Template-aware Live Migration of Virtual Machines

Roja Eswaran, Mingjie Yan, Kartik Gopalan
{reswara1,myan28,kartik}@binghamton.edu
Computer Science, Binghamton University
Binghamton, NY, USA

ABSTRACT

One of the key challenges of edge computing is working with a limited amount of resources available at the edge, especially memory and bandwidth. Virtual Machine (VM) Templating is a technique to start multiple VM instances quickly from a shared pre-configured read-only image (or template). The new VM instances share the memory of the template in a copy-on-write (COW) manner. In edge computing platforms, VM templating can help to reduce the collective memory footprint and deployment time of multiple VMs. Live migration of VMs can also improve task placement on edge nodes for latency reduction, service availability, and cost-effectiveness. However, existing live migration techniques fail to maintain memory sharing among multiple templated VMs that are migrated to a common destination. Consequently, identical pages at the source are replicated several times at the destination, increasing memory pressure on the destination node, network traffic during migration, and total migration time. Lack of templating awareness can also trigger migration failure if the destination lacks sufficient memory to accommodate the increased memory footprint. To address this shortcoming of live migration, we introduce Template-aware Live Migration (TLM), which preserves preexisting COW memory sharing between templated VMs that are migrated to a common destination machine. Specifically, TLM ensures that multiple virtual pages from different VMs that are mapped to the same template page at the source are mapped to the same page at the destination. We implement TLM on the QEMU/KVM virtualization platform and demonstrate a significant reduction in memory footprint, shorter migration time, and reduced network traffic.

ACM Reference Format:
Roja Eswaran, Mingjie Yan, Kartik Gopalan. 2023. Template-aware Live Migration of Virtual Machines. In *The Eighth ACM/IEEE Symposium on Edge Computing (SEC '23) EdgeComm Workshop, December 6–9, 2023, Wilmington, DE, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583740.3626812

1 INTRODUCTION

Multi-access edge computing nodes offer an attractive option for executing tasks where users require low latency and high bandwidth [13, 16]. Virtual machines (VMs) can ensure both isolation and efficient resource utilization at the edge computing infrastructure [5, 15, 26]. Live migration [4, 10] is a key technology in an edge computing infrastructure that transfers running VMs from one physical node to another. It is widely used for a variety of purposes, such as load balancing [3, 8, 20], meeting service level agreements [17], energy savings [22], and seamless maintenance of physical edge nodes.

Multiple VMs that run different components of a distributed application may be colocated on the same physical node to reduce their inter-VM communication costs [14, 24]. The colocated VMs may run different instances of same guest operating system (OS) or similar applications. VM templating [1, 12, 18, 25] is one approach to quickly instantiate multiple lightweight VMs from a common read-only image, called a template, which is shared copy-on-write (COW) memory among the VM instances [2, 21]. Templated VM instances may often need to be migrated together to the same destination node for various reasons. For example, templated VM instances that run different components of a distributed application may require migration to the same destination node to maintain low inter-VM communication latency or to meet other QoS targets. Additionally, physical node availability, hardware availability, and multi-tenancy limitations may necessitate the migration of templated VM instances to the same destination node.

Unfortunately, current live VM migration techniques do not consider page sharing among templated VM instances that are being migrated to the same destination. As a result, shared pages are transferred and replicated multiple times, as if they were separate pages, resulting in an increase in memory pressure at the destination node. This can also lead to migration failures when the edge destination lacks sufficient memory to accommodate the expanded memory footprint of the VMs that were comfortably co-located at the source. Replication of previously shared pages among VMs also results in longer migration time and increased network traffic.

Previous approaches to reduce the transfer of duplicate pages during live migration (such as [5, 6, 9, 15, 26] among others) use content-based hashing to detect identical pages and avoid their retransmission. However, they do not identify or maintain preexisting COW mappings among VMs when pages are transferred to the destination. Additionally, while hashing may be used to identify identical pages that are not COW-shared, it is unnecessary and computationally expensive for COW-shared pages.

We address this problem of memory footprint expansion of templated VMs instances as they are live migrated together to a common destination node. Our contributions are as follows:

(1) We identify and demonstrate the problems caused by live migration being unaware of underlying page sharing among templated VM instances, leading to a larger memory footprint at the destination, longer total migration time, and higher network traffic.

SEC '23, December 6–9, 2023, Wilmington, DE, USA

ACM ISBN 979-8-4007-0123-8/23/12…$15.00
https://doi.org/10.1145/3583740.3626812

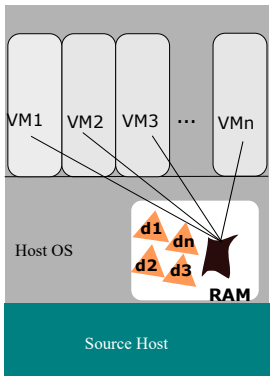

Figure 1: VM Templating: Multiple VMs can be started from a common shared template to reduce their startup times and initial memory footprint. Memory pages that are dirtied by a VM (represented by deltas *dk*) are not shared.

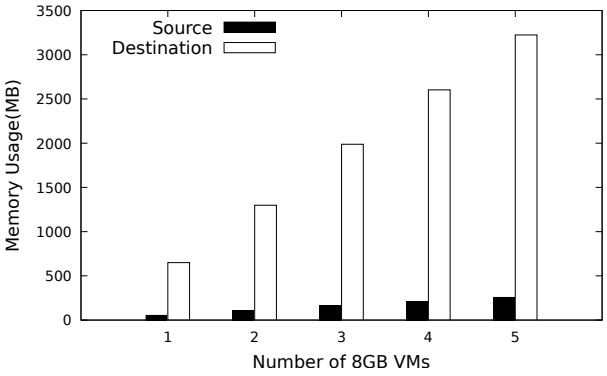

Figure 2: Memory footprint increase at destination when the templated VMs are migrated using generic pre-copy

(2) We present *Templating-aware Live Migration* (TLM) which migrates templated VM instances to a common destination while maintaining COW memory sharing with the base template. We also discuss potential ways to account for other forms of page sharing during live migration besides those due to templating.
(3) We implement prototype of TLM in KVM/QEMU [11] virtualization platform and evaluate it using several benchmarks. Besides maintaining preexisting page sharings among templated VMs at the destination machine, TLM reduces the total migration time by up to 95.37% and network traffic by up to 92.15%.

In the rest of this paper, we first present the background on pre-copy live migration techniques and demonstrate the problem. Next, we present the design and implementation of TLM followed by its evaluation. The paper concludes with a discussion of related work and summary of results.

2 BACKGROUND AND PROBLEM DEMONSTRATION

VM Templating: Instantiating a VM from scratch typically takes a long time because it involves initialization of software, guest OS, and virtual hardware, including time to load the corresponding contents to memory from the disk. VM templating allows multiple new VM instances to be quickly instantiated from a single pre-checkpointed VM image (or template). Figure 1 shows the steps involved in instantiating VMs from a template image. First step is to save a custom VM template as a snapshot of a pre-booted VM that is pre-initialized with necessary software. The template image can be pre-loaded into memory to reduce disk access latency. Next step is to quickly instantiate multiple VM instances from the common template image by mapping the base template image COW into the memory of each new VM instance. This is essentially a variant of a typical checkpoint-restore operation, where we checkpoint the base image once and restore it multiple times for concurrent VM instances.

Pre-copy Live Migration: Pre-copy live migration [4] is the most common technique to migrate VMs running on one physical host (called the source) to another physical machine (called the destination). It works by first transferring the VM's memory pages to the destination, even as the VM continues running at the source, and then transfers the CPU execution state at the end; hence the name *pre*-copy which means to transfer memory before CPU state. But, as the VM's memory is being transferred, its virtual CPUs (VCPUs) may write to previously transferred pages, thus *dirtying* them again. Such dirtied pages are retransmitted over multiple iterations, until very few dirtied pages remain, at which point the VM is paused, its execution state and remaining dirty pages are transferred, and the VM is resumed at the destination.

Problem Demonstration: In Figure 2, we show the problem that arises during the migration of templated VMs using pre-copy. The X-axis represents the number of VMs instantiated from a shared template that are being migrated, and the Y-axis represents their collective memory footprint as measured by the `free` command in Linux. Prior to migration, we measured the memory usage of the templated VMs at the source. Subsequently, we measured memory usage again at the destination after migration. Since the generic pre-copy live migration is unaware of page sharing among templated VMs, it redundantly transfers and replicates identical pages, breaking the underlying COW sharing. Figure 2 shows that the lack of templating awareness increases the total memory footprint at the destination, the network traffic during migration, and the total migration time.

3 TEMPLATE-AWARE LIVE MIGRATION (TLM)

Traditional pre-copy live migration is unaware of the underlying COW page sharing among templated VMs. Hence it ends up transferring the shared base pages of the template multiple times. We propose Template-aware Live Migration (TLM) which addresses this shortcoming of pre-copy migration by ensuring that multiple templated VM instances maintain their COW page sharing with the base template even at the destination node and transfers only the *delta* pages that differ among various VM instances. TLM requires that we first track delta (or dirty) pages for VM instances before

migration. Then, during TLM, only the delta pages are transferred for each instance. We describe these steps below in greater detail.

Delta Tracking before Migration: As mentioned earlier, the common template image is COW-mapped into each VM's memory. The template image could also be pre-loaded into the host's memory (such as into `tmpfs` [19]) to minimize access latency. When a VM tries to write to a COW-mapped page, a write fault is triggered and the hypervisor allocates a new private page into which the original page is copied and the VM can write to. We call these new pages *delta* pages, which are stored separately from the shared template.

TLM uses a dirty page tracking mechanism [7] to keep track of the *delta* pages of the templated VMs before the migration begins [8]. Templated VMs have COW access to the base pages of the template, so a write by a VM instance to a COW-shared page triggers a trap to the hypervisor (a KVM kernel module in QEMU/KVM), which updates a dirty bitmap, and finally grants write permission on the trapped page. Any subsequent writes to the same page by the VM are no longer trapped until the migration starts.

Live Migration: We assume that the base template image is already accessible at the destination over network storage; if not, it could be transferred to the destination once before live migration begins. Initially, at the destination, TLM maps the base template image COW into the memory of each new VM instance. As illustrated in Figure 3, TLM then exclusively transfers the delta pages from the source. These delta pages replace the corresponding COW-mapped pages at the destination, resulting in new memory allocations for the delta pages. In each round of TLM, a user-space per-VM manager process, called QEMU, fetches the latest dirty bitmap from KVM, similar to the conventional pre-copy approach. The delta pages in each round are subsequently marked as read-only and sent to the destination. Eventually, downtime is initiated when a minimal number of delta pages remain. At this point, the VCPUs are paused, the remaining VM states are transferred, and the templated VM instances are then resumed at the destination. Multiple VMs that started from the same base template are migrated concurrently. With TLM, these VMs preserve the same collective memory footprint at the destination as they did the source. When compared to traditional pre-copy migration, TLM has a shorter total migration time because, besides the one-time transfer of base template image, only the *delta* pages need to be transferred to the destination during live migration.

Addressing other forms of inter-VM page sharing during live migration: While the above TLM approach works well in efficiently live migrating multiple templated VMs, we realized that the problem of sharing-awareness in live migration extends beyond just templated VMs. Specifically, TLM does not account for pages shared among VMs due to other memory sharing mechanisms besides templating, such as memory deduplication performed by Kernel Samepage Merging (KSM) [2] in Linux, or simple COW mappings due to process fork and file I/O operations. Figure 4 shows that memory expansion problem during live migration exists even for regular non-templated VMs, though to a lesser extent than templated VMs shown in Figure 2. Out future work will address mechanisms to retain all existing COW sharing during migration irrespective of the underlying memory optimization technique.

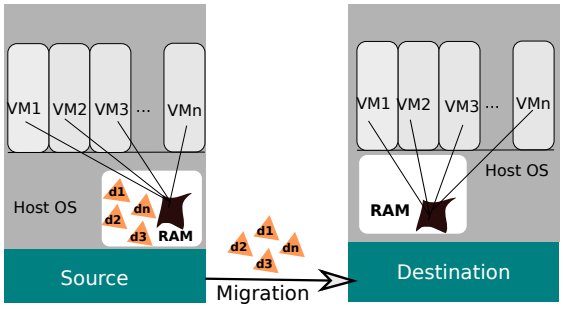

Figure 3: Template-aware migration works by migrating only the *delta* pages during migration. The shared VM template is available to the destination either over a networked storage or transferred ahead of time before migration begins.

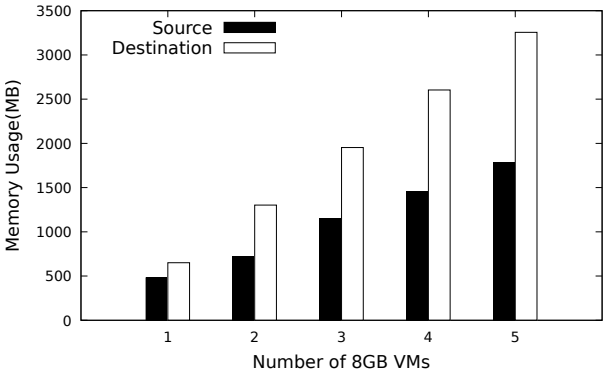

Figure 4: Memory footprint expansion problem in regular non-templated VMs at destination after live migration using generic pre-copy.

4 EVALUATION

We evaluated the performance of TLM against generic pre-copy live migration. Our experimental setup consists of three machines with two Intel Xeon E5-2620 v2 processors and 128GB DRAM. We implemented TLM versions of pre-copy in the KVM/QEMU [11] virtualization platform on Linux. We modified QEMU's default pre-copy algorithms, with no changes to the guest operating system. Each experiment was repeated at least five times on idle VMs to compute average values. Our key performance metrics are as follows:

- **Memory Usage:** The collective memory footprint of the VMs at the source (before migration) and destination (after migration). This is measured using the `free` command and includes the memory used by both the QEMU processes and the VMs.
- **Total Migration Time:** The period which refers to how long it takes to move a virtual machine from one place to another. When moving a single VM using the pre-copy, the total migration time is measured from when the migration starts on source to when the virtual machine is up and running on

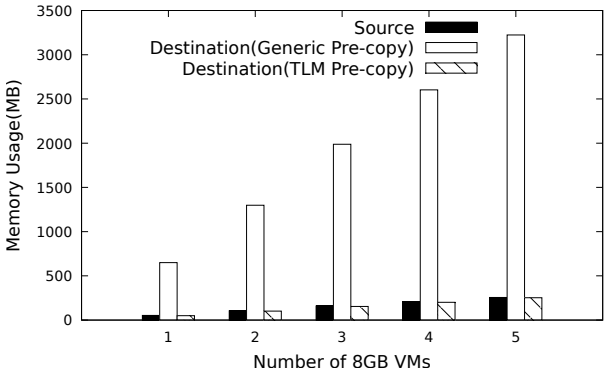

Figure 5: Memory footprint of templated VMs at the source before migration and destination after migration using generic and TLM pre-copy.

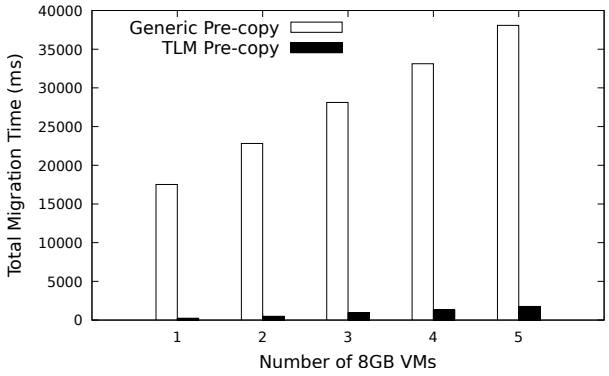

Figure 6: Total migration time of multiple VMs started from the same template and migrated concurrently using generic and TLM pre-copy.

the destination. For moving multiple VMs using pre-copy, the total migration time is calculated from when the first virtual machine's migration begins on the source to when the last virtual machine's migration is finished on destination.

- **Downtime:** The period during which a VM's execution is fully suspended during live migration. In pre-copy, downtime is used to transfer the VM's remaining dirty pages, I/O device and VCPU states to the destination.
- **Network Traffic Overhead:** The total pages transferred during live migration.

Figure 5 shows the memory footprint of templated VMs migrated using generic and TLM pre-copy. The X-axis shows the number of VMs started from the same template, and the Y-axis shows their memory usage before migration at the source and after migration at the destination using `free` command. With increasing number of VMs, the generic pre-copy results in significant expansion of memory footprint at the destination since it is unaware of memory sharing with the underlying template image. Hence it transfers pages that were shared with the template multiple times in addition to *delta* pages. In contrast TLM preserves the original memory footprint of templated VMs at the destination irrespective of the number of VMs started using the template.

Figure 6 shows the total migration time of multiple templated VMs using generic and TLM pre-copy. The X-axis indicates the number of VMs booted from the same template to be migrated concurrently. The Y-axis shows the total migration time. TLM reduces the total migration time up to 94% when considering only the transfer of *delta* pages.

Figure 7 shows that the downtime experienced during live migration of templated VMs is slightly longer (by a few tens of milliseconds) than that of generic VMs, even when the same number of dirty pages are transferred within the downtime window. The X-axis shows the number of pending pages transferred during the downtime. The Y-axis shows the downtime with respect to the pending page size. Upon closer examination of the code, we identified that the increased downtime is influenced by the destination component of live migration. After the final packet arrives at the destination,

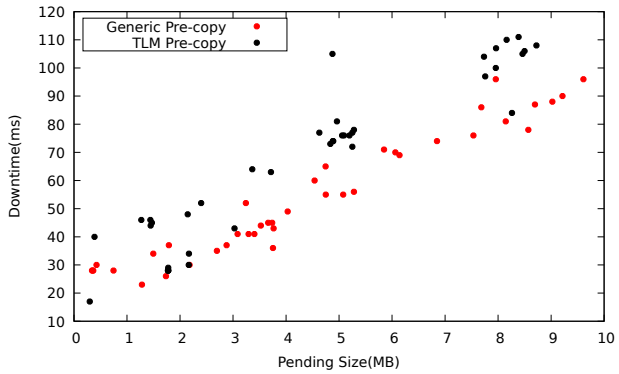

Figure 7: Downtime of multiple templated VMs migrated using generic and TLM pre-copy.

the `vm_start()` function to resume the VM can introduce variable overhead during the migration of both templated and generic VMs. However, in the case of templated VMs, the `vm_start()` function tends to result in higher resumption times more frequently. Addressing the downtime issue involves the VCPU thread invocation which will be addressed in our future work. Nonetheless, the downtime can still be minimized by configuring a smaller pending page count threshold for initiating the downtime phase.

Figure 8 shows the total 4KB pages transferred of multiple templated VMs using generic and TLM pre-copy. The X-axis indicates the number of templated VMs to be migrated concurrently. The Y-axis shows the total pages transferred during live migration. TLM significantly reduces the total pages transferred since it only transfers the delta pages.

5 RELATED WORK

Several VM templating techniques have been developed to efficiently launch multiple lightweight VMs from a common COW template image [12, 18, 23, 25]. VM templating can reduce the memory pressure and instantiation time of VMs on resource constrained

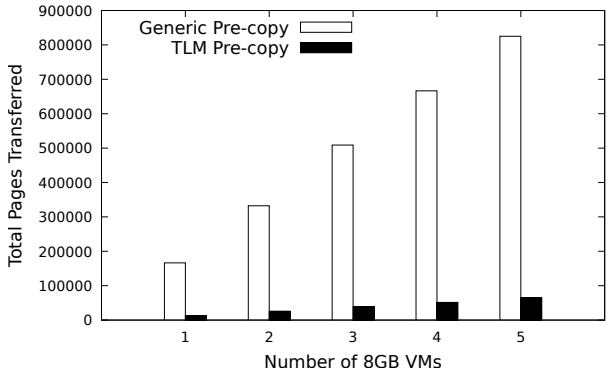

Figure 8: Total pages transferred of multiple VMs started from the same template and migrated concurrently using generic and TLM pre-copy.

edge computing nodes. To the best of our knowledge, these techniques lack support for live migration for templated VMs while maintaining COW sharing at the destination. Our TLM approach addresses this gap. Several studies have also attempted to minimize the migration time of containers or VMs in distributed edge platforms [5, 15, 26]. These efforts have employed techniques like delaying the transfer of writable working sets, using lightweight file systems, applying delta encoding, compressing data, and deduplicating identical pages. However, unlike TLM, these works do not focus on preserving COW page sharings at the destination to prevent an increase in memory footprint.

6 CONCLUSION

In this paper, we addressed the problem that traditional live VM migration techniques do not maintain page sharing among templated VM instances that are migrated to the same destination nodes on edge platforms. This leads to increased memory footprint at a resource constrained destination node, longer migration time, and increased network traffic. We presented the design, implementation, and evaluation of a technique called TLM for pre-copy that retains the templating benefits at the destination after the live migration. Our evaluation of TLM on the QEMU/KVM platform shows that TLM not only avoids memory footprint expansion at the destination but also significantly reduces the migration time and the amount of data transferred. Our future work aims to extend TLM to incorporate other forms of inter-VM page sharing besides those due to templating.

7 ACKNOWLEDGMENTS

We'd like to thank Kevin Cheng and Yongheng Li for their valuable contribution during the implementation of TLM. This work was supported in part by Industrial Technology Research Institute (ITRI), Taiwan.

REFERENCES

[1] [n. d.]. Template-Patch. http://patchwork.ozlabs.org/project/qemu-devel/list/.
[2] Andrea Arcangeli, Izik Eidus, and Chris Wright. 2009. Increasing memory density by using KSM. In *Proc. of the Linux Symposium.*
[3] Wissal Attaoui, Essaid Sabir, Halima Elbiaze, and Mohsen Guizani. 2023. VNF and CNF Placement in 5G: Recent Advances and Future Trends. *IEEE Transactions on Network and Service Management* (2023).
[4] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI).*
[5] Rohit Das and Subhajit Sidhanta. 2021. LIMOCE: Live Migration of Containers in the Edge. In *Proc. of International Symposium on Cluster, Cloud and Grid Computing (CCGrid).*
[6] Umesh Deshpande, Wang Xiaoshuang, and Kartik Gopalan. 2011. Live Gang Migration of Virtual Machines. In *Proc. of High Performance Parallel and Distributed Computing (HPDC).*
[7] fatalerrors. [n. d.]. Dirtybitmap. https://www.fatalerrors.org/a/qemu-synchronization-dirty-pages-principle.html.
[8] Dinuni Fernando, Jonathan Terner, Kartik Gopalan, and Ping Yang. 2019. Live migration ate my VM: Recovering a virtual machine after failure of post-copy live migration. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM).*
[9] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C Snoeren, George Varghese, Geoffrey M Voelker, and Amin Vahdat. 2010. Difference Engine: Harnessing Memory Redundancy in Virtual Machines. In *Communications of the ACM.* ACM New York, NY, USA.
[10] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. 2009. Post-copy live migration of virtual machines. *ACM SIGOPS Operating Systems Review* (2009).
[11] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. kvm: the Linux Virtual Machine Monitor. In *Proc. of the Linux Symposium.*
[12] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M Rumble, Eyal De Lara, Michael Brudno, and Mahadev Satyanarayanan. 2009. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *Proc. of the ACM European Conference on Computer Systems (EuroSys).*
[13] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. 2019. Computation offloading toward edge computing. *Proc. of IEEE International Conference on Cloud Computing* (2019).
[14] Quang-Trung Luu, Sylvaine Kerboeuf, and Michel Kieffer. 2022. Admission control and resource reservation for prioritized requests with guaranteed SLA under uncertainties. *IEEE Transactions on Network and Service Management* (2022).
[15] Lele Ma, Shanhe Yi, Nancy Carter, and Qun Li. 2019. Efficient Live Migration of Edge Services Leveraging Container Layered Storage. *IEEE Transactions on Mobile Computing* (2019).
[16] Pavel Mach and Zdenek Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *Proc. of IEEE International Conference on Communications(ICC)* (2017).
[17] Saad Mubeen, Sara Abbaspour Asadollah, Alessandro Vittorio Papadopoulos, Mohammad Ashjaei, Hongyu Pei-Breivold, and Moris Behnam. 2017. Management of service level agreements for cloud services in IoT: A systematic mapping study. *IEEE access* (2017).
[18] Open Infrastructure Foundation. [n. d.]. Kata Containers. https://katacontainers.io/.
[19] Christoph Rohland. [n. d.]. Tmpfs. https://docs.kernel.org/filesystems/tmpfs.html.
[20] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. 2018. VM Live Migration At Scale. *ACM SIGPLAN Notices* (2018).
[21] Jonathan M. Smith and Gerald Q Maguire Jr. 1988. Effects of copy-on-write memory management on the response time of UNIX fork operations. *Computing Systems* (1988).
[22] Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *Proc. of Middleware.*
[23] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C Snoeren, Geoffrey M Voelker, and Stefan Savage. 2005. Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP).* 148–162.
[24] Jian Wang, Kwame-Lante Wright, and Kartik Gopalan. 2008. XenLoop: A transparent high performance inter-VM network loopback. In *Proc. of High Performance Parallel and Distributed Computing (HPDC).*
[25] Kun Wang, Jia Rao, and Cheng-Zhong Xu. 2011. Rethink the Virtual Machine Template. *ACM SIGPLAN Notices* (2011).
[26] Zhe Zhou, Xintong Li, Xiaoyang Wang, Zheng Liang, Guangyu Sun, and Guojie Luo. 2020. Hardware-assisted Service Live Migration in Resource-limited Edge Computing Systems. In *Proc. of ACM/IEEE Design Automation Conference (DAC).*