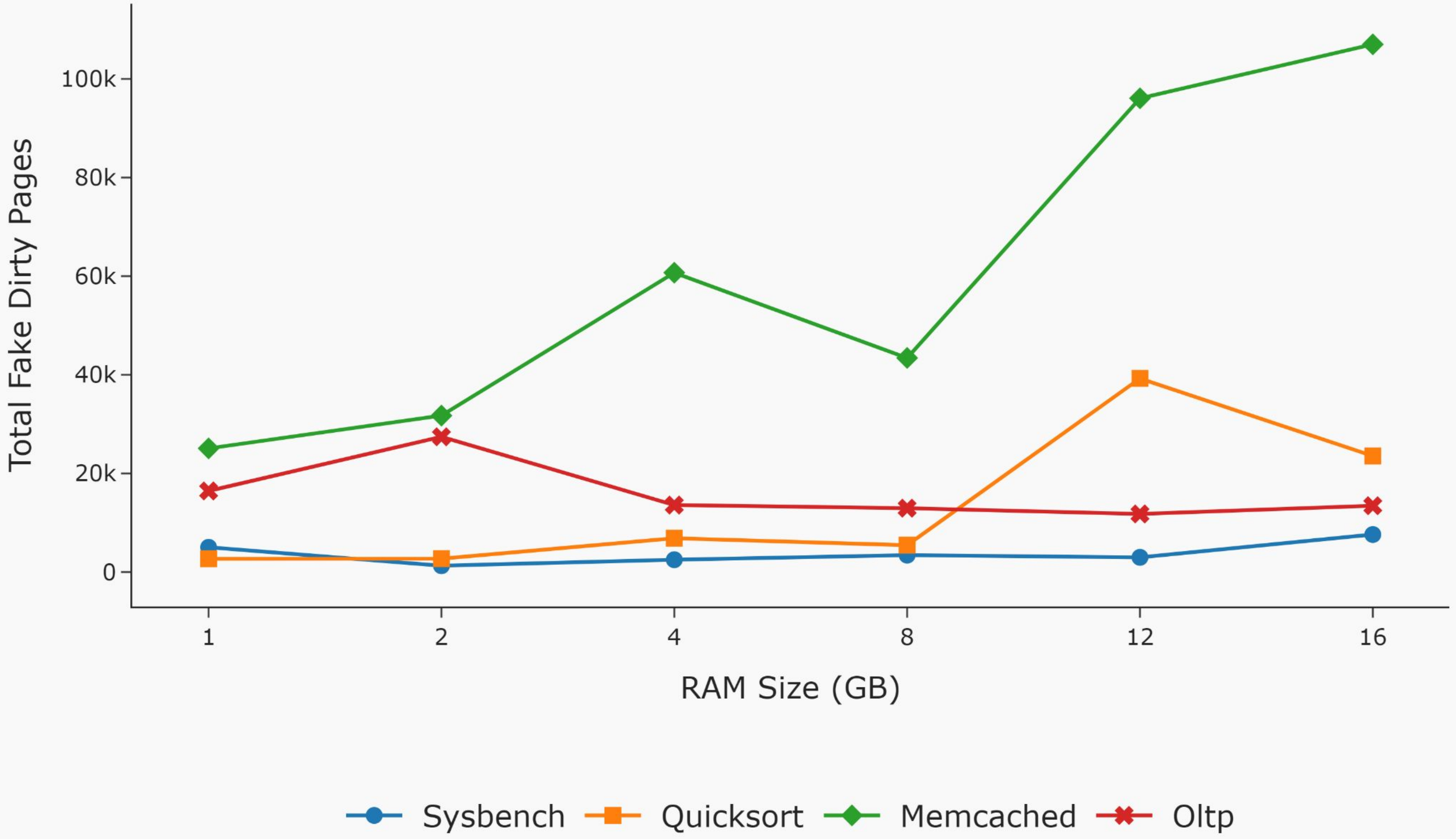
Preliminary Experiments

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.

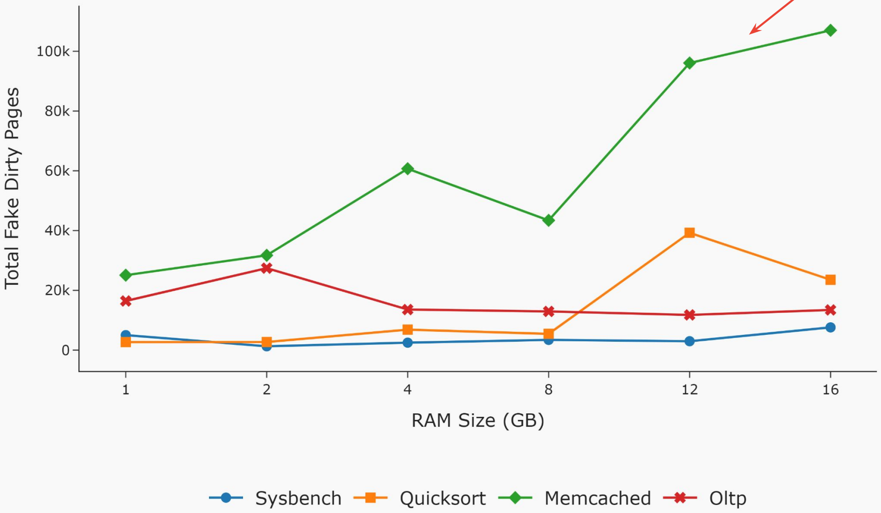
Analysis in the Next Slide »

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

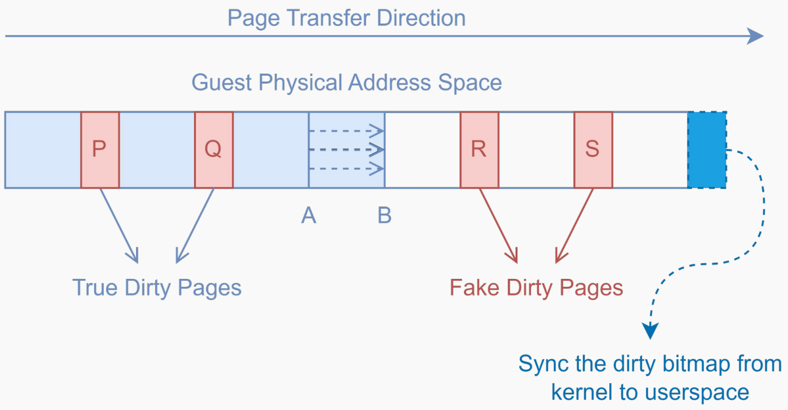
RAM (GB)	Fake Dirty Count	Transferred Page Count
1	33603 (9%)	380511
2	46093 (8%)	607484
4	79643 (7%)	1211109
8	58229 (3%)	2248032
12	113848 (3%)	3458888
16	138258 (3%)	4552333

08

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.


Page Transfer Direction
Guest Physical Address Space
P
Q
R
S
A
B
True Dirty Pages
Fake Dirty Pages
Sync the dirty bitmap from kernel to userspace

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

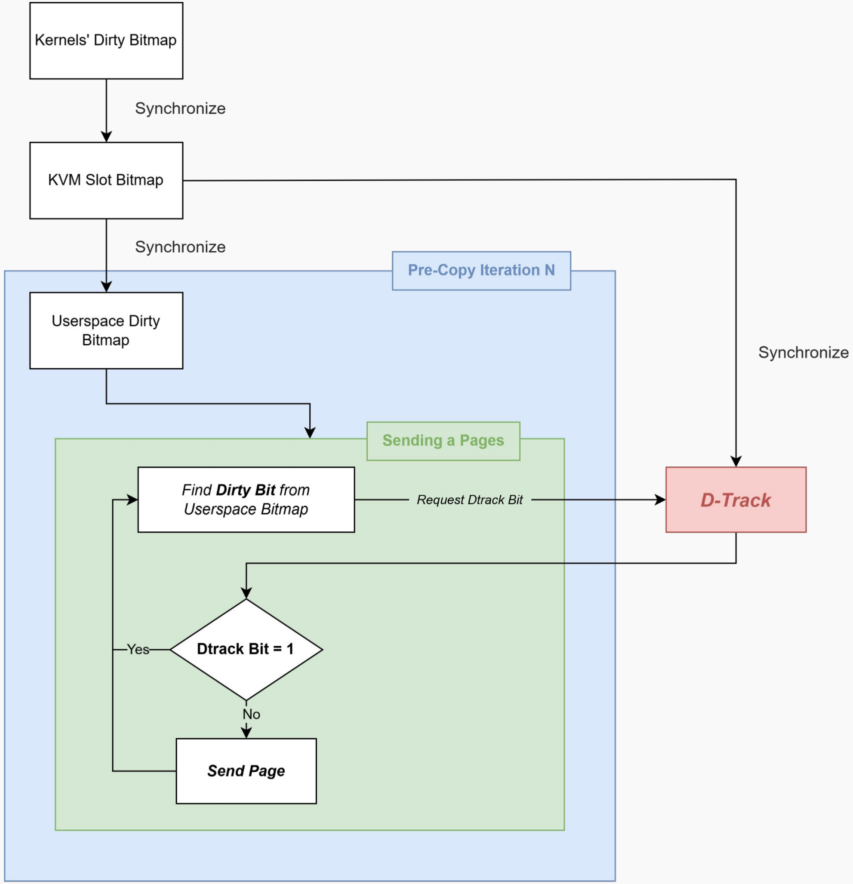
09

D-Track

D-Track

Design

- D-Track's tracking module is enabled, initializing a custom bitmap with all zeros.
- During iteration, D-Track continuously synchronizes with the kernel's dirty bitmap.
- When a page is selected for transfer, D-Track checks the corresponding bit in its bitmap. If that **bit is set to 1**, this page is a **fake dirty page** and will not be transferred. Else if the **bit is 0**, the page is **valid for transfer.**



10

Implementatio
n

Implementation

D-Track is implemented as two components:

1. Dirty Page Tracking
2. Selecting pages to transfer During Migration

Dirty Page Tracking

D-Track introduces a **dedicated tracking thread**, separate from the **main migration thread**, to monitor page dirtiness in parallel.

1. At the start of migration, the D-Track thread begins **tracking dirty pages concurrently**. D-Track maintains its own dirty bitmap, known as the **dtrack bmap**, which is periodically updated by synchronizing with the kernel's dirty bitmap via the **ioctl()** system call.
2. This tracking continues throughout each migration iteration and is paused immediately after the last page is transferred.
3. The synchronization process is performed in all iterations except the final one, where dirty tracking is disabled to prevent unnecessary operations.

Implementation

Selecting pages to transfer During Migration

The decision logic, works as follows:

1. If the corresponding bit in **dtrack bmap is set to 1**, the page was modified before being transferred. Since it will **appear as dirty in the next iteration**, it is skipped to avoid redundant transmission.
2. If the **bit is 0,** the page has not been modified pre-transfer and is therefore **safe to send** in the current iteration.

11

Evaluation

Evaluation

Evaluation Metrics

The performance of D-Track is measured using the following metrics:

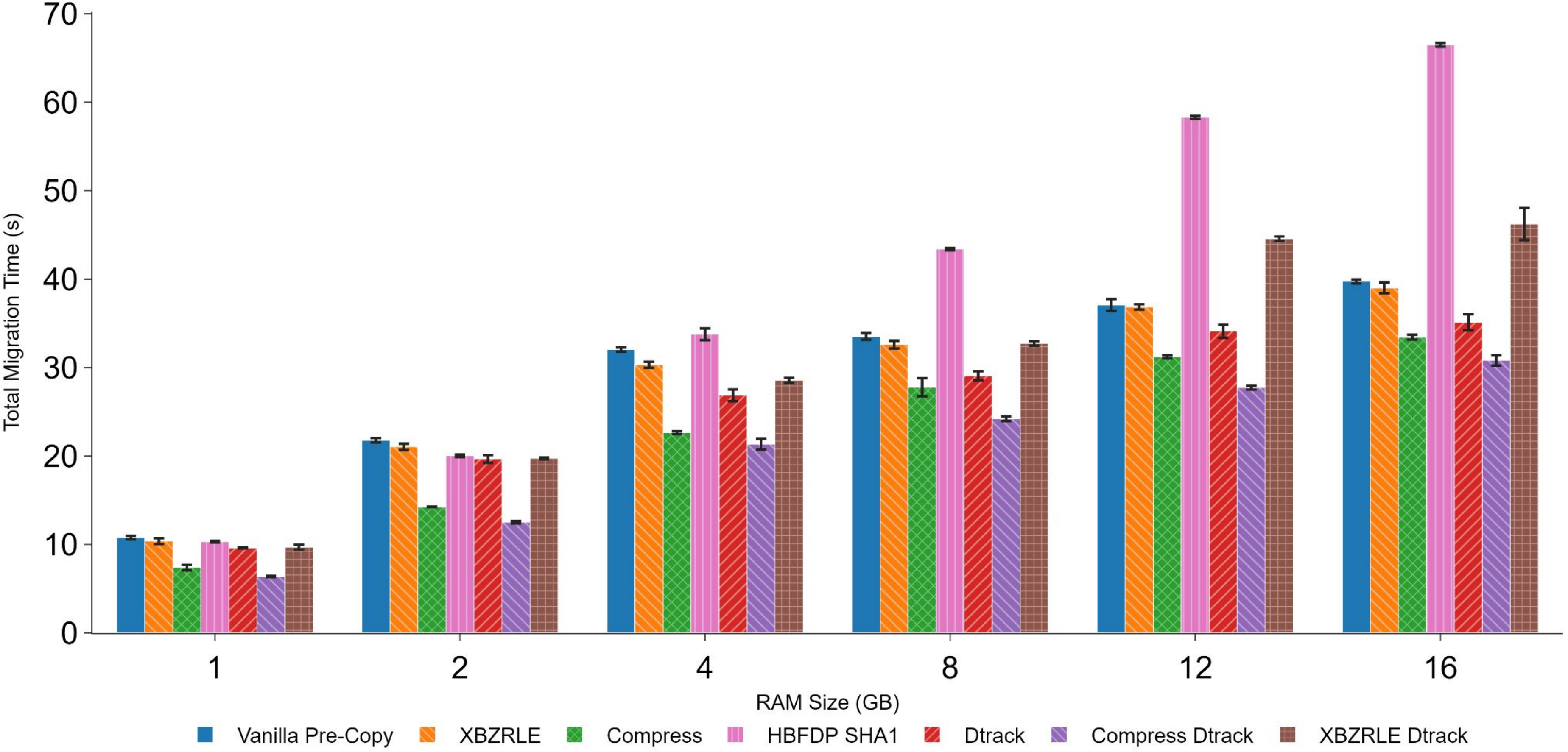
- **Total Migration Time:** Total time to migrate a VM from the source host to the destination host.
- **Downtime:** Time Duration during which the VM is suspended to transfer CPU/IO states.
- **Application performance degradation:** Any adverse impact on application performance during migration.
- **Network Bandwidth degradation:** Reduction in network bandwidth during migration.

Evaluation

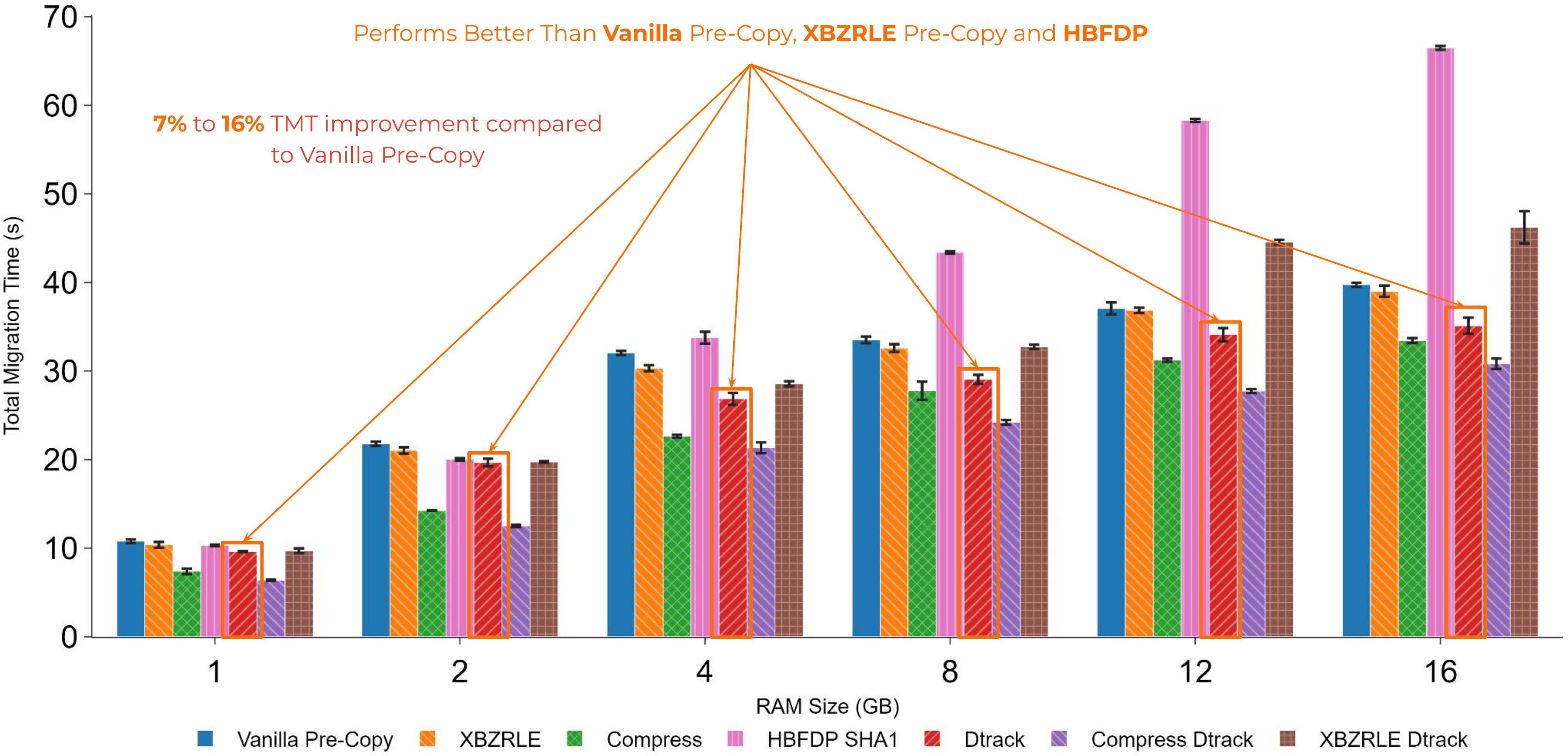
Baseline Techniques

Technique	Description
Vanilla Pre-copy	Default pre-copy migration that comes bundled with QEMU/KVM
XBZRLE	A run length encoding technique primarily used in QEMU/KVM to compress and transfer only dirtied content rather than transferring whole page
Compression	Compression technique bundled with QEMU/KVM which uses multiple threads to compress pages
HBFDP	Algorithm proposed by Li et al. [14]

Effect on Memory-Intensive Workload



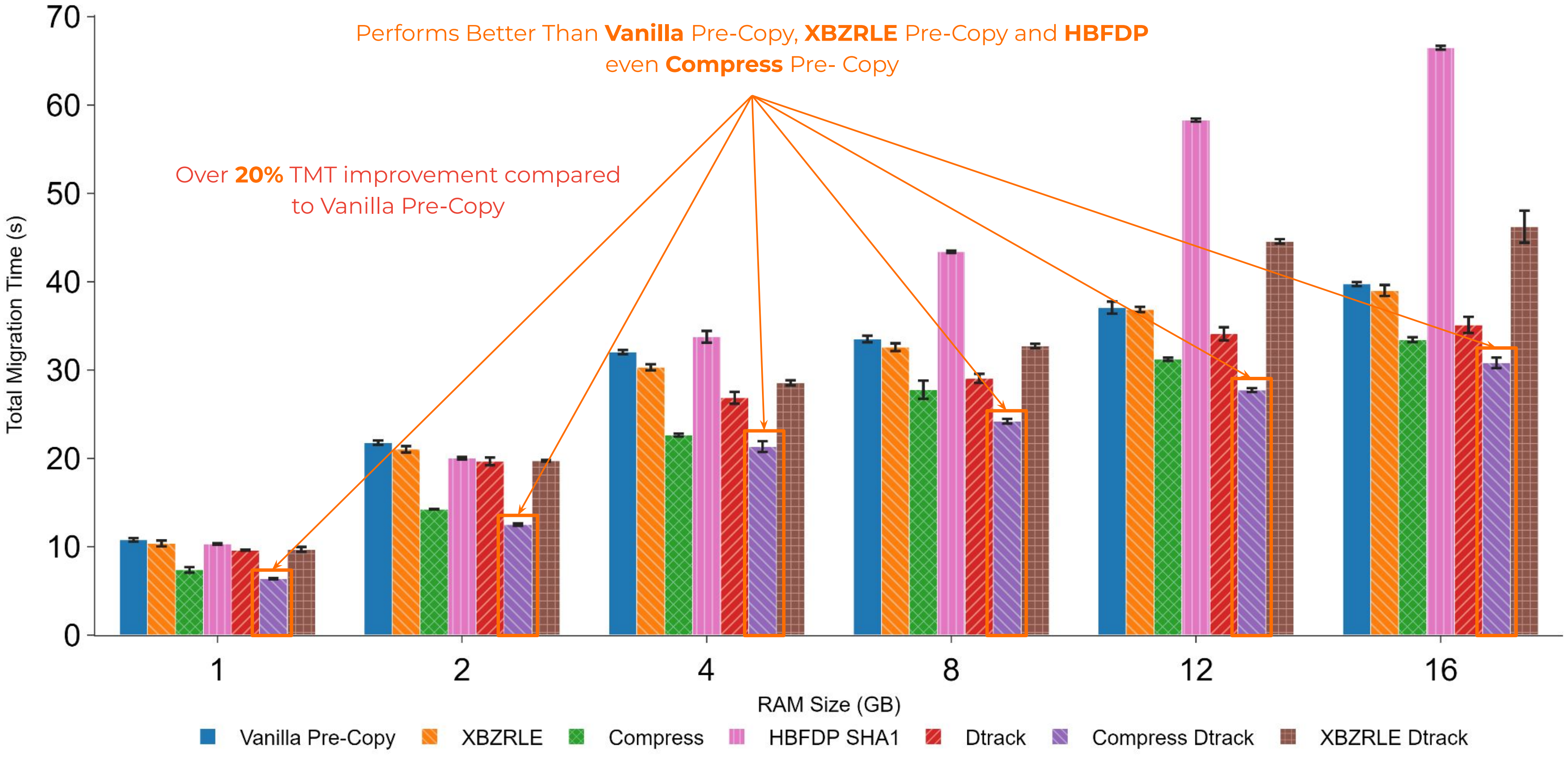
Effect on Memory-Intensive Workload



Performs Better Than **Vanilla** Pre-Copy, **XBZRLE** Pre-Copy and **HBFDP**

7% to **16%** TMT improvement compared to Vanilla Pre-Copy

Effect on Memory-Intensive Workload

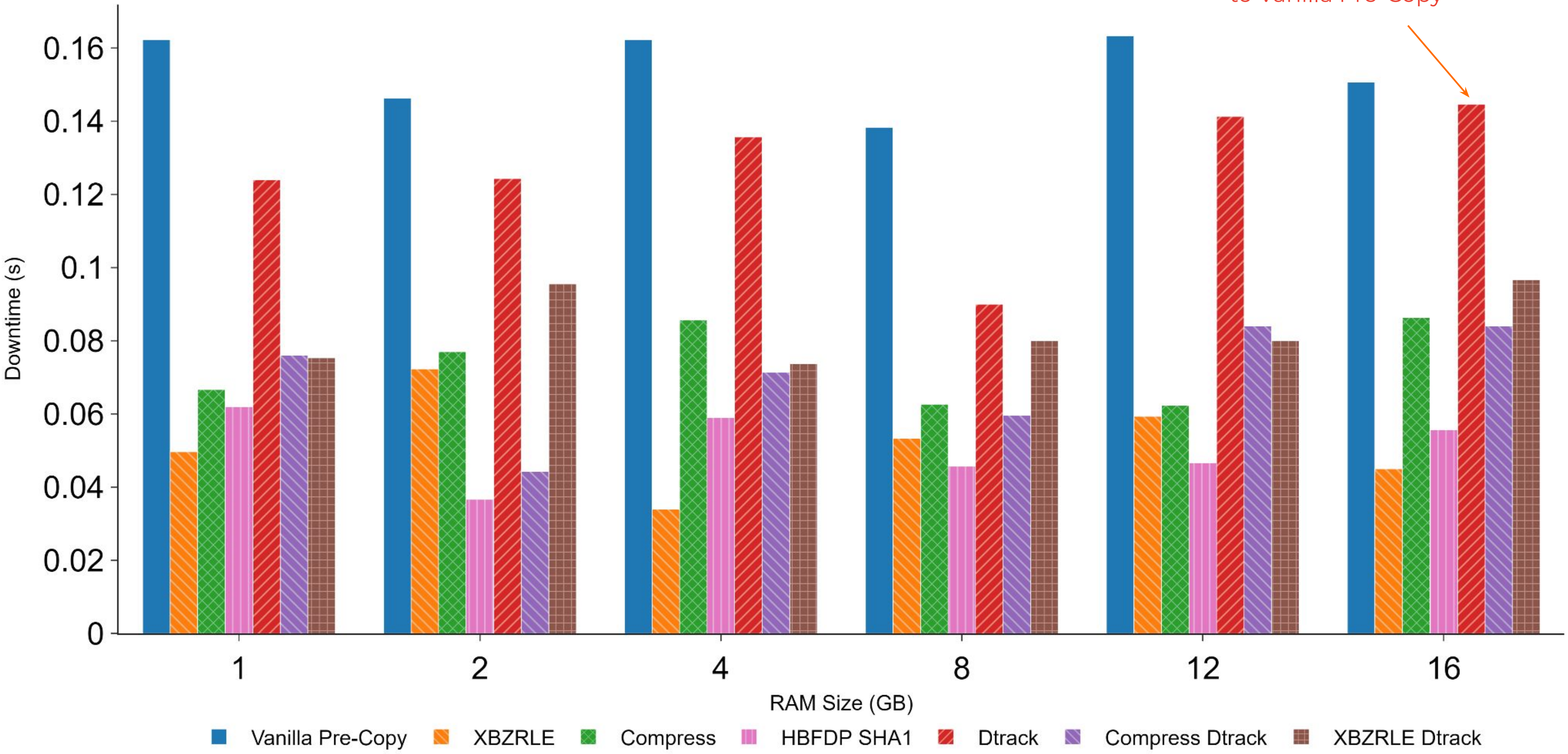


Performs Better Than **Vanilla** Pre-Copy, **XBZRLE** Pre-Copy and **HBFDP** even **Compress** Pre- Copy

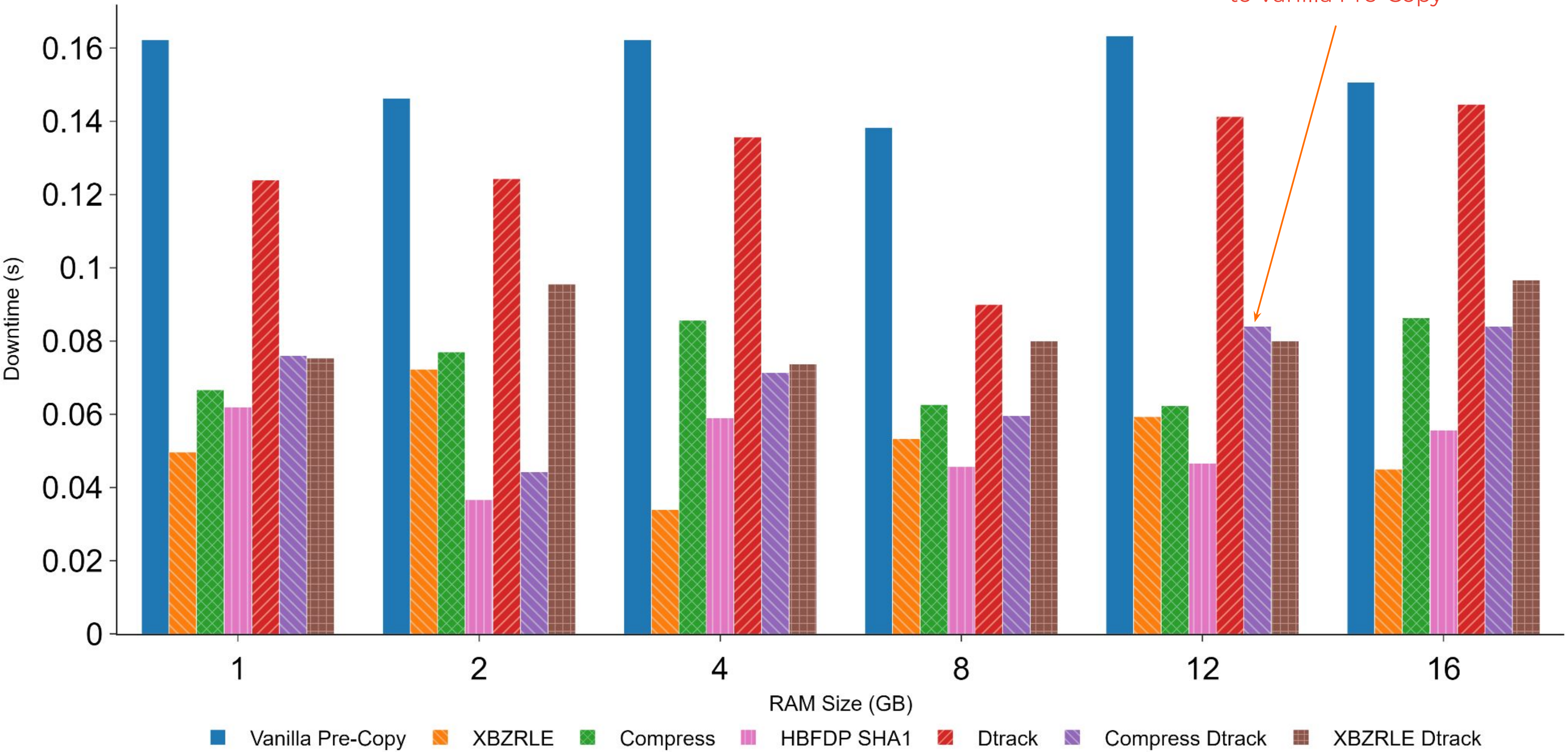
Over **20%** TMT improvement compared to Vanilla Pre-Copy

Effect on Memory-Intensive Workload

4% to **75%** DT improvement compared to Vanilla Pre-Copy



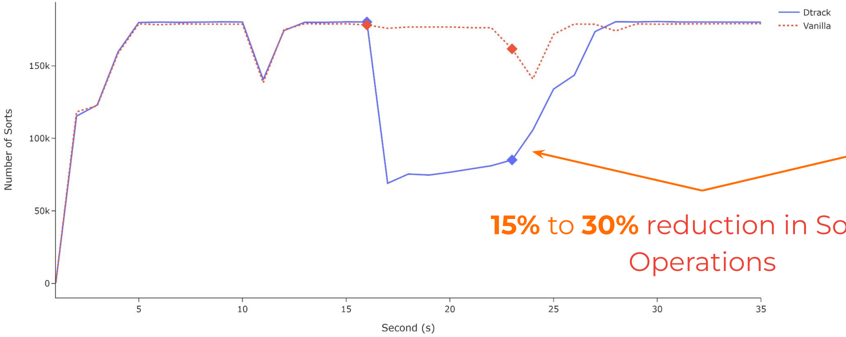
Effect on Memory-Intensive Workload



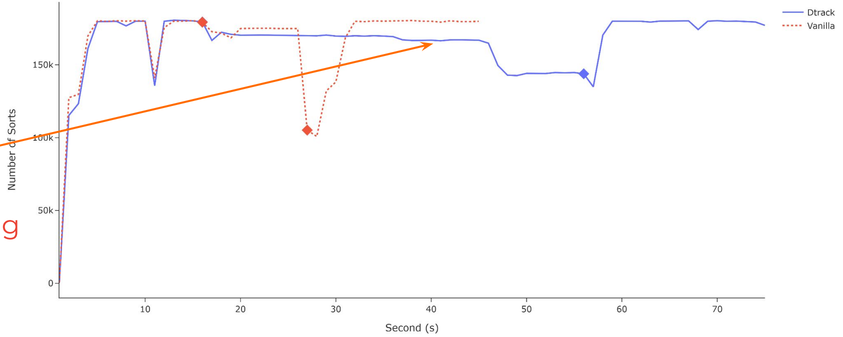
Over **45%** DT improvement compared to Vanilla Pre-Copy

Impact on CPU-intensive Workload

(a) D-Track - RAM 1 GB



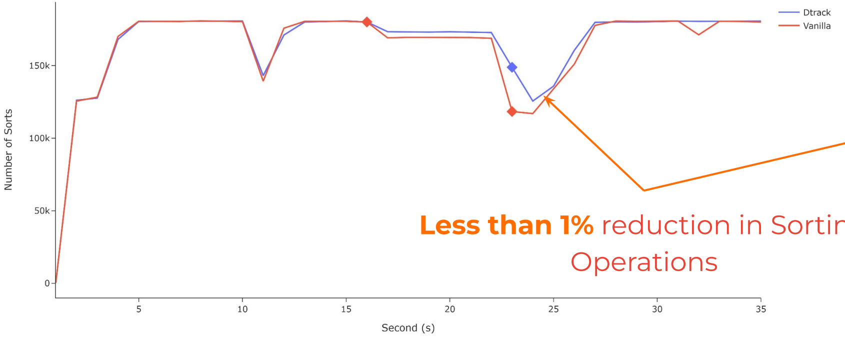
(b) D-Track - RAM 16 GB



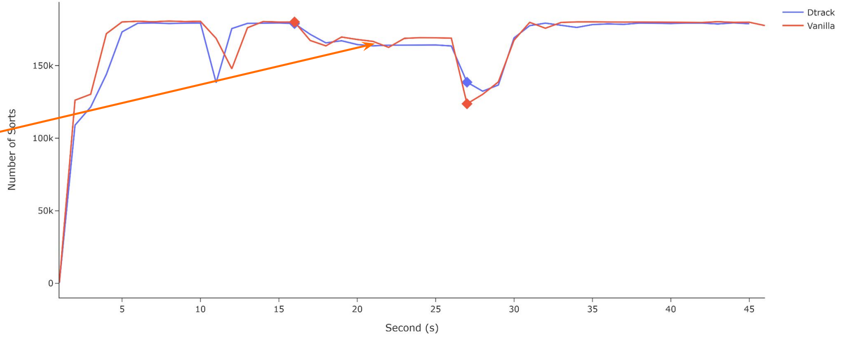
15% to **30%** reduction in Sorting Operations

The impact on CPU-intensive Quicksort benchmark is represented by figures **(a)** and **(b)** for vanilla pre-copy (*Red Dotted Line*) and D-Track (*Blue Line*) and figures **(c)** and **(d)** for vanilla pre-copy (*Red Dotted Line*) and **enhanced** D-Track (*Blue Line*)

(c) Enhanced D-Track - RAM 1 GB



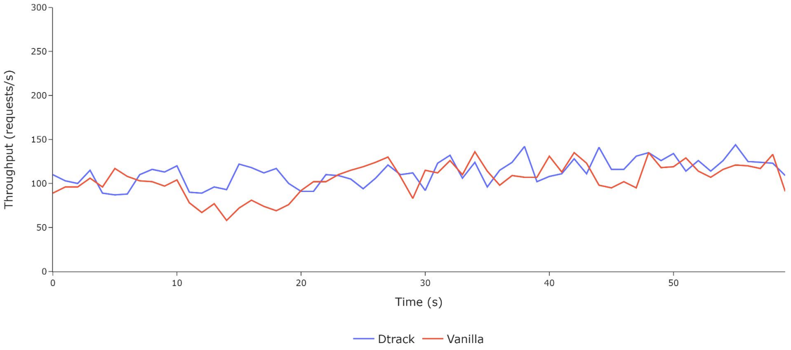
(d) Enhanced D-Track - RAM 16 GB



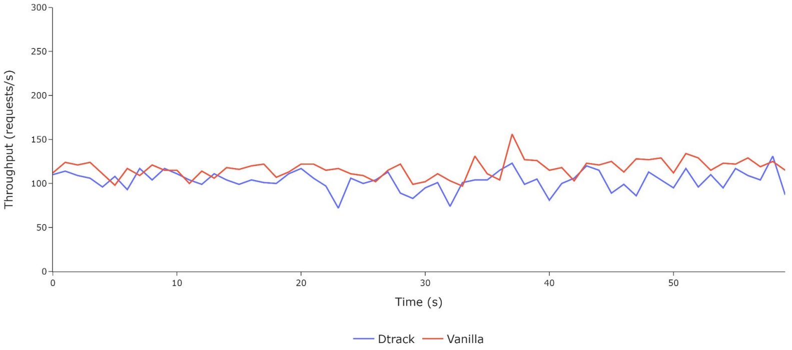
Less than 1% reduction in Sorting Operations

Throughput and latency of Mix-intensive YCSB benchmark

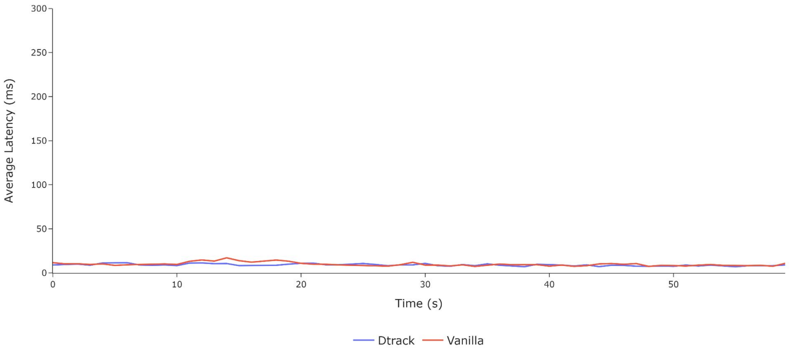
Vanilla pre-copy (*Red Dotted Line*) and **enhanced** D-Track (*Blue Line*)

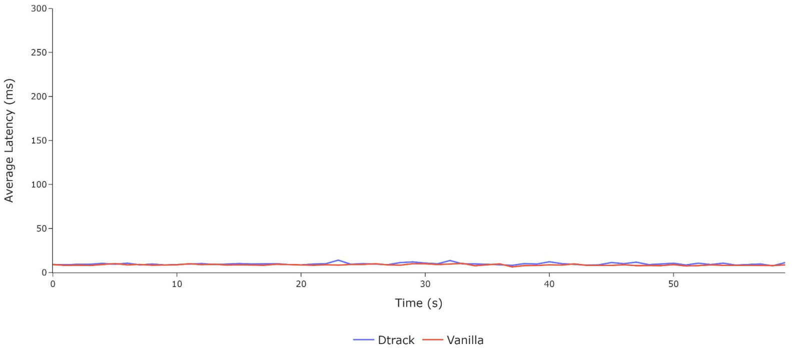


(a) RAM 1 GB



(b) RAM 16 GB





12

Addressing Research Questions

RQ1: How to mitigate redundant data transfers by identifying fake dirty pages with minimal overhead while solving the limitations of QEMU's existing dirty page tracking mechanism.

- The proposed D-Track mechanism introduces an efficient bitmap-based tracking mechanism that identifies fake dirty pages.
- D-Track avoids fake dirty page transfers **without introducing excessive computational overhead**.
- **Key findings** are as follows:
    - HBFDP, despite its theoretical benefits, **introduced significant hashing overhead**, increasing migration time compared to Vanilla Pre-Copy.
    - D-Track, in contrast, achieved up to **40% reduction** in total migration time for memory-intensive workloads by eliminating fake dirty pages without relying on costly hashing.
    - Also, the performance degradation of D-Track on CPU-intensive workloads was **less than 32%**, which is considerable when compared with the Vanilla Pre-Copy.
    - This performance degradation was **reduced to 1%** by modifying the D-Track's bitmaps synchronization by adding a delay between ioctl() calls (**enhanced D-Track**).
- This research demonstrates that bitmap-based tracking is more efficient than hash-based approaches for fake dirty page detection, as it avoids computational overhead while still mitigating redundant transmissions.

RQ2: How to reduce the data transferring load in Pre-Copy Migration to improve the performance of memory-intensive workloads

- Since D-Track **reduces the number of pages transferred**, applying compression on the remaining data further improves efficiency
- This hybrid approach achieved the best over all performance, outperforming standalone compression or XBZRLE.
- **Key findings** are as follows:
  - D-Track combined with Compression reduced migration time by **20%-40%** compared to Vanilla Pre-Copy for memory-heavy workloads like Memcached.
  - D-Track combined with XBZRLE was less effective than D-Track combined with compression, particularly in larger VMs, due to its higher CPU overhead.
  - D-Track alone still **outperformed HBFDP and Vanilla Pre-Copy**, proving that fake dirty page elimination is a fundamental optimization before applying further data reduction techniques.
- These results highlight that combining optimization with D-Track improves the live VM migration by reducing fake dirty pages and reducing the transferring load during the migration process for memory-intensive workloads.

13

Research Publication

Research Publication

As a result we were able to submit this research finding as a journal article to **IEEE Transactions on Computers** and its currently under review

- **S. Cooray**, and D. Fernando, “D-Track: Eliminating Fake Dirty Pages in Live Virtual Machine Migration,” **IEEE Transactions on Computers**, 2025. (Submitted - under review)

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.