SmartNIC-Enabled Live Migration for Storage-Optimized VMs

Jiechen Zhao[♥♣], Ran Shu[♣], Lei Qu[♣], Ziyue Yang[♣],
Natalie Enright Jerger[♥], Derek Chiou[♠◇], Peng Cheng[♣], Yongqiang Xiong[♣]
[♥] University of Toronto, [♣] Microsoft Research, [♠] University of Texas at Austin, [◇] Microsoft

Abstract

Cloud providers offer storage-optimized VMs equipped with locally attached storage to meet the high performance requirements of cloud users. However, current cloud providers cannot enable live migration for storage-optimized VMs due to the high resource overheads. Moreover, resources should be permanently provisioned for live migration as on-demand provisioning needs to de-allocate resources from either VMs or the hypervisor, thus violating SLA. We propose a storage live migration acceleration system on SmartNICs. Our design achieves minimal resource overhead and SLA violations by proposing (1) a SmartNIC-managed live migration architecture and (2) an efficient consistency algorithm. We implement a basic prototype on an FPGA-based SmartNIC. Preliminary results show that we can migrate storage-optimized VMs with no host resource usage and minimal performance interference to RocksDB running inside the VM. This project is part of the Terminus Project [28].

CCS Concepts

• **Computer systems organization** → **Cloud computing**; **Maintainability and maintenance**; • **Networks** → **Programmable networks**; • **Hardware** → **Networking hardware**; • **Software and its engineering** → **Virtual machines**; **Operating systems**.

Keywords

live migration, NVMe storage, cloud computing, hypervisors, virtual machines, SmartNICs, FPGAs

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
APSys '24, September 4–5, 2024, Kyoto, Japan
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1105-3/24/09
https://doi.org/10.1145/3678015.3680487

ACM Reference Format:
Jiechen Zhao, Ran Shu, Lei Qu, Ziyue Yang, Natalie Enright Jerger, Derek Chiou, Peng Cheng, Yongqiang Xiong. 2024. SmartNIC-Enabled Live Migration for Storage-Optimized VMs. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '24), September 4–5, 2024, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3678015.3680487

1 Introduction

Large-scale online services are usually deployed in clouds [31]. Many of these services rely highly on the performance of backend storage [13, 32]. Service owners usually have specific performance requirements when building their own storage applications. To meet those requirements, cloud providers offer storage-optimized VMs [21, 22]. These VMs have directly mapped local NVMe storage whose overall performance can be up to several millions of IOPS on a single VM. Such local storage is ephemeral (i.e., not persistent); data does not survive a VM stop or termination.

Meanwhile, live migration is a key technique for cloud providers to offer minimal service interruptions during management tasks including planned maintenance, system updates, configuration changes, hardware failure handling, and allocation optimizations [16, 17]. Although the total live migration time can be up to hours, tenant VMs only get paused temporarily at second or sub-second levels [49]. This benefit significantly alleviates minute-level service interruptions, compared to cases without live migration.

However, live migration is currently not available for storage-optimized instances in clouds [16, 22]. The main reason is the unaffordably high overhead alongside co-running tenant VMs. To provide live migration capability, the VM hypervisor has to track every disk access to guarantee consistency between the migration source and destination. Therefore, a virtualization layer is mandatory because it mediates local NVMe I/O and consistency tracking operations on both sides of a migration process. Unfortunately, bringing in such an additional layer is against the design principle of storage-optimized VMs that have directly mapped local NVMe storage. Worse, achieving such virtualization would take too much host CPU computation power [37]. While statically reserving CPU resources for live migration is not efficient, there is no on-demand way for providers to initialize and

manage live migration processes. In the worst case, if all cores are sold to tenant VMs, there are no compute resources left on the CPU for the provider to conduct live migration. Finally, the live migration will contend for host resources such as LLC, memory bandwidth, and PCIe bandwidth with tenant VMs. Due to the extremely high bandwidth of modern NVMe SSD cloud used, such contention has a high potential to degrade tenants' performance stability.

To this end, we propose to use FPGA-based SmartNICs[1] to accelerate the live migration of local storage for storage-optimized VMs. There are two reasons for doing so. (1) The SmartNIC is the perfect location to remove host involvement during live migration. This eliminates host-related performance interference e.g., LLC and memory bandwidth. It also reduces PCIe transactions needed in the existing approach due to host involvement. (2) Merely transferring live migration from the host CPU to SoC-based SmartNICs has performance limitations due to the wimpy SoC cores [1, 9, 18]. ASIC-based SmartNICs can only support limited states, making it hard to support storage migration with rich states [12, 30]. In contrast, FPGA-based SmartNICs can be tailored for live migration with greater cost efficiency and scalability.

Similar to host-managed live migration, there should be compute and memory resources on the SmartNIC provisioned for live migration. The key challenges to realizing our idea of SmartNIC-enabled live migration are (1) how to minimize SmartNIC resource overheads and (2) how to guarantee data consistency with high efficiency.

In this paper, we present an efficient local storage live migration acceleration system for storage-optimized VMs. We adopt the following key design principles to solve the above challenges. First, during live migration, we enable a hardware module that interposes all commands from the tenant VM and handles the data movement. We employ a one-to-one queue mapping scheme for both VM→SmartNIC and SmartNIC→disk communications. Second, we propose an efficient data consistency algorithm. We use double writes to both the migration source and destination to keep copied data up to date on both the migration source and the destination. For write commands targeting the in-flight copy range, our approach employs a low-overhead back-pressure scheme. This reduces performance interference on tenant VMs and SmartNIC resource overheads allocated for live migration.

We implement our prototype on an FPGA-based SmartNIC. Preliminary results show that we can migrate a VM running RocksDB on high-speed local SSDs without any host resource overhead and with only 12% extra latency to RocksDB.

[1]Data Processing Units (DPUs) and Infrastructure Processing Units (IPUs) are terms describing SmartNICs with more offloading features. Here we use SmartNICs as a general term including SmartNICs, DPUs, and IPUs.

2 Background and Motivation

This section gives the basics of block storage in the cloud and storage live migration (Sec. 2.1), and our motivation for using an FPGA-based SmartNIC to enable live migration of storage-optimized VMs (Sec. 2.2).

2.1 Block Storage in the Cloud and Storage Live Migration

There are three main types of block storage available for VMs in the cloud. We introduce their concepts according to Fig. 1. Note that live migration resides in the hypervisor in Fig. 1(a) and Fig. 1(c), but does not exist in Fig. 1(b) due to the direct mapping.

Persistent block storage. Persistent storage systems store data in remote storage clusters and emulate a block interface at the host side for VMs. Its system architecture is shown in Fig. 1(a). It provides high durability and availability for customers. Examples of persistent storage are AWS EBS [15], Azure Managed Disks [14], and Google Persistent Disk [20]. Live migration of VMs with persistent storage just needs to move memory states and reconnect the destination node with the remote storage, which is relatively simple and fast [49]. Although persistent storage provides many benefits, it cannot achieve high performance due to complicated software protocols (e.g., replication, data consistency, network congestion). Thus, cloud providers also offer two *local storage* options, which are introduced next.

Directly mapped local storage. Cloud providers also offer directly mapped NVMe disks (both SSDs and HDDs) to VMs to provide TB-scale capacity and several millions of IOPS on a single VM. This type of VM is called a storage-optimized VM [21, 22]. Fig. 1(b) illustrates its system architecture. These local disks are not managed by providers' hypervisor, but by users themselves, and the entire disk(s) is directly mapped to those VMs and not shareable with other tenants.[2] In current cloud systems, there is no support for live migration on directly mapped local disks from the cloud provider side. Moreover, cloud providers are unaware of tenants' accesses to the directly mapped local storage, thus unable to guarantee data consistency on their side. Such limitations can only be eliminated if the disk supports features required for live migration including an additional interface and user access tracking. However, none of the existing disks on the market support such features.

Hypervisor managed local storage. To acquire controlability of local storage, providers use hypervisors to manage local storage, as Fig. 1(c) shows. Architecturally, the hypervisor's management can run on the host CPU [3–5, 25], or

[2]The "entire disk" can be a logical one. Multiple logical disks can be located on the same physical device, but their access channels are isolated.

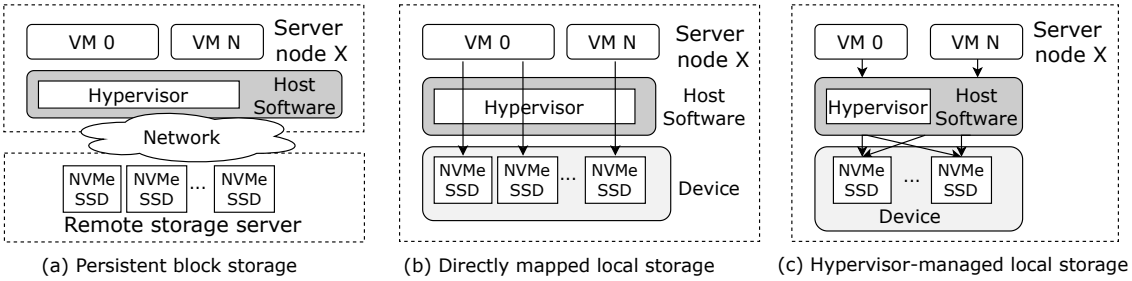

Figure 1: Different storage systems in the cloud.

be offloaded onto the SmartNIC [1, 37, 38]. Live migration is supported in the hypervisor software [3–5, 7, 25]. The hypervisor-level management usually introduces a certain amount of overhead, which scales as overall I/O activity increases.

A walk-through example on storage live migration. Since data is non-persistent and temporary in local disks, migrating data is necessary when migrating storage-optimized VMs. We walk through the live migration process for the local virtual disk of a VM. Initially, the hypervisor starts a background data copy from the migration source to the destination. This phase is called the pre-copy phase. As data on the source disk can be modified by the VM, consistency tracking methods like dirty block tracking or IO mirroring should be adopted [44]. At a point when the source and destination are nearly converged or as per the cloud provider's decisions, the second phase starts, where the VM is stopped, copied, and restored to the new location. This stage is called stop-and-copy. To reduce the stop-and-copy time, post-copy is an optimization which does not finalize the source and destination convergence, but letting the VM keep fetching updated data from the source after moving to the destination [23]. The post-copy approach is usually complex and costly [44], e.g., resources on the source cannot be released until the two sides fully converged. Major local storage live migration systems use pre-copy plus stop-and-copy [52].

2.2 Motivation

To enable live migration (LM) for storage-optimized VMs, the naive solution is for cloud providers to change the local storage type from directly mapped to hypervisor managed. Although this solution brings LM manageability for storage-optimized VMs, it is so costly that cloud providers cannot afford it. Next, we describe several drawbacks of the current LM approach and summarize the reasons for its inefficacy in managing storage-optimized VMs.

Extra software overheads. Efficiency-wise, there are three sources of overheads that commonly exist across current hypervisors [3–5, 7, 25] that employ software-based LM. First, between 22% and 39% of active CPU cycles are used to poll and trigger guest interrupts [37]. Second, long latency nested page walks in the host memory significantly affect performance [43]. Third, events such as hypervisor trapping incur CPU mode switches and cache pollution [24, 36]. With the above software overheads accumulated, the resultant host overhead of copying data between two virtual NVMe SSDs at line rate is almost 6 Xeon cores [37].

For single-disk VMs, these observed software overheads make migration efficiency scale poorly with increasing disk speeds, either at the cost of more host cores or suffering from slower LM. Largest storaged optimized VMs can hold up to 10 high-speed SSDs [22]. LM should migrate TB-scale disk data in one process. Such software overheads will slow the LM performance to a crawl, e.g., tens of hours. Concurrently paralleling the LM of different disks can accelerate this process [50], but it linearly increases the demand for CPU resources. Besides high CPU overheads, higher CPU occupancy also potentially leads to more host resource contention, which we discuss next.

Resource contention. Existing software approaches degrade the application's performance predictability. Our experiment migrates 64 GB data between two SSDs using a host-based KVM/QEMU implementation. We co-locate the LM process with a CPU-intensive, latency-critical key-value store application, MICA [40]. We initialize a MICA instance with 28 threads, 50%/50% GET/SET, and leave the remaining 4 threads for KVM (hyperthreading is off). The speed of LM itself fluctuates by 37-80% of the line rate and ultimately slows down by 40%. The tenant VM suffers up to 13× SET latency increase. In this case, the LM contends for host memory with MICA due to its data copies and NVMe I/O queue operations, which takes 33 GB/s memory bandwidth for 120 MOPs throughput. In addition, the miss rate of LLC increases due to co-located LM.

FPGA-based SmartNIC enabled storage live migration. In this work, we choose to offload the LM feature to SmartNICs widely deployed in a variety of public clouds [1, 11,

19, 33, 45]. Because storage LM, by its nature, is moving data between disks through network, we have new opportunities to take advantage of the efficiency benefits from SmartNICs [33, 42, 46].

SoC-based SmartNICs have limited computational capability with wimpy cores, thus they cannot feasibly handle the offloading of storage live migration for storage-optimized VMs [37]. Therefore, we choose to use FPGA-based SmartNICs. We believe this is a viable and practical design choice, taking into account (1) high software overheads, (2) resource contention, and (3) the high I/O speeds of storage-optimized VMs. FPGA-based SmartNICs have been successfully deployed to accelerate data plane operations for virtual networks [33]. The solution significantly saves CPU cycles, improves efficiency, and offers near-native stable performance.

3 System Design

As an FPGA-based SmartNIC is customizable by the provider for critical features such as live migration (LM), it gives us a new design space to make effective use of SmartNICs by leveraging the following two design principles.

Efficient live migration architecture. We adopt a similar access flow like that in FVM [37], i.e., only interposing the command path while keeping the direct data path between disk and VM memory. For command path, we use one-to-one mappings between VM-SmartNIC I/O queues and SmartNIC-disk I/O queues. This design not only eliminates the need of queue multiplexing/demultiplexing on the SmartNIC, but also it keeps users' multi-queue scheduling unchanged.

On-SmartNIC consistency management between VM and LM access. Although the SmartNIC is the right location to remove host resource contention caused by LM processes (Sec. 2.2), synchronization mechanism is needed on the FPGA-based SmartNIC. We propose efficient lock-based synchronization mechanisms purely running on the FPGA. A double write algorithm preserves atomicity and reduces the hardware complexity of managing concurrent in-flight commands from the LM process and the VM being migrated.

3.1 Efficient Live Migration Architecture

The SmartNIC and NVMe disks are both connected to the PCIe bus. Fig. 2 provides a system overview of our on-SmartNIC LM approach. The LM module on the SmartNIC is responsible for triggering, monitoring, and managing an LM process on behalf of the software hypervisor running on the host. There are four possible paths handled by the LM module. For VM access path, we adopt the access flow introduced by FVM [37] between VMs and NVMe disks, where each NVMe command goes through the SmartNIC. Such design eliminates host involvements and unnecessary PCIe transactions. The SmartNIC on the migration source node should handle

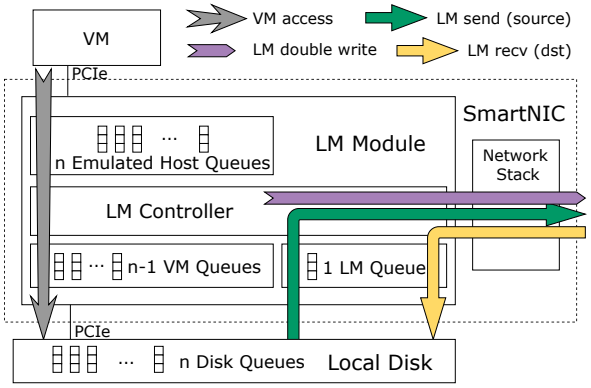

Figure 2: Overview of the proposed system with SmartNIC-enabled live migration (LM), with four possible paths supported by our SmartNIC.

two LM-related path, i.e., LM double write and LM send. The SmartNIC on the destination should handle LM receive. We clarify that the four paths in Fig. 2 separately exist in either the source SmartNIC or the destination SmartNIC, but they do not simultaneously exist in one SmartNIC.

SmartNIC-disk direct access. The NVMe protocol relies on I/O queues to submit/complete commands to/from disk controllers. Our SmartNIC functions as an NVMe I/O hypervisor with such I/O queues, thus acquiring direct access from the SmartNIC to disks. Based on such direct accessibility, this hypervisor can talk to its local disks with two types of paths: an LM path and a user access path. The former is the path between LM NVMe I/O queues and the network stack. The latter is the path between user NVMe I/O queues and the host NVMe queues. Only one I/O queue is required by the LM path as the FPGA is able to achieve peak read/write performance thanks to its internal parallelism and specialized circuits. Other NVMe I/O queues are mapped to user paths in an one-to-one fashion. In other words, if a device supports n disk queues, the SmartNIC provisions one NVMe I/O queue dedicated to the LM, and other $n-1$ queues on the SmartNIC are mapped as users' NVMe queues. The queue mapping as described is shown in Fig. 2. All queues are stored in the on-SmartNIC SRAM. As a result, the disk controller reads or writes NVMe commands stored in the on-NIC memory instead of the host DRAM, reducing PCIe transactions and potential contention on the PCIe or memory bus.

Host-SmartNIC interface. Another $n-1$ host NVMe queues are used to emulate the NVMe frontend to VMs in a one-to-one mapping fashion. This eliminates the queue multiplexing and demultiplexing overhead in the FPGA logics. Such design also keeps the user I/O scheduling unchanged. Thus we avoid unexpected performance loss which affects user

satisfaction. The addresses of those I/O queues are mapped to the MMIO address space allocated by the NVMe driver on the host. The mappings are managed by the provider.

Live migration controller. The controller proactively generates LM-related send/recv commands and talks to its local disks through the direct access interface introduced above. During an LM process, the LM controller has full interposition over commands on both the LM path and user access path(s). Thus, the controller has full visibility to handle data consistency among all paths, as well as that between migration source and destination nodes. The detailed consistency algorithm in the controller is introduced in Sec. 3.2.

Network stack. On each SmartNIC, we use a network stack provide reliable communication with other SmartNICs on other hosts. There are multiple design options. Providers can choose to implement an I/O stack on the FPGA to drive the NIC using RDMA or TCP engines [26, 41, 54]. The NIC can be a stand-alone one or an FPGA-integrated one. Another choice is to adopt a pure FPGA-based network stack like LTL [27]. We choose LTL as a reliable network transport in our prototype.

Scalability. Our LM design is lightweight to be implemented. Our prototype for single disk LM only consumes ~3% of the FPGA logic on an FPGA-based SmartNIC using Intel Arria 10 [6]. One easy approach to migrate VM with multiple disks mapped is duplicating the LM module. There is plenty of room left to support the largest number of disks of a VM which is 24 [21]. Please note that Arria 10 is an old FPGA released in 2013, we believe the LM resource overhead should be much lower on the latest SmartNIC like latest version of Azure Boost [19]. Such overhead will not become the resource bottleneck of SmartNIC.

3.2 Live Migration Controller

The algorithm we use, "double writes", is one seen elsewhere in industry and academia [44, 49]. The algorithm locks a part of the physical address region that involves in-flight migration commands. If user write I/Os fall in the locked region, they are double written to both the source disk and the destination disk, until the writes are properly merged. Additionally, we add new design considerations to the double writes to fit our scenario and system.

Consistency algorithm with low hardware overhead. Since the consistency algorithm is executed by our LM controller instead of by host software, the complexity of the algorithm is resource-constrained and should be hardware-friendly. To achieve this goal, this work partitions the physical address space of the disk that is being migrated into three logical regions: the region that has finished being copied (`Migrated`), the region with in-flight copy requests

Algorithm 1 Proposed consistency algorithm working with SmartNIC-enabled storage live migration.

```
1:  ▷ At source node (all functions run on SmartNIC): ◁
2:  procedure SmartNIC-triggered Migrate
3:      ResetPointers()
4:      while LM_Tail < LEN do
5:          if LM_Head - LM_Tail + Inflight_User < TH then
6:              data = ReadDisk(LM_Head)
7:              SendCopyData(LM_Head, data, FLAG_COPY)
8:              LM_Head++
9:  function OnTenantWrite(address, data)
10:     if address < LM_Head then
11:         SendUserData(address, data, FLAG_USER)
12: function OnRecvAck(address, isCopy)
13:     if isCopy then
14:         LM_Tail++
15:     else
16:         Wait for ACKs from local writes for completion
17: ▷ At destination node: ◁
18: function OnRecvData(address, data, isCopy)
19:     WriteDisk(address, data)
20:     SendAck(address, isCopy)
```

(i.e., `InFlightMigrating`), and the region to be copied (i.e., `ToBeMigrated`). The regions are shown in Fig. 3. There are 4 hardware pointers to track the progress of the LM. `LM_RD_head` and `LM_WR_head` indicate the progress of LM (local) read and (remote) write commands submitted, while `LM_RD_tail` and `LM_WR_tail` indicate the progress of those commands committed. The consistency algorithm is described in Algorithm 1. We simplify the algorithm's presentation with only `LM_head` and `LM_tail`, while using the whole 4 pointers are good for better pipeline parallelism on the FPGA. All VM user I/Os to the local disk are omitted in the algorithm.

The LM controller would not give acknowledgement to the tenant until double writes are both committed. User commands do not need to be stored in the LM controller but stored in the VM queues. In the worst case, if users want to keep writing the address region being migrated, the host queues will become full, which in return back-pressures the VM to stop sending new user commands. This design results in a small buffer requirement on the FPGA to store in-flight commands.

Mitigate user performance interference. On the migration source, LM traffic may contend disk bandwidth with traffic from VMs. We use a simple design to mitigate user performance interference. The LM queue can only issue reads from the source disk if the number of users' in-flight requests `InflightUser` is bellow a threshold (`line5` in Algorithm 1). The threshold `TH` is chosen to be the number of in-flight

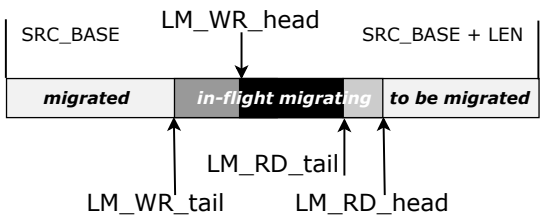

Figure 3: Snapshot of the address regions being migrated. LEN is the length of data region to be migrated.

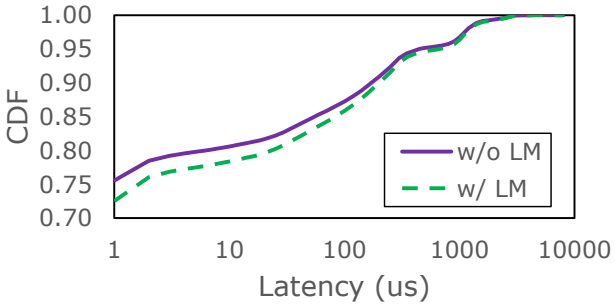

Figure 4: RocksDB latency CDF running with/without live migration (LM) on our prototype.

NVMe commands that populates a disk at its peak throughput. Cloud providers can easily get the right TH by profiling their disks. In essence, such design approximates a strict priority that favors VM traffic over LM traffic. On the destination node, the LM has no interference with other VMs as storage-optimized VMs typically use disk exclusively. The target disk is only provisioned for the VM being migrated.

4 Preliminary Evaluation

We implement a prototype on an FPGA-based SmartNIC using ~6,000 lines of System Verilog code. The implementation uses ~9,000 adaptive logic modules (ALMs) which takes less than 2% of our Intel Arria 10 FPGA logic resources.

Our prototype can achieve 2.25 GB/s LM performance – the peak performance of our SSDs if no other application is running. The SSD we use is a 280 GB Intel Optane 900P [2].

Furthermore, we conduct an experiment that showcases our prototype which migrates a VM running RocksDB [10] to another SSD. We use the RocksDB official benchmarking tool [8] to generate test traffic. We set the RocksDB key size to 30 bytes and the value size to 30,720 bytes. Such a large value size makes the disk load extremely heavy, acting as a worst-case evaluation. The RocksDB file size is 4.319 GiB. For simplicity, we only migrate 8 GiB of disk space, which contains the RocksDB file. We use `readwhilewriting` pattern and set the number of read commands to 1 million. Compression is disabled to push high disk workload and exhaustively compete for bandwidth with the LM job.

We find the LM lasts for 78.749 seconds with an average throughput of 109.8 MB/s. RocksDB uses the rest of the available disk bandwidth (about 2.1 GB/s). Fig. 4 shows the read latency distribution during the whole running time without or with the LM job co-located. The average latency increases from 83.57 μs to 93.55 μs. The 90 percentile latency increases from 171.71 μs to 200.99 μs. Therefore, our solution achieves efficient LM with only 12% average latency and 17% 90th percentile latency for this RocksDB setting.

5 Open Questions

Handling performance issues. Double writes performance depends on network status. Network congestion, as an example, can degrade user experience as users need to wait for acknowledgment from migration destination. Designing schemes that offer better user experience while keeping the LM module simple and effective is one of our future work.

Failure handling. As we introduce new components into the system, several potential failure cases need to be addressed. The network failures between migration source and destination is easy to handle as we can cancel the migration. The SmartNICs themselves can also fail. On one hand, the potential failures on the SmartNIC on the migration destination node can be solved in the same way as how we handle network failures. On the other hand, the potential SmartNIC failures on migration source node is trickier to handle. They can happen even if the LM is disabled and the SmartNIC only intercept user commands. A fallback mechanism to host software [33] is one promising approach and the detailed solution will be one of our future work.

Use SmartNIC to accelerate other resources' live migration. There are proposals to migrate other resources lively, such as migrations of memory [34, 35, 49], network states [29, 39, 47–49, 51, 53], and local storage [44, 49]. Most of them use the hypervisor or leverage the network switches to manage and redirect migrated resources. Besides, providers cannot migrate other types of resource lively, e.g., GPUs and TPUs instances [16]. It is possible to extend our agenda of SmartNIC-enabled live migration to these types of resources.

Acknowledgments

We would like to thank our anonymous reviewers for their valuable feedback. This work was supported in part by a Canada Research Chair and the University of Toronto McLean Award.

References

[1] [n. d.]. AWS Nitro System. https://aws.amazon.com/ec2/nitro/. Accessed: 2024-7-8.

[2] [n. d.]. Intel® Optane SSD 900P Series 280GB 12 Height PCIe x4 20nm 3D XPoint Product Specifications. https://ark.intel.com/content/www/us/en/ark/products/123628/intel-optane-ssd-900p-series-280gb-12-height-pcie-x4-20nm-3d-xpoint.html. Accessed: 2024-7-8.

[3] [n. d.]. Kernel-based Virtual Machine (KVM). http://www.linux-kvm.org. Accessed: 2024-7-8.

[4] [n. d.]. Microsoft Hyper-V. http://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx. Accessed: 2024-7-8.

[5] [n. d.]. VMware. http://www.vmware.com. Accessed: 2024-7-8.

[6] 2013. Intel®FPGAs - Intel®Arria®10 GX FPGA. https://www.intel.com/content/www/us/en/products/details/fpga/arria/10/gx.html. Accessed: 2024-7-8.

[7] 2018. QEMU: the FAST! processor emulator. https://www.qemu.org/. Accessed: 2024-7-8.

[8] 2022. Benchmarking tools · facebook/rocksdb Wiki · GitHub. https://github.com/facebook/rocksdb/wiki/Benchmarking-tools. Accessed: 2024-7-8.

[9] 2022. Broadcom Stingray PS1100R. https://docs.broadcom.com/doc/PS1100R-PB. Accessed: 2022-1-1.

[10] 2022. RocksDB: A Persistent Key-Value Store for Fast Storage Environments. https://rocksdb.org/. Accessed: 2024-7-8.

[11] 2023. Alibaba CIPU. https://www.alibabacloud.com/blog/a-detailed-explanation-about-alibaba-cloud-cipu_599183. Accessed: 2024-7-8.

[12] 2024. Agilio CX SmartNICs. https://www.netronome.com/products/agilio-cx/. Accessed: 2024-7-8.

[13] 2024. Amazon DynamoDB. https://aws.amazon.com/dynamodb/. Accessed: 2024-7-8.

[14] 2024. Azure Disk Storage Overview - Azure Virtual Machines | Microsoft Learn. https://learn.microsoft.com/en-us/azure/virtual-machines/managed-disks-overview. Accessed: 2024-7-8.

[15] 2024. Cloud Block Storage - Amazon EBS - AWS. https://aws.amazon.com/ebs/. Accessed: 2024-7-8.

[16] 2024. Google Cloud - Live Migration Process during Maintenance Events. https://cloud.google.com/compute/docs/instances/live-migration-process. Accessed: 2024-7-8.

[17] 2024. Maintenance and Updates - Azure Virtual Machines. https://learn.microsoft.com/en-us/azure/virtual-machines/maintenance-and-updates#live-migration. Accessed: 2024-7-8.

[18] 2024. Marvell OCTEON 10 DPU. https://www.marvell.com/products/data-processing-units.html. Accessed: 2024-7-8.

[19] 2024. Overview of Azure Boost. https://learn.microsoft.com/en-us/azure/azure-boost/overview. Accessed: 2024-7-8.

[20] 2024. Persistent Disk: Durable Block Storage | Google Cloud. https://cloud.google.com/persistent-disk. Accessed: 2024-7-8.

[21] 2024. Storage Optimized Instances - Amazon EC2. https://docs.aws.amazon.com/ec2/latest/instancetypes/so.html. Accessed: 2024-7-8.

[22] 2024. Storage Optimized Virtual Machine Sizes - Azure Virtual Machines. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-storage. Accessed: 2024-7-8.

[23] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. 2011. Vmflock: Virtual Machine Co-migration for the Cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. 159–170.

[24] Nadav Amit, Muli Ben-Yehuda, Dan Tsafrir, and Assaf Schuster. 2011. vIOMMU: Efficient IOMMU Emulation. In *Proceedings of the 2011 USENIX Annual Technical Conference (ATC)*. 73–88.

[25] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*. 164–177.

[26] Junehyuk Boo, Yujin Chung, Eunjin Baek, Seongmin Na, Changsu Kim, and Jangwoo Kim. 2023. F4T: A Fast and Flexible FPGA-based Full-stack TCP Acceleration Framework. In *Proceedings of the 50th ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 1–13.

[27] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A Cloud-scale Acceleration Architecture. In *2016 49th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13.

[28] Derek Chiou, Ran Shu, Lei Qu, Peng Cheng, Yongqiang Xiong, Ram Huggahalli, Arun Kishan, Mark D. Hill, and Steve Scott. 2024. Terminus: Moving the Center of Cloud Servers from Cores to SmartNICs and Beyond. HPCA 2024 Keynote.

[29] Inho Choi, Nimish Wadekar, Raj Joshi, Joshua Fried, Dan R.K. Ports, Irene Zhang, and Jialin Li. 2023. Capybara: μSecond-Scale Live TCP Migration. In *Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*. 30–36.

[30] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. 2019. λ-NIC: Interactive Serverless Compute on Programmable SmartNICs. *arXiv preprint arXiv:1909.11958* (2019).

[31] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (2013), 74–80.

[32] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: Evolution of Development Priorities in a Key-value Store Serving Large-scale Applications. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 1–32.

[33] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 51–66.

[34] Michael R Hines and Kartik Gopalan. 2009. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 51–60.

[35] Chinmay Kulkarni, Aniraj Kesavan, Tian Zhang, Robert Ricci, and Ryan Stutsman. 2017. Rocksteady: Fast Migration for Low-latency In-memory Storage. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*. 390–405.

[36] Yossi Kuperman, Eyal Moscovici, Joel Nider, Razya Ladelsky, Abel Gordon, and Dan Tsafrir. 2016. Paravirtual Remote I/O. In *Proceedings of the 21st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 49–65.

[37] Dongup Kwon, Junehyuk Boo, Dongryeong Kim, and Jangwoo Kim. 2020. FVM: FPGA-assisted Virtual Device Emulation for Fast, Scalable, and Flexible Storage Virtualization. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 955–971.

[38] Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R.K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and Anirudh Badam. 2020. Leapio: Efficient and Portable Virtual NVMe Storage on ARM SoCs. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 591–605.

[39] Xiaoyu Li, Ran Shu, Yongqiang Xiong, and Fengyuan Ren. 2024. Software-based Live Migration for Containerized RDMA. In *Proceedings of the 8th ACM SIGCOMM Asia-Pacific Workshop on Networking (APNet)*. 52–58.

[40] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. 2014. MICA: A Holistic Approach to Fast In-Memory Key-Value Storage. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 429–444.

[41] Katie Lim, Matthew Giordano, Theano Stavrinos, Baris Kasikci, and Thomas Anderson. 2024. Beehive: A Flexible Network Stack for Direct-Attached Accelerators. *arXiv preprint arXiv:2403.14770* (2024).

[42] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3: Energy-efficient Microservices on SmartNIC-accelerated Servers. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*. 363–378.

[43] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion, and Boris Grot. 2019. Prefetched Address Translation. In *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1023–1036.

[44] Ali José Mashtizadeh, Emré Celebi, Tal Garfinkel, and Min Cai. 2011. The Design and Evolution of Live Storage Migration in VMware ESX. In *Proceedings of the 2011 USENIX Annual Technical Conference (ATC)*. 187–200.

[45] Nirav Mehta. 2022. Introducing C3 machines with Google's custom Intel IPU | Google Cloud Blog. https://cloud.google.com/blog/products/compute/introducing-c3-machines-with-googles-custom-intel-ipu. Accessed: 2024-7-8.

[46] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. 2021. Gimbal: Enabling Multi-tenant Storage Disaggregation on SmartNIC JBOFs. In *Proceedings of the 2021 ACM SIGCOMM Conference*. 106–122.

[47] Maksym Planeta, Jan Bierbaum, Leo Sahaya Daphne Antony, Torsten Hoefler, and Hermann Härtig. 2021. MigrOS: Transparent Live-Migration Support for Containerised RDMA Applications. In *Proceedings of the 2021 USENIX Annual Technical Conference (ATC)*. 47–63.

[48] George Prekas, Marios Kogias, and Edouard Bugnion. 2017. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*. 325–341.

[49] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. 2018. VM Live Migration At Scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*. 45–56.

[50] Xiang Song, Jicheng Shi, Ran Liu, Jian Yang, and Haibo Chen. 2013. Parallelizing Live Migration of Virtual Machines. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*. 85–96.

[51] Xin Xu and Bhavesh Davda. 2016. SRVM: Hypervisor Support for Live Migration with Passthrough SR-IOV Network Devices. *ACM SIGPLAN Notices* 51, 7 (2016), 65–77.

[52] Fei Zhang, Guangming Liu, Xiaoming Fu, and Ramin Yahyapour. 2018. A Survey on Virtual Machine Migration: Challenges, Techniques, and Open Issues. *IEEE Communications Surveys & Tutorials* 20, 2 (2018), 1206–1243.

[53] Jiechen Zhao, Iris Uwizeyimana, Karthik Ganesan, Mark C. Jeffrey, and Natalie Enright Jerger. 2022. Altocumulus: Scalable Scheduling for Nanosecond-scale Remote Procedure Calls. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 423–440.

[54] Guanwen Zhong, Aditya Kolekar, Burin Amornpaisannon, Inho Choi, Haris Javaid, and Mario Baldi. 2023. A Primer on RecoNIC: RDMA-enabled Compute Offloading on SmartNIC. *arXiv preprint arXiv:2312.06207* (2023).

Received 2nd May 2024; accepted 1st July 2024