Zero-Copy, Minimal-Blackout Virtual Machine Migrations Using Disaggregated Shared Memory

Andreas Grapentin
Operating Systems and Middleware Group, Hasso Plattner Institute,
University of Potsdam, Germany

Felix Eberhardt
Operating Systems and Middleware Group, Hasso Plattner Institute,
University of Potsdam, Germany

Tobias Zagorni
Operating Systems and Middleware Group, Hasso Plattner Institute,
University of Potsdam, Germany

Andreas Polze
Operating Systems and Middleware Group, Hasso Plattner Institute,
University of Potsdam, Germany

Michele Gazzetti
IBM Research Europe, Dublin, Ireland

Christian Pinto
IBM Research Europe, Dublin, Ireland

Abstract

We propose a new live-migration paradigm for virtual machines called *zero-copy migration*. By making the working set of the virtual machine available on the destination host through transparently byte-addressable disaggregated memory, we remove the need for a pre-copy phase while simultaneously reducing the performance impact of the post-copy phase. We describe an open-source implementation of the proposed paradigm based on QEMU-KVM and libvirt, and we evaluate the efficiency of the approach with a deployment on a functional hardware prototype of a memory disaggregation system realized using ThymesisFlow. Using a series of configurable benchmarks, we show that the lead time and blackout time of the migration are equal to best-case scenarios of traditional pre-copy, post-copy and hybrid approaches. Key performance metrics from the perspective of applications running in the virtual machine, such as memory latency and throughput, are improved by up to three orders of magnitude, increasing both flexibility and responsiveness of live-migrations in the datacenter.

2012 ACM Subject Classification Hardware → Memory and dense storage; Computer systems organization → Cloud computing; Software and its engineering → Virtual machines; Software and its engineering → Distributed memory; Software and its engineering → Cloud computing; Information systems → Data centers; Information systems → Computing platforms

Keywords and phrases disaggregation, disaggregated memory, vm live migration, thymesisflow, power9, opencapi, performance evaluation, zero copy

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2024.3

Supplementary Material *Software*: `https://github.com/disaggr/qemu/tree/thymesisflow` archived at `swh:1:dir:b1c1f4d8767857b66e0d2b1c6f2c76177ec9e976`

1 Introduction & Motivation

Flexibility of deployment and serviceability of the infrastructure are key features of the cloud datacenter. Workloads are contained in Virtual Machine (VM) instances as a unit of deployment with diverse resource footprints. This results in an unbalanced allocation of server

15th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 13th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2024).
Editors: João Bispo, Sotirios Xydis, Serena Curzel, and Luís Miguel Sousa; Article No. 3; pp. 3:1–3:13

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resources that creates fragmentation over time. VM migration is a useful tool to rearrange workloads, and both mitigate resource fragmentation as well as respond dynamically to deployment challenges without losing availability. However, live-migrating a virtual machine between physical machines incurs a severe penalty on the service quality of the contained workload. Both in literature and in practice it has been established that the performance implications of migrating VM instances with large memory footprints and write-intensive guest workloads can be so substantantial as to render the instance un-migratable [17].

However, modern applications such as graph analytics, in-memory databases and artificial intelligence oriented applications require increasingly large amounts of memory, in the order of hundreds of Gigabytes[21]. Migrating VMs hosting such workloads and transferring such large amounts of data over the network saturates the available network bandwidth, interfering with applications running on the infrastructure, and further exacerbating the performance issues of traditional live-migration approaches.

Techniques exist that are aimed at mitigating these issues, such as using Remote DMAs (RDMA) [9]. While these solutions are indeed effective, they don't completely mitigate the latency degradation during post-copy transfers, and consequently don't solve the problem of migrating very large VM instances.

The advent of Composable Disaggregated Infrastructures (CDI) [3], and in particular the ability of accessing remote memory over a standard PCIe 5.0 fabric thanks to Computer Express Link (CXL) [5] fabric interconnect standard, opens new avenues for improving virtual machine migration techniques. In a CDI, the physical boundaries of a server become indistinct. Memory and compute resources are broken into disaggregated components over a dedicated network fabric and can be composed via software. Existing prototypes show that in the near future, systems may become commercially available that have the ability of transparently accessing memory from a shared pool or from neighboring nodes with a load/store semantic. CXL also shows promise of breaking the bandwidth bottleneck of modern x86 based server processors [2] in a similar manner that the introduction of the Open Memory Interface (OMI) allowed on POWER10™ systems. CXL over ethernet is also being explored [20], both with the possibility of forming memory pools, but also for allowing disaggregated access to a nodes private memory.

In this paper we present the first implementation of virtual machine live-migration based on disaggregated memory. To evaluate the approach on real hardware, we utilize the open-source disaggregated memory prototype, ThymesisFlow [18] that enables borrowing memory from a neighboring machine. Although ThymesisFlow relies on the OpenCAPI [14] coherent interconnect and is targeting IBM POWER9™ based server systems, the approaches outlined in this paper are independent of the choice of disaggregated memory interconnect and are applicable to similar architectures created on the basis of CXL or other technologies. We investigate how traditional live-migrations of virtual machines can utilize disaggregated memory as a medium for memory access and transfer, in order to improve the performance metrics from the perspective of both the hypervisor and the guest workload during the migration.

2 State of the Art

VM live migration was first presented as a response to the necessity for more flexible workload distribution in the datacenter by Clark et al. and Nelson et al. on both the Xen and VMWare VSphere virtualization platforms [1][4][12]. Virtual machine instances present the opportunity of migrating self-contained units of work, mutually independent

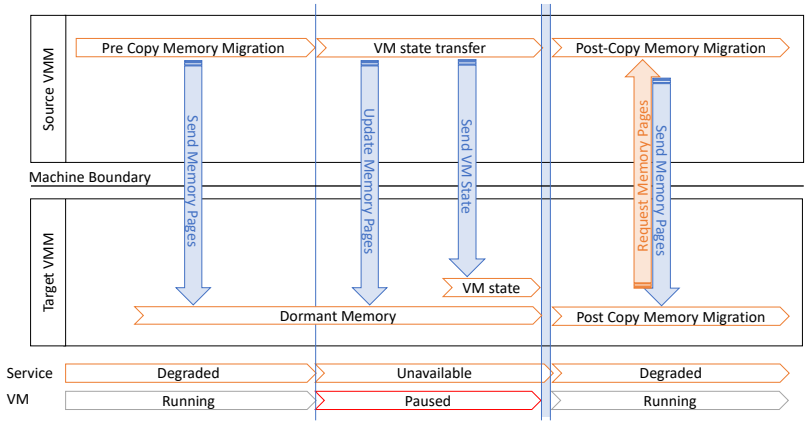


Figure 1 The phases of virtual machine migration, highlighting the duration in which the machine is paused and thus the service unavailable (*blackout* time) as well as the duration in which the service is degraded (*brownout* time).

and decoupled from the operating system, as opposed to previous work which focused on the migration of individual processes. Later approaches extended the scope of live virtual machine migration to new transfer media such as wide area networks [7] and RDMA [19]. To further mitigate the performance impact of live-migrations, the *pre-copy* paradigm heavily utilized in traditional VM migrations has been steadily improved [8, 11] and the *post-copy* paradigm was introduced [6], enabling hybrid migrations that combine the benefits of pre- and post-copy. In 2018, Ruprecht et al. have described the application of VM migration in Google datacenters and outlined the necessity for an improved deployment flexibility with minimal application downtime. They have also shown that using state-of-the-art migration technologies, the performance impact on guest systems is still a relevant concern [16].

2.1 Sequence of a Traditional Hybrid VM Live-Migration

Figure 1 outlines the sequence of a standard virtual machine live-migration utilizing both a *pre-* and *post-copy* phase and summarizes their associated impact on the guest. First is a *pre-copy* phase, in which memory pages are transferred concurrently from the *source* to the *destination* host. The degradation of the guest system due to the introduced contention on the memory subsystem is described as a *pre-copy brownout*.

After concluding the pre-copy phase, the virtual machine is paused on the source and the state of the runtime is transferred to the destination to be resumed there. In the *blackout* time between pausing and resuming the virtual machine, the services provided by the guest are unavailable.

Lastly, when the guest has been successfully resumed on the destination, the remaining memory pages are transferred in a *post-copy* phase. During this phase the guest performance can be degraded significantly if memory pages are accessed that are not yet locally available on the destination, leading to a *post-copy brownout* where the application must wait for the memory to be transferred from the source before execution can continue.

2.2 Beyond State of the Art

While both post-copy and pre-copy phases of the virtual machine migration have been shown to be useful techniques to reduce the duration of the blackout time and increase service availability, both techniques have disadvantages. During the pre-copy phase, pages that are overwritten (*dirtied*) by the guest after successful transfer to the destination will need to be invalidated, and eventually re-transferred. As a consequence, workloads that produce frequent write accesses drastically reduce the effectiveness of a pre-copy phase[10]. Conceptually, the pre-copy phase also introduces a *lead-time* delay between initiating the migration and the release of the compute resources on the source, which may be undesirable when the migration needs to be performed quickly, for example to reduce thermal load.

Similarly, the post-copy phase can cause severe performance issues if the guest frequently needs to wait for the remote memory to become available. Access to pages not yet migrated would require the generation of traffic on the network, a physical link characterized by much higher latency compared to the memory bus. As a consequence, applications that exhibit unfavorable *memory access patterns* can become unresponsive during traditional migrations, leading to service downtime much longer than the machines apparent blackout time.

The introduction of disaggregated memory with the paradigm of zero-copy live-migrations allows the migration of virtual machines unimpeded by these issues. Zero-copy live-migration means that the memory of the guest can be made immediately and transparently byte-addressable on the destination without the need to pre-copy any memory, and without incurring expensive page-faults when accessing remote memory. The virtual machine can be paused on the source and immediately resumed on the destination, eliminating the lead-time delay and minimizing the blackout time. Coupled with a modified post-copy phase also utilizing the disaggregated memory to efficiently copy and re-map the memory from the source to the destination, the virtual machine can be migrated completely with greatly reduced performance degradation during the post-copy brownout phase when compared to traditional live-migrations.

Very recently, first steps have been made to design a virtual machine migration over CXL by the *nil-migration* project [13], utilizing CXL 3.0 to create a shared disaggregated memory pool device. To perform the migration, the working set of the virtual machine on the source will be transferred to pooled memory, and then transferred back to local memory on the destination. However, no concrete implementation and evaluation of the technique is available at the time of writing this article. In our prototype, we don't use a memory pool and instead directly promote the private memory of the source machine that contains the working set of the virtual machine instance to shared memory. However, the same approach could be used in case of a pooled memory system.

3 Architecture of the Memory Disaggregation Prototype

This work is based on a real hardware-based disaggregated shared memory system, implemented thanks to the ThymesisFlow [15] [18] open-source project. ThymesisFlow is a HW/SW framework that uses OpenCAPI [14] attached FPGAs to enable memory disaggregation across IBM POWER9™ servers via the memory borrowing template (Figure 2). The minimum working system is composed of two nodes: a *lender* and a *borrower*. The *lender* node is capable of reserving a portion of its local main memory for access from a remote machine. The borrower accesses memory from a remote lender using load/store semantics, as if the memory was physically attached to it. The remote memory is attached to the lender machine using either the abstraction of an additional, CPU-less, NUMA node or as a memory

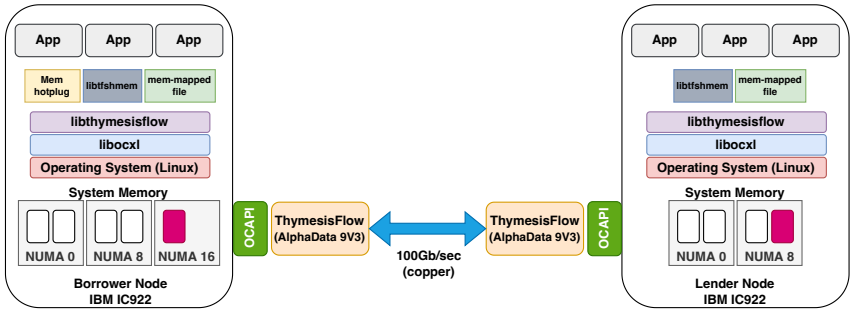


Figure 2 Abstract representation of the ThymesisFlow HW/SW prototype.

mapped file. On the hardware side, each node part of a ThymesisFlow deployment must support OpenCAPI and be equipped with a supported FPGA card. In the case of this paper, we used IBM IC922 POWER9™ servers and AlphaData 9V3 FGPAs.

On the software side, ThymesisFlow offers *libthymesisflow* which consists of a set of user-space applications that are used for controlling the configuration of the FPGAs and the actual attachment of memory to the borrower OS memory subsystem. Memory in ThymesisFlow can be assigned in exclusive mode to the borrower or shared between borrower and lender machine. In the exclusive mode, the borrower can decide to hotplug the memory to the OS or to access it via a custom `mmap` and Linux character device (useful for assigning memory to a specific application/process). In the shared case, instead, both parties can either use the shared memory library (*libtfshmem*) provided as part of the ThymesisFlow suite or have the memory exposed as a memory mapped file. In this paper we leverage the shared memory via memory mapped file.

Logically this disaggregated memory architecture is what we call a *peer-to-peer* approach, where every node in the system is able to map every other nodes memory. In contrast, there is the *pool* architecture with a separate memory pool, which nodes with private memory can map and share. While we require the former architecture, it is possible to adapt our prototype to the latter as well. Additionally, even though the underlying technology is OpenCAPI, which has been merged with the CXL Consortium in 2022, our techniques can be applied to CXL as well. That said, the integration of CXL in Thymesisflow is beyond the scope of this work.

4 Design of the VM Migration

We implemented the disaggregated zero-copy virtual machine migration by extending the open-source virtualization and system emulation software *QEMU-KVM*[1]. *QEMU-KVM* is a *type 1 hypervisor* in Linux, capable of supporting both hardware accelerated virtualization backends, as well as advanced platform and orchestration abstractions such as *libvirt*. For the live-migration of virtual machines, QEMU already natively supports the pre- and post-copy paradigms. The existing advanced migration capabilities of *QEMU*, coupled with the well-documented and active open-source codebase and community made it an ideal testbed for implementing the zero-copy live migration. To implement our prototype, we extended QEMU to enable a migration from the disaggregated memory *lender* to the *borrower*.

[1] Our modifications to QEMU are also open source and available via `https://github.com/disaggr/qemu/tree/thymesisflow`

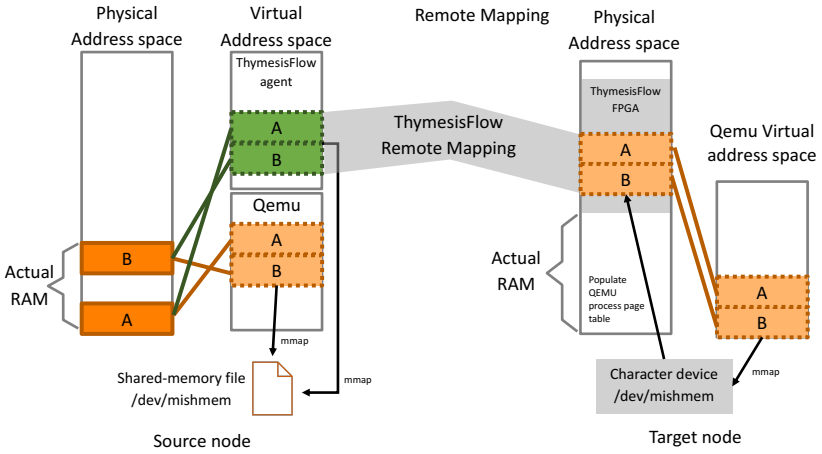


Figure 3 A conceptual visualization of the page table mappings provided to QEMU during the zero-copy migration.

To enable the disaggregated live-migration, we replace the live-migration memory moving behavior of QEMU both on the source and the destination by supplying the relevant function tables with custom implementations. We disable the pre-migration handling of all RAM blocks, so no pre-copy phase can occur on the source, and install a custom handler for the post-copy phase.

4.1 Organizing the Memory Backing Store

In order to make the disaggregated memory available to the virtual machine on both the source and destination, we utilize the file-mapped memory backends implemented in QEMU to provide the VM with the same view of its memory on both the borrower and lender side. To achieve this, we create a shared memory segment of sufficient size on the lender side and supply its path to QEMU. This memory segment is shared with the *ThymesisFlow Agent* that communicates with the FPGA and that owns the memory that is lent to the borrower. By using a shared memory segment for this lent memory, we can make sure that the memory presented on the lender side is consistent with the memory on the borrower side after migration.

On the borrower, the system is configured with a custom kernel module to interface with the FPGA and present the remote memory as a special character device that supports the *mmap* system call. We configure QEMU to use this character device as mapping file for the VM memory. This ensures that the order of the memory pages is preserved from lender to borrower side, making the view on the physical address space from the perspective of the guest identical on both source and destination. The layout of the addresses and translations is visualized in Figure 3. This way, with one exception outlined below, all memory remains transparently accessible to the guest machine at all times, even while the memory is being copied during the post-copy phase.

4.2 Disaggregated Post-Copy Transfers

To implement a disaggregated post-copy transfer, we start a concurrent thread that incrementally moves the VM memory from the source to the destination by copying and immediately remapping pages of memory as outlined in Figure 4.

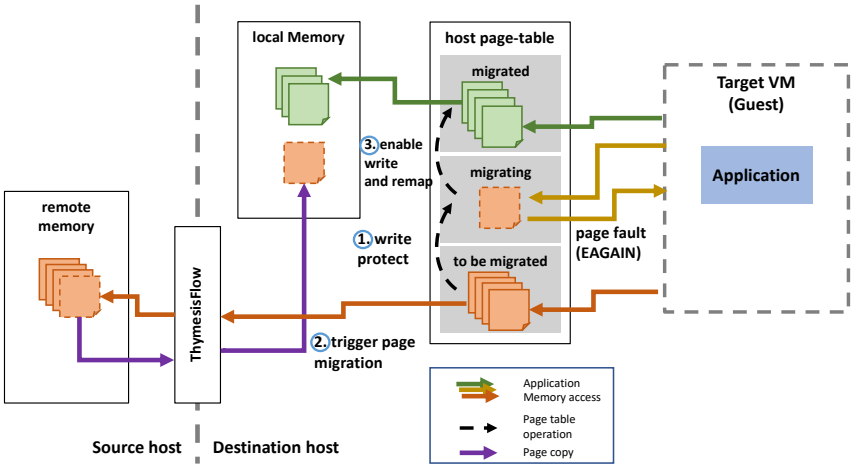


Figure 4 A conceptual visualization of the RAM transfer in QEMU during the disaggregated post-copy phase.

Each page is transferred in three steps. Firstly, to avoid data loss during the small time window of transfer, the page is briefly locked for writing, such that no VM thread may be modifying a page that is currently being moved from source to destination. This lock causes an access violation to occur in case the guest does write to the locked memory, upon which the failed memory access is deferred and retried. Secondly, the page is transferred from the lender to the borrower. Finally, the page is used to replace the mapping of the corresponding page of remote memory in the VMs working set, and the write lock is lifted, completing the page transfer.

4.3 Finishing the Migration

The remainder of the virtual machine state, such as the CPU registers and the device configurations, are transferred via the network using the existing implementation in QEMU. If the disaggregated post-copy phase is enabled, all memory of the VM will be local on the destination after the migration completes. Otherwise, in *cpu-only* mode, the live-migration completes with all memory remaining remotely but still entirely transparently byte-addressable on the borrower machine.

5 Performance Evaluation

In the evaluation of the performance characteristics of the different types of VM migration, we distinguish between two types of performance metrics that we can observe in our testbed.

- *Macro-properties* of the migration are performance characteristics observable by the host system and the virtual machine process, including the *total time of migration*, *total ram transferred* and the *blackout* and *pre- and post-brownout times* of the guest.
- *Micro-properties* of the migration are performance characteristics that are internal to the VM and are observable by the guest workload processes. These metrics include the *memory latency* and the *application throughput* during the migration.

To evaluate the performance characteristics of the zero-copy virtual machine migration, we compare our approach against state-of-the-art hybrid live-migration and analyze individual migration runs to gain insight on the behavior of the guest in order to determine the impact of a migration.

5.1 Infrastructure setup and orchestration

The testbed used for this evaluation is described in Section 3. A third machine on the same network was used to host the evaluation orchestrator and remotely collect the performance data of the migration.

We use *ansible* playbooks and *libvirt*[2] to automate repeatable runs of the benchmarks, as well as orchestrate the virtual machines and make the required hypervisor calls. The virtual machine subjected to the live-migration is a *Debian bullseye GNU/Linux* installation with 2 virtual CPUs, 20GiB of RAM, and configured with an unmodified kernel and firmware[3].

5.2 Description of the Benchmarks

One of the most influential factors on the migration of a virtual machine is the memory behavior of the guest. In particular, the rate at which the guest machine produces *dirtied* pages of memory can influence both the macro and micro performance characteristics of the migration. A second important factor to consider is the memory access pattern of the guest workload. For this reason, we implemented the *SMOG* microbenchmark suite[4] capable of producing dirtied pages in the guest at a configurable rate. We also extended *SMOG* to be capable of accessing memory at configurable access patterns, such as a randomized pointer-chaser workload. These configurable workloads are a useful tool to examine the micro performance characteristics of the guest during migration, in particular memory latency and throughput.

5.3 Memory Latency Degradation during the VM Migration

In this work, we try to investigate the detailed performance characteristics of the guest under the stress of the migration, where certain areas of the memory may still be unavailable, or only available at high latency costs.

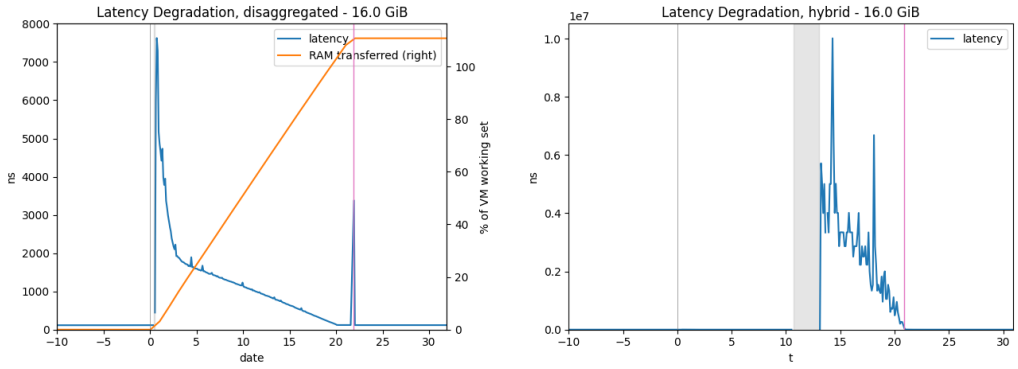
Figure 5 visualizes the average latency in nanoseconds of the randomized pointer-chasing workload of *SMOG* over a 16GiB memory region over the course of the migration. The graphs show that before the migration starts and after it concludes, the memory latency rests at a consistent low average of 80 nanoseconds, which is in alignment with the local memory latency of the machines in our testbed.

During the migration, the impact of the remote memory on the workloads average latency can be separated into two phases. With the start of the migration at a time of 0.0 seconds, the measurements show a sharp increase in latency. This is due to the high latency of the memory, which at this point resides entirely in the remote memory of the lending machine. Since the caches on the borrowing machine are cold and the page tables of the guest machine, which are required by the guest to make memory accesses are also not prefetched, the first accesses to the remote memory are expensive. Through the caching of the page table structures in the L3 cache as well as the beginning transfer of the physical memory region of the VM, we can observe a sharp decrease of the memory latency during the first few seconds of the migration. Once the majority of the page tables used by the benchmark have been transferred or cached,

[2] The playbooks and scripts used to create the performance measurements used in this evaluation are available on github: `https://github.com/disaggr/vm-migration-public`

[3] The scripts used to bootstrap and configure the guest systems is available on github via `https://github.com/disaggr/qemu-image-factory`

[4] the SMOG benchmark suite is open-source and available on github via `https://github.com/disaggr/smog`



(a) Memory latency degradation during a disaggregated post-copy migration.

(b) Memory latency degradation during a traditional hybrid migration.

Figure 5 The memory latency as measured by a randomized pointer-chasing workload during both state-of-the-art hybrid and disaggregated zero-copy migrations.

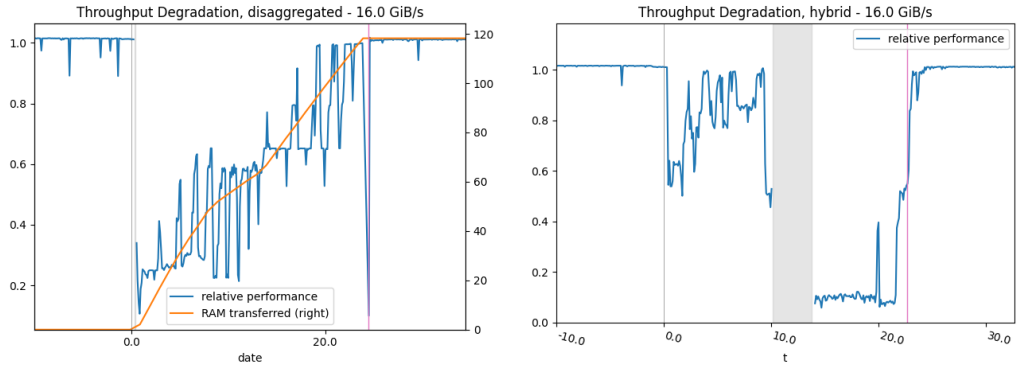
the memory latency decrement levels out into the second phase, in which the amount of transferred working set pages of the pointer chasing workload determines the average latency, which decreases linearly with the remaining portion of remote pages.

In total, approximately 10 percent of the memory has to be transferred more than once during the migration. This corresponds to cache lines that are requested by the guest MMU and provided by ThymesisFlow before the pages are transferred by the RAM mover thread in qemu. When compared to the hybrid migration, it becomes evident that the average memory latency of the guest workload during the zero-copy migration is up to 3 orders of magnitude lower than for the state-of-the-art approach. There are two reasons for this dramatic difference. Firstly, the post-copy migration relies on expensive page-fault mechanisms to make the remote memory locally available. Secondly, the disaggregated memory benefits from a finer granularity of the transferred memory, since only cache-lines of 128 Bytes are transferred for each remote memory request, whereas traditional post-copy transfers entire pages of 4 KiB of memory, which further impacts the average memory latency.

5.4 Memory Throughput Degradation during the VM Migration

The second metric we investigated is the throughput of SMOG processing a sequential combined read/write workload with strides equal to the memory page size. For this benchmark, SMOG was configured to a nominal throughput of 16 GiB/s of memory pages every second, and we measured the actual throughput during migration and visualized the results in Figure 6.

Similarly to the compared memory latency metrics, the throughput of the benchmark running in the disaggregated post-copy phase consistently outperforms the benchmark running in the traditional post-copy phase. It is noteworthy that the pre-copy phase of the hybrid migration provides little benefit to both latency and throughput due to the write-heavy workload configuration.



(a) Memory throughput degradation during a disaggregated post-copy migration.

(b) Memory throughput degradation during a traditional hybrid migration.

Figure 6 The memory throughput as measured sequential memory read/write workload during both state-of-the-art hybrid and disaggregated zero-copy migrations.

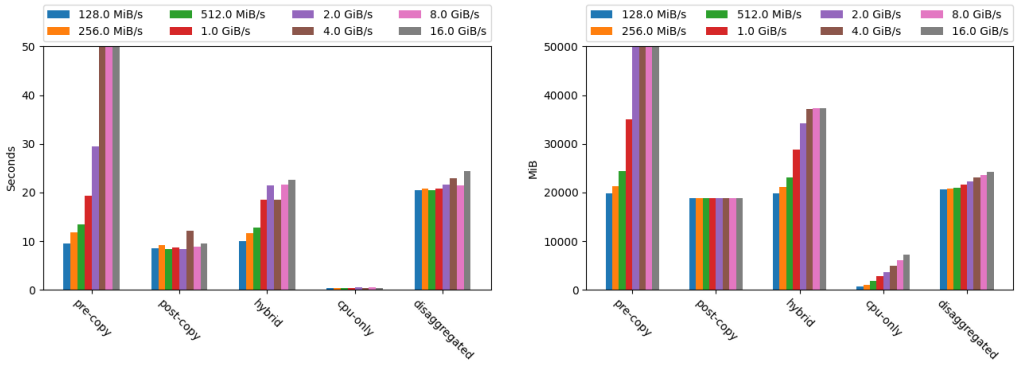
5.5 Analysis of Migration Times and Transferred Ram

We additionally compared two types of performance metrics of the virtual machine migration from the perspective of the hypervisor. Firstly, the total amount of memory transferred as an indicator of increased contention on the network, which may be shared between the implementation of the migration and services running on the same network. In this context, the following times are particularly of interest.

- *Total Time of Migration* – The time between initiating the migration until it is completed.
- *Blackout Time* – The time in which the guest is paused, and no CPU cycles are allocated to applications and services running in the guest.
- *Pre- and Post-Copy Brownout Time* – The time in which the performance of the applications and services running in the guest are degraded because of an active pre-copy or post-copy transfer.
- *Migration Delay* – The time between initiating the migration until the execution of the guest is paused and the last CPU cycle is allocated to the guest and its applications and services running on the source machine.

Some of these metrics are independent of the differences between a zero-copy and a traditional live-migration. For instance, the blackout time of zero-copy live-migrations is guaranteed to be equal to the best case of traditional migrations. Also, the total time of migration as well as the duration of the brownouts will only depend on the size of the VM working set and the bandwidth of the transfer medium. However, by using a dedicated medium for the disaggregated memory transfer, we reduce resource contention on the network, which may be relevant to some applications. A comparison of the total time of migration, and the related metric of total amount of ram transferred are outlined in Figure 7 for different amounts of memory dirtying rate by the guest workload, as well as different types of VM migration.

Our measurements reaffirmed the known issue that pure pre-copy migrations can become unable to meet the threshold of valid pages on the destination for higher rates of dirtied memory in the guest. Because these are academic edge-cases, we are not showing the full extent of these effects on the plots. In real world scenarios, typically a hybrid live-migration approach is used, which limits the number of passes during the pre-copy phase, but still uses



(a) The total time of migration.

(b) The total amount of ram transferred.

Figure 7 A comparison of the total time of migration and total transferred memory during both traditional and disaggregated virtual machine migrations.

a pre-copy phase to help mitigate the performance impact of the post-copy phase. In our experiments, the hybrid migration introduced a maximum overhead of the size of the memory region used by the benchmark. The disaggregated migrations also incur a slight overhead in transferred ram due to memory that is being accessed remotely by the application before being transferred to the borrower. This overhead is also visible in the total time of migration, but this comparison is dominated by a difference in the throughput between our hardware prototype and the network. We expect that future hardware for disaggregated memory will show even more impressive results and easily bridge that gap in throughput.

In consequence, the zero-copy virtual machine migration paradigm is capable of migrating virtual machine instances regardless of working set size or workload behavior with both a greatly reduced performance impact on the guest, as well as a more predictable runtime behavior than traditional virtual machine migration techniques, pushing the boundaries for datacenter workload placement flexibility and transparency of deployment.

6 Conclusions and Future Direction

In this paper, we have introduced the concept of a disaggregated zero-copy virtual machine migration utilizing byte-addressable, disaggregated shared memory, for which we have used the ThymesisFlow prototype technology. We have shown that using disaggregated memory for the resource sharing and transfer during a virtual machine migration is beneficial in two ways. Firstly, from the perspective of the hypervisor it shows potential to improve the flexibility of migrating resources through reduced delays, and to mitigate problems with write-intensive guests. Secondly, from the perspective of the application the degradation of memory latency and throughput are greatly reduced when compared to traditional post-copy migrations. In conclusion, we believe that resource migration, and in particular virtual machine migrations in cloud datacenters are a good use-case for newly emerging memory disaggregation technologies. In future work, it needs to be evaluated whether memory pools could also be utilized in a similar manner, and how CXL shared resources as defined in the recently released CXL 3.0 standard could be utilized to that regard, to make the concept outlined in this work more widely available. In the future, fully disaggregated virtual machine working sets could become possible, where the migration could be reduced to merely reallocating the compute resources of the virtual machine instances inside the datacenter.

References

1 Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5):164–177, 2003.

2 Albert Cho, Anish Saxena, Moinuddin Qureshi, and Alexandros Daglis. A case for cxl-centric server processors. *arXiv preprint*, 2023. `arXiv:2305.05033`.

3 I-Hsin Chung, Bulent Abali, and Paul Crumley. Towards a composable computer system. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, pages 137–147, 2018.

4 Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

5 Computer Express Link. `https://www.computeexpresslink.org/`.

6 Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review*, 43(3):14–26, 2009.

7 Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K Panda. High performance virtual machine migration with rdma over modern interconnects. In *2007 IEEE International Conference on Cluster Computing*, pages 11–20. IEEE, 2007.

8 Khaled Z Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. Optimized pre-copy live migration for memory intensive applications. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2011.

9 C. Isci, J. Liu, B. Abali, J. O. Kephart, and J. Kouloheris. Improving server utilization using fast virtual machine migration. *IBM Journal of Research and Development*, 55(6):4:1–4:12, 2011. `doi:10.1147/JRD.2011.2167775`.

10 Canturk Isci, Jiuxing Liu, Bülent Abali, Jeffrey O Kephart, and Jack Kouloheris. Improving server utilization using fast virtual machine migration. *IBM Journal of Research and Development*, 55(6):4–1, 2011.

11 Yuji Muraoka and Kenichi Kourai. Efficient migration of large-memory vms using private virtual memory. In *Advances in Intelligent Networking and Collaborative Systems: The 11th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2019)*, pages 380–389. Springer, 2020.

12 Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. Fast transparent migration for virtual machines. In *USENIX Annual technical conference, general track*, pages 391–394, 2005.

13 nil-migration. `https://nil-migration.org`.

14 OpenCAPI Consortium. OpenCAPI Specification. Online: `http://opencapi.org`, 2017. Accessed: January 2019.

15 C. Pinto, D. Syrivelis, M. Gazzetti, P. Koutsovasilis, A. Reale, K. Katrinis, and H. Hofstee. Thymesisflow: A software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 868–880, Los Alamitos, CA, USA, October 2020. IEEE Computer Society. `doi:10.1109/MICRO50266.2020.00075`.

16 Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. Vm live migration at scale. *ACM SIGPLAN Notices*, 53(3):45–56, 2018.

17 Petter Svard, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 111–120, 2011.

18 ThymesisFlow. `https://github.com/OpenCAPI/ThymesisFlow`.

19 Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees De Laat, Joe Mambretti, Inder Monga, Bas Van Oudenaarde, Satish Raghunath, and Phil Yonghui Wang. Seamless live migration of virtual machines over the man/wan. *Future Generation Computer Systems*, 22(8):901–907, 2006.

20 Chenjiu Wang, Ke He, Ruiqi Fan, Xiaonan Wang, Yang Kong, Wei Wang, and Qinfen Hao. Cxl over ethernet: A novel fpga-based memory disaggregation design in data centers. *arXiv preprint*, 2023. `arXiv:2302.08055`.

21 Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948, 2015. `doi:10.1109/TKDE.2015.2427795`.