Performance Implications of SEV Virtual Machine Live Migration

Jian-Lin Li and Shih-Wei Li

National Taiwan University, Taiwan
b07902103@ntu.edu.tw, shihwei@csie.ntu.edu.tw

Abstract. AMD introduced Secure Encrypted Virtualization (SEV) to support confidential computing. This work evaluated the performance of SEV virtual machine (VM) live migration using the state-of-the-art Linux Kernel Virtual Machine (KVM) implementation for the first time. We found that the live migration of SEV VMs exhibits much higher performance overhead than regular VMs. Instead of adopting a clean-slate approach, we call for retrofitting KVM migration's optimizations, including parallel migration and post-copy, to support SEV VMs. We identified key limitations of SEV that prohibit the proposed extension and demonstrated that KVM's approaches can be incorporated to improve performance effectively by addressing the limitation.

Keywords: Live VM Migration · Confidential Computing · AMD SEV

1 Introduction

To secure user data in virtual machine deployments to the shared cloud environments, confidential VMs (CVMs) have been developed recently to protect VM data from privileged attackers that control the hypervisor. AMD introduced Secure Encrypted Virtualization (SEV) [3,4] support CVMs by encrypting their data. Cloud vendors [10,19] have leveraged SEV to host CVMs for their users.

VM live migration is an essential virtualization feature in which a hypervisor migrates running VMs from one hosted server hardware to another without interrupting the VMs' regular services and operations. Cloud vendors leverage live migration to support VM load balancing across multiple sites. Despite the benefits, the current VM live migration approaches employed by hypervisors do not work for CVMs because hypervisors, by default, cannot access the unencrypted states of CVMs. To address the limitation, SEV has exposed a set of commands to untrusted hypervisors to migrate CVM states while preserving their protection. The hypervisor can use the commands to establish a trusted migration session and to export and import the encrypted VM CPU and memory states. The existing live migration logic in hypervisors like KVM [17] was retrofitted to use these commands to migrate SEV VMs.

This work evaluated the live migration performance of SEV VMs on the state-of-the-art KVM implementation on AMD server hardware for the first time. We found that migrating SEV VMs could suffer many orders of magnitude longer service downtime and total migration time than regular VMs. We found that cryptography operations involved in migrating VM memory pages during the live migration process resulted in significant performance overhead.

Clean-slate approaches that leverage a SEV VM [8,24] to reduce the SEV's live migration overhead have been proposed. However, to our best knowledge, none of the approaches support SEV VM live migration on actual SEV hardware. To address the performance issue, we call for extending KVM's existing optimizations, including post-copy and parallel live migration to support migrating SEV VMs. Retrofitting KVM's existing migration codebase for regular CVMs to support SEV simplifies implementation efforts and enhances maintainability.

We identified that SEV's migration support is incompatible with the performance requirements of parallel live migration and functionality requirements of post-copy, making it impossible for SEV VM to benefit from their efficiency. We proposed updates to AMD SEV to address the limitations in live migration and mimicked the changes in a QEMU/KVM prototype. We evaluated the prototype and demonstrated that adopting KVM's optimization approaches, including post-copy and parallel migration, could significantly improve the live migration performance of SEV VMs.

2 Background

2.1 VM Live Migration

To migrate a virtual machine from a *source* to a *destination* platform, the source hypervisor saves the VM's execution states and transfers them to the destination without stopping the VM. The primary bottleneck in live migration is the transfer of VM memory. Stopping the VM and sending all its state can lead to long VM downtime. KVM, by default, uses pre-copy[9], which transfers all VM memory at the beginning and only transfers the dirty pages in successive iterations. Pre-copy pauses the source VM when the number of dirty pages in an iteration stabilizes within a predefined threshold and enters the *stop-and-copy* phase. The source sends the dirty pages and the VM's CPU and device states to the destination. Parallel migration strategies, like KVM/QEMU's MultiFD [16], use concurrent sending and receiving threads to increase memory transfer bandwidth. In contrast, post-copy [13] transfers a VM's CPU and device states to the destination to resume the VM, then migrates memory upon the destination VM's Nested Page Table (NPT) faults. The destination requests missing page contents from the source. Upon receiving the contents, the destination hypervisor maps the VM's NPT to a newly allocated page containing the retrieved data, resolving the NPT fault.

2.2 Secure Encrypted Virtualization (SEV)

AMD's SEV [3] extends AMD's EPYC CPUs to protect CVMs against privileged attackers who control the hosted hypervisor. AMD extends the memory controller to include an encryption engine that encrypts VM memory with a per-VM AES-128 encryption key (VEK). The VEK is used to encrypt the VM's CPU states in the SEV-ES (Encrypted State) extension.

SEV incorporates a secure firmware that runs on an Arm-based Platform Security Processor (PSP) to support key management. The firmware provides commands to the hypervisor to export and import encrypted VM states in CPU registers and memory, securing CVMs' states when migrating them across sites. Since VM states are encrypted, a malicious hypervisor cannot compromise the safety of the CVM during the live migration process. Since SEV assumes VMs using end-to-end I/O protection, CVMs can reuse regular VM's approach to migrate virtual I/O devices. KVM [17] was extended to leverage SEV's migration commands to support pre-copy. The hypervisors' functionality, including networking and file systems, and the hypervisors' existing logic for migrating regular VMs can be reused to simplify implementation efforts.

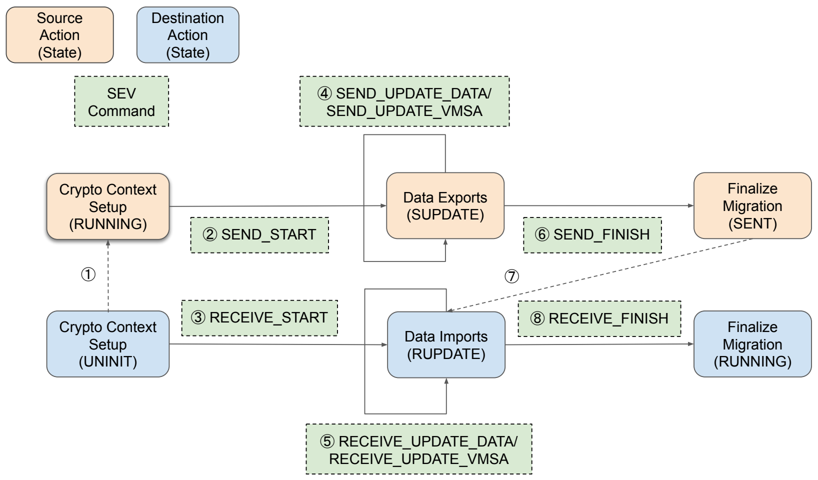
Source Action (State)
Destination Action (State)
SEV Command
④ SEND_UPDATE_DATA/ SEND_UPDATE_VMSA
Crypto Context Setup (RUNNING)
Data Exports (SUPDATE)
Finalize Migration (SENT)
② SEND_START
⑥ SEND_FINISH
①
⑦
Crypto Context Setup (UNINIT)
③ RECEIVE_START
Data Imports (RUPDATE)
⑧ RECEIVE_FINISH
Finalize Migration (RUNNING)
⑤ RECEIVE_UPDATE_DATA/ RECEIVE_UPDATE_VMSA

Fig. 1. Live migration flow and state diagram of SEV with pre-copy

AMD SEV does not migrate a VM's VEK across platforms. Instead, it uses Diffie-Hellman to establish a shared transport key called the transport encryption key (TEK) to encrypt the migrated VM states. Figure 1 shows that the source authenticates the credential sent from the destination at ① then makes `SEND_START` at ② to generate a new cryptographic context that includes the TEK for the migration session. The source sends the context to the destination, and the latter runs `RECEIVE_START` at ③ to derive the TEK.

The SEV firmware uses the per-migration session TEK to encrypt and decrypt the VM states transferred between the migration source and destination. SEV's secure firmware provides the source hypervisor with `SEND_UPDATE_DATA` and `SEND_UPDATE_VMSA`, to export VM memory and CPU states. To export encrypted states for VM live migration, the firmware first decrypts them using the VEK and re-encrypts them using the VM's TEK. During the import, the firmware first uses the TEK to decrypt the encrypted states and re-encrypt them using the destination VM's VEK. The VEK is created by the destination hypervisor for the migrated VM. At ④ in Figure 1, to support pre-copy, the source hypervisor invokes `SEND_UPDATE_DATA` to export dirty pages; when the source pauses the VM, it makes `SEND_UPDATE_VMSA` to export VCPUs. At ⑤, the destination hypervisor uses `RECEIVE_UPDATE_DATA` and `RECEIVE_UPDATE_VMSA` to import the encrypted memory pages and VCPU registers into the SEV VM.

3 Live Migration Performance of SEV VMs

We evaluated the performance of SEV-based CVM live migration on the current KVM implementation using pre-copy. We measured the CVMs running widely used server applications to quantify the performance in practical use cases.

3.1 Experimental Setup

We used mainline Linux kernel v5.19, and an extended QEMU [7] and OVMF [6] for SEV VM live migration; the latter is used for the firmware of guest VMs. We used the AMD EPYC Rome server with a 16-core 64-bit 3GHz processor, 128 GB of RAM, a 480 GB SATA3 SSD, and a dual-port Mellanox ConnectX-5 25Gb NIC. VMs were configured with its standard virtio network and with `cache=none` for its virtual block storage devices [18,21,12]. We ran an unmodified Linux v5.19 in the VM. We compiled the kernels for KVM and VMs using the same configuration. Hosts and VMs ran on Ubuntu 20.04 and Ubuntu 18.04, respectively. In our setup, the source and the destination machine share the VM's virtual disk image via a shared folder on an NFS file system.

For all client-server workloads, clients ran on a third machine separated from the source and destination. All machines were identical. We tested multiprocessor VMs with the following configurations.

- VM with 4 virtual CPUs and 1 GB RAM
- VM with 4 virtual CPUs and 4 GB RAM

Benchmarks. We evaluated the live migration of VMs running the three benchmarks shown in Table 1. The VM config with 1 GB RAM is used to evaluate the Idle and Apache benchmark, while the VM config with 4 GB RAM is employed for testing the Memcached benchmark. We observed that Apache, by default, results in a small working set. Thus, we tested Apache with stress to enlarge the working set. In addition, we tested Memcached, which exhibits a much larger working set than Apache on a VM with more RAM than the Apache config to increase the amount of transferred memory without employing stress. The clients ran natively on Linux for Apache and Memcached experiments and used the full hardware.

Configuring downtime limit. QEMU uses a migration parameter called *downtime limit* to determine when a pre-copy-based live migration enters the last iteration, i.e., the stop-and-copy phase. Therefore, a proper setting for downtime limit can be crucial for the evaluation. For our experiment, every configuration has a fixed workload. Thus, we can first find a proper downtime limit for each setting and then use the respective downtime limit to evaluate performance. We configured a minimal downtime limit for each benchmark that converges migration for different settings. "Convergence" means that the migration is completed.

Table 1. Evaluated benchmarks

Benchmark	**Description**
Idle	Idle VM with 1 GB RAM.
Apache	*Apache* v2.4.29 server running in the VM to host its default web page (about 10KB). The remote client runs the ApacheBench [23] v2.4.54 to establish 100 concurrent connections to the web server.
Apache Stress	The VM uses the same config as **Apache** to serve 100 concurrent connections from a remote client. The VM runs Linux's stress command to generate memory pressure in the background. (stress –vm 1 –vm-bytes 256M)
Memcached	*Memcached* v1.5.6 server configured with 4 threads and 3 GB cache running in the VM, handling set-get operation ratio of 1:1 from a remote YCSB [1] 0.17 client using 8 threads. The cache is filled up before the benchmark starts.

3.2 Performance Results

Table 2. Performance of VMs with pre-copy (in millisecond)

	Idle		Apache		Apache Stress		Memcached	
	Down	Total	Down	Total	Down	Total	Down	Total
KVM-x86	195.8	876.3	214.8	1059.6	305.9	1490.8	574.4	2767.2
SEV	266.2	197393.1	17016.7	346057.5	131660.9	363965.8	241768.4	2905760.4

We present the live migration performance of SEV-based CVM using pre-copy in Table 2. The results are the average of 10 migration sessions. We retrieved the numbers from the source QEMU via the command `info migrate` for the normal VMs (KVM-x86) and CVMs (SEV). We report the following metrics:

– **Downtime:** the time spent in the stop-and-copy phase at the source QEMU. (column *Down*).

– **Total time:** the time the QEMU takes to complete the entire migration process, including data migration and setup time. (column *Total*).

The result shows that the downtime of SEV-based CVM is more than 35% worse than normal VMs for idle VMs. A few dirty pages were sent during the stop-and-copy phase in, so the migration overhead was relatively insignificant. On the other hand, the total time of migrating idle CVMs is significantly longer than normal VMs due to the cryptography operations involved during the migration process. Running an Apache (with memory stress) and Memcached that actively serves network requests increases the dirty pages in the working set; more dirty pages need to be sent during the migration process, resulting in significantly worse performance than idle VMs. The increase in the VM's working set results in more dirty pages that must be migrated. When using pre-copy, SEV-based CVMs required more than 1050× longer total migration time than regular VMs when running Memcached.

4 Optimizing Live Migration Performance of SEV VMs

Previous works [24] introduced clean-slate approaches to support the live migration of SEV VMs. Specifically, these work offload operations from the SEV firmware to optimize performance. They either rely on a mirror SEV VM [24] or an in-CVM Service Module [8] to provide migration service to the hypervisor. However, to our best knowledge, no existing implementation of these approaches supports migrating SEV VMs. Further, whether these approaches support existing optimizations, such as parallel migration, is uncertain. In contrast to the previous efforts, we propose extending KVM's existing migration optimizations, including parallel migration (MultiFD) and post-copy to support SEV VMs — this simplifies implementation efforts and improves maintainability and deployments. We identified that the current live migration commands provided by SEV's secure firmware is incompatible with the performance and functionality requirements of the existing optimization approaches for the parallel live migration with MultiFD and post-copy; therefore, SEV CVMs cannot leverage their performance efficiency.

In this section, we first discuss our attempts to extend KVM/QEMU to support parallel migration (MultiFD). Further, we substantiate key limitations of the SEV firmware that make it incompatible with post-copy in Section 4.2. We detail SEV's limitations that result in compatibility issues concerning performance and functionality. We then propose extensions for the secure firmware to address the issues.

4.1 Extending KVM/QEMU FOR MultiFD Support of SEV VM

We extended the current KVM/QEMU implementation for SEV, which implements the pre-copy approach to support MultiFD. The implementation already makes commands to setup a migration session (see Figure 1) to initiate a migration session and perform transport key exchange. We added additional export and import SEV commands in the sending and receiving MultiFD threads. We modified the source hypervisor to make the command `SEND_UPDATE_DATA` before placing VM memory in the sending queue. The destination hypervisor is extended to issue the command `RECEIVE_UPDATE_DATA` to decrypt the migrated VM memory. We were unable to extend QEMU/KVM to support post-copy.

We evaluated the live migration performance of our MultiFD implementation for SEV VMs running the three benchmarks from Table 1. We found that MultiFD resulted in modest performance improvement for SEV CVMs. The migration downtime and total time reduction was no more than 5% in all cases. We detail why the current QEMU/KVM implementation for SEV fails to benefit from the MultiFD in Section 4.2.

4.2 Compatibility Issues

Performance Incompatibility with MultiFD. AMD's secure firmware poses limitations that make it unable to export and import SEV VM's memory in multiple execution threads in parallel based on MultiFD. The AMD SEV API Specification [3] requires that no SEV firmware commands can be processed before the last command receives a response. This prevents commands from executing in parallel. Given that AMD does not disclose the hardware design of PSP, we hypothesize this is likely because the SEV firmware runs on a single-processor PSP. This requirement prevents live migration benefits from MultiFD. Although each sending thread on the source can issue the `SEND_UPDATE_DATA` command in parallel from the hypervisor's point of view, those commands can only be processed by the SEV firmware sequentially. Hence the costly crypto operations involved in handling the command cannot be performed in parallel.

Functionality Incompatibility with Post-Copy. SEV's firmware ensures that the hypervisor calls SEV migration commands in proper order. The invocation against the migration of a specific VM must match the expected VM `GSTATE`. Only certain commands are allowed to be executed in one GSTATE. As shown in Figure 1, the SEV firmware maintains a guest state (`GSTATE`) machine for each guest to track the VM's execution state during its lifetime. After the `SEND_START` command is called, the source transitions from the RUNNING state to the SUPDATE state; similarly, the destination moves from the UINIT state to the RUPDATE state after it calls the `RECEIVE_START` command. After all VM state is sent to the destination, the source issues `SEND_FINISH` at ⑥. This transitions the VM to the SENT state as shown in Figure 1. The source notifies the destination of the end of data export at ⑦, which the destination subsequently invokes `RECEIVE_FINISH` at ⑧ to finalize the migration session. Executing the command transitions the VM state to RUNNING.

Such restrictions render post-copy incompatible with the current AMD SEV's migration commands. As discussed in Section 2.1, post-copy requires the destination hypervisor to resume the VM and then migrate memory contents on demand when the VM later page faults. The SEV firmware only permits the invocation of the RECEIVE_UPDATE_DATA command when the destination

VM's GSTATE is RUPDATE. However, to run a migrated VM, the firmware requires that the destination SEV VM's GSTATE be set as the RUNNING state. This incompatibility prevents the destination hypervisor from using the SEV's migration import commands `RECEIVE_UPDATE_DATA` or `RECEIVE_UPDATE_VMSA` to import the encrypted VM data transferred from the source. The migration commands are essential for migrating the SEV VM's state. The SEV firmware provides no other command to the hypervisor to decrypt the received page contents using the migration transport key, TEK.

Address SEV's Incompatibility. We proposed the following extensions to SEV to lift the restrictions.

- **MultiFD.** Process SEV migration commands concurrently to increase the migration throughput. The extension requires hardware and firmware multiprocessor support on PSP.
- **Post-copy.** To (1) support importing VM memory when the VM's GSTATE is RUNNING; (2) support exporting VM memory from the source VM after it switches to the SENT state. Specifically, the secure firmware could add a new command that can be executed after the VM resumes at the destination; the command should match the functionality of RECEIVE_UPDATE_DATA. The firmware could add another command to deliver the functionality of SEND_UPDATE_DATA for the source hypervisor to export VM memory states after the VM is paused.

4.3 Evaluation of SEV VM Live Migration with Optimizations

We mimicked SEV's performance overhead on the codebase in QEMU/KVM for regular VM migration to demonstrate the potential performance improvement the SEV VM should achieve when incorporating the live migration optimizations. In QEMU, we simulate the proposal by building Pseudo-SEV QEMU, which lifts the compatibility issues incurred by AMD SEV's secure firmware. Pseudo-SEV mimics the cost of SEV commands on top of a mainline KVM. Pseudo-SEV QEMU is based on QEMU 6.0.91. We ported AMD's updates to QEMU for SEV to Pseudo-SEV QEMU. AMD's QEMU implementation makes the ioctl system calls to KVM. Each SEV's ioctl makes a respective SEV command to support the VM live migration process shown in Figure 1. Pseudo-SEV QEMU supports the same migration logic as AMD's QEMU but does not make the actual SEV ioctls. Instead, we instrumented Pseudo-SEV QEMU to replace the ioctls with respected functions that mimic the exact cost shown in Figure 1 as the intended SEV ioctl requires. In the worst case, the cost difference between the real and the pseudo ioctls are bounded by 5%. To implement pseudo-SEV QEMU, we added and modified 624 LOC to QEMU.

Pseudo-SEV QEMU supports pre-copy, post-copy, and MultiFD migration. SEV's functions for pre-copy for the setup phase and VM resume are reused to mimic the cost of starting and finishing a migration session. Additionally, we instrumented the existing code paths for post-copy and MultiFD in pseudo-SEV QEMU in the migration Setup phase and for places where VM states import and export were required to mimic the cost when migrating a SEV-based CVM. By instrumenting existing code paths, we removed the limitations of the firmware API. For MultiFD, we mimicked the migration cost in each respective sending and receiving thread to do the memory export and import in parallel.

Table 3. Performance of VMs on pseudo-SEV (Normalized to pre-copy)

	MultiFD-2		MultiFD-4		MultiFD-8		post-copy	
	Down	Total	Down	Total	Down	Total	Down	Total
Idle	0.93	1.01	0.92	0.68	0.95	0.40	0.99	0.23
Apache	0.39	0.55	0.16	0.29	0.07	0.15	0.01	0.53
Apache Stress	0.57	0.52	0.23	0.31	0.11	0.17	0.0025	0.51
Memcached	0.58	0.69	0.37	0.49	0.28	0.32	0.001	0.51

We evaluated the benchmarks from Table 1 with the same configuration used in Section 3.1 to investigate the efficacy of SEV VM employing the VM live migration optimizations. Table 3 presents the performance of SEV and pseudo-SEV. The results are normalized to the pre-copy-based implementation with the same setting to present the efficacy of adopting the optimization approaches. For idle SEV VM, we normalized the total time and downtime of MultiFD and post-copy to the results of SEV's default pre-copy approach. Table 3 shows that SEV VMs employing KVM's existing optimizations could significantly improve VM migration performance. In the best case, MultiFD can reduce the total time by more than 80% and the downtime by 90% (MultiFD-8 for Apache and Apache Stress). The results show that incorporating MultiFD to parallelize cryptographic operations involved in migrating VM memory effectively reduced the downtime and total time. The post-copy approach can reduce more than 1000× in downtime, as it avoids transferring VM memory in the stop-and-copy phase. Post-copy only transfers the VM pages upon the occurrence of Nested Page Table faults from the destination VM. Unlike pre-copy, post-copy avoids transferring dirty VM pages multiple times, thus also achieving a much shorter total time (50% less) than pre-copy.

While the post-copy approach can effectively reduce the downtime reported by the source QEMU, it did not cover the cost at the destination platform. Typically, when migrating confidential VMs, the destination hypervisor must perform crypto operations to decrypt the migrated VM states for imports and handle NPT page faults if post-copy is adopted. The cost could be nontrivial. To evaluate the VM migration overhead at destination platforms, we measured the Apache server's throughput at various sample points throughout the VM migration process. The results on KVM and pseudo-SEV were shown in Figure 2.

Figure 2 shows that the pseudo-SEV VM's unresponsive time (i.e., requests per second hits zero) is much higher than the regular VM's because the crypto operations used for VM states encryption and decryption throttled the migra-

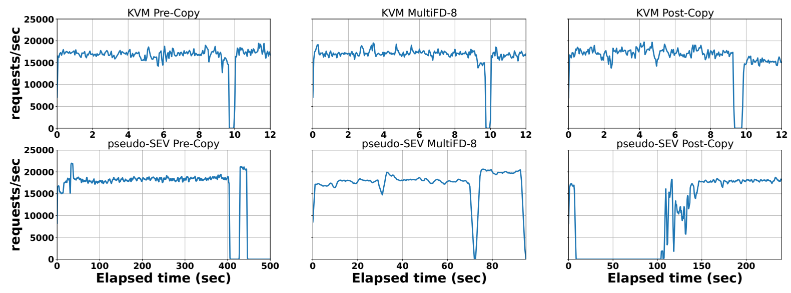

Fig. 2. Apache Throughput during VM Live Migration

tion throughput. Compared to pre-copy, the unresponsive time is reduced when MultiFD is adopted, as MultiFD increases the migration throughput. The VM's unresponsive time is much longer than the downtime showed in Table 2 and Table 3. As discussed earlier, the source QEMU reported the time taken during the stop-and-copy phase as downtime. When post-copy is employed, the cost of handling NPT page faults at the destination platform communicating with the source is non-negligible. Such cost was not counted in QEMU's reported downtime but appeared in the VM's unresponsive time.

5 Related Work

VM Live Migration. In addition to pre-copy[9], parallel migration [20], and post-copy [13], Svärd et al. [22] proposed to compress the transferred memory pages to boost the migration throughput and thus reduce downtime, while Hacking et al. [11] proposed compressing dirty pages sent during live migration iterations. Abe et al. [2] paravirtualized the guest kernel to provide the VM's working set to the hypervisor for transferring in advance to reduce the performance impact resulting from the post-copy transfer. Pre-paging and dynamic ballooning [13] were also proposed to optimize migration downtime. These methods could be employed to optimize the live migration performance of SEV VMs.

Confidential Virtual Machines. Intel TDX [14] introduced hardware extensions to secure the data of its CVMs, Trusted Domains (TDs). It runs a secure firmware module in an isolated CPU operation context called the Secure-Arbitration Mode (SEAM) mode. The module deprivileges the hypervisor, performs VM enters, and interposes VM exits to protect VM data. Intel TDX leverages hardware cryptographic support to protect VM memory. It uses the TME-MK engine to encrypt VM memory and protect its integrity. Intel TDX [15] proposes an auxiliary CVM, MigTD, and a privileged TDX module to support CVM live migration. Unlike AMD SEV, Intel TDX supports post-copy and parallel live migration. In addition, Arm has also introduced CCA [5] for the Arm v9.2 architecture to support confidential VMs called the Realms. CCA does not support live migration. Hardware with Intel TDX and CCA has not been publicly available at the time of writing. We were unable to evaluate their CVMs' live migration performance. Should the hardware be available, we plan to evaluate the live migration performance of TDX-based CVMs. Further, we aim to explore designing live migration support for Arm CCA.

6 Conclusion

We evaluated the live migration performance of SEV-based confidential VMs in this work for the first time. We found that migrating SEV VMs on KVM incurs a significant performance slowdown. We proposed extending the KVM implementation for SEV to address the performance issue and support the existing optimization approaches, such as parallel and post-copy migration. We identified SEV's limitations that make it incompatible with these approaches. We proposed extensions to SEV to address the limitations. We mimicked the intended changes in a KVM implementation and showed that the live migration performance of SEV VMs could be substantially improved when the optimizations are adopted. We hope the paper's findings will help to improve SEV's ecosystem in cloud deployments. We call to address the incompatibility issues in AMD SEV and facilitate the adoption of KVM's existing optimization approaches.

Acknowledgement

We thank the anonymous reviewers for their insightful feedback. Kaiwen Xue helped with writing the early drafts of the paper. This research was supported in part by the National Science and Technology Council of Taiwan under research grants 111-2628-E-002-022- and 112-2628-E-002-027-.

References

1. Yahoo! Cloud Serving Benchmark. `https://github.com/brianfrankcooper/YCSB`
2. Abe, Y., Geambasu, R., Joshi, K., Satyanarayanan, M.: Urgent virtual machine eviction with enlightened post-copy. SIGPLAN Not. **51**(7), 51–64 (mar 2016). `https://doi.org/10.1145/3007611.2892252`, `https://doi.org/10.1145/3007611.2892252`
3. Advanced Micro Devices: Secure Encrypted Virtualization API Version 0.24. `https://www.amd.com/system/files/TechDocs/55766_SEV-KM_API_Specification.pdf` (Apr 2020)
4. Advanced Micro Devices: SEV Secure Nested Paging Firmware ABI Specification. `https://www.amd.com/system/files/TechDocs/56860.pdf` (Nov 2022)
5. ARM Ltd.: Introducing Arm Confidential Compute Architecture Version 1 (May 2022), `https://developer.arm.com/documentation/den0125/0100/What-is-Arm-CCA-`
6. Ashish Kalra: Retrofitted EDK2. `https://github.com/ashkalra/edk2-1/tree/sev_live_migration_v5_11` (Sep 2021)
7. Ashish Kalra: Retrofitted QEMU. `https://github.com/AMDESE/qemu/tree/sev_live_migration_v4_1` (Sep 2021)

8. Carlos Bilbao: The Linux SVSM project (Jan 2023), https://lwn.net/Articles/921266/
9. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: 2nd Symposium on Networked Systems Design & Implementation (NSDI 05). USENIX Association, Boston, MA (May 2005)
10. Google Inc.: Confidential VM overview, https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview
11. Hacking, S., Hudzia, B.: Improving the live migration process of large enterprise applications. In: Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing. p. 51–58. VTDC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1555336.1555346, https://doi.org/10.1145/1555336.1555346
12. Hajnoczi, S.: An Updated Overview of the QEMU Storage Stack. https://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_hajnoczi.pdf (Jun 2011)
13. Hines, M.R., Gopalan, K.: 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. p. 51–60. VEE '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1508293.1508301, https://doi.org/10.1145/1508293.1508301
14. Intel Corporation: Intel® Trust Domain Extensions (Intel® TDX). https://cdrdv2.intel.com/v1/dl/getContent/733582 (Feb 2023)
15. Intel Corporation: Trust Domain Extension (TDX) Migration TD Design Guide. https://cdrdv2.intel.com/v1/dl/getContent/733580 (Mar 2023)
16. Juan Quintela: QEMU/Migration-Multiple-fds, https://wiki.qemu.org/Features/Migration-Multiple-fds
17. Kalra, A.: Add amd sev guest live migration support. https://lwn.net/Articles/851648/, accessed: 2023-05-14
18. KVM Contributors: Tuning KVM. http://www.linux-kvm.org/page/Tuning_KVM (May 2015)
19. Microsoft Corporation: AMD and Azure, https://azure.microsoft.com/en-us/partners/directory/amd-corporation/
20. Song, X., Shi, J., Liu, R., Yang, J., Chen, H.: Parallelizing live migration of virtual machines. In: Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. p. 85–96. VEE '13 (2013)
21. SUSE: Performance Implications of Cache Modes. https://www.suse.com/documentation/sles11/book_kvm/data/sect1_3_chapter_book_kvm.html (Sep 2016)
22. Svärd, P., Hudzia, B., Tordsson, J., Elmroth, E.: 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. p. 111–120. VEE '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1952682.1952698, https://doi.org/10.1145/1952682.1952698
23. The Apache Software Foundation: ab - Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/2.4/programs/ab.html (Apr 2015)
24. Tobin Feldman-Fitzthum, Dov Murik: Secure Live Migration of Encrypted VMs (Sep 2021), https://research.ibm.com/publications/secure-live-migration-of-encrypted-vms