Efficient and Flexible Checkpoint/Restore of Split-memory Virtual Machines

doi: https://doi.org/10.1109/ICCI51257.2020.9247679

Efficient and Flexible Checkpoint/Restore of Split-memory Virtual Machines

Tokito Murata
Kyushu Institute of Technology
tokito@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology
kourai@ksl.ci.kyutech.ac.jp

***Abstract*—Recently, clouds provide virtual machines (VMs) with a large amount of memory for big data analysis. For easier migration of such VMs, split migration divides the memory of a VM into several pieces and transfers them to multiple hosts. Since the migrated VM called a *split-memory VM* needs to exchange memory data between the hosts, it is inherently subject to host and network failures. As a countermeasure, a checkpoint/restore mechanism has been used to periodically save the state of a VM, but the traditional mechanism is not suitable for split-memory VMs. It has to move a large amount of memory data between hosts during checkpointing and can just restores a normal VM on one host. This paper proposes *D-CRES* for efficient and flexible checkpoint/restore of split-memory VMs. D-CRES achieves fast checkpointing by saving the memory of a VM in parallel at all the hosts without moving memory data. For live checkpointing, it consistently saves the memory of a running VM by considering memory data exchanged by the VM itself. In addition, it enables a split-memory VM to be restored in parallel at multiple hosts. We have implemented checkpoint/restore of D-CRES in KVM and showed that the performance was up to 5.4 times higher than that of using the traditional mechanism.**

1. Introduction

As Infrastructure-as-a-Service (IaaS) clouds are widely used, they also provide VMs with a large amount of memory. For example, Amazon EC2 provides VMs with 24 TB of memory [1]. Such large-memory VMs are used for big data analysis [2], [3] and in-memory database [4], [5]. When host maintenance or load balancing is performed in clouds, it is necessary to migrate VMs from a target host to other hosts and continue their execution. Since VM migration transfers the entire memory of a VM, it requires a destination host with sufficient free memory. However, it is neither cost-efficient nor flexible to always preserve such large hosts as the destinations of occasional VM migration.

To make the migration of such VMs easier, split migration [6] has been proposed. It divides the memory of a large-memory VM into several pieces and transfers them to multiple smaller hosts. Specifically, it transfers the state of the VM core and likely accessed memory to one *main host*. The rest of the memory is transferred to *sub-hosts*. After split migration, the main host runs the migrated VM, while the sub-hosts provide the main host with part of the memory of the VM. Such a VM running across multiple hosts is called a *split-memory VM*. It exchanges memory data between hosts using *remote paging*. When a split-memory VM requires the memory existing in a sub-host, the memory data is transferred to the main host. At the same time, unlikely accessed memory data is transferred to that sub-host.

Since a split-memory VM needs network communication among multiple hosts, it is inherently subject to host and network failures, compared with a normal VM running on one host. As a countermeasure against such failures, a *checkpoint/restore* mechanism has been used. It periodically saves the state of a VM as a checkpoint and restores it from the latest checkpoint when a failure occurs. However, two issues arise when the traditional mechanism is applied to split-memory VMs. First, remote paging is frequently caused by checkpointing and degrades checkpointing performance. This is because the memory existing in the sub-hosts is moved to the main host and is saved. Second, a VM can be restored on only one host. Therefore, a large host with sufficient free memory is always required for restoring a VM.

To address these issues, this paper proposes *D-CRES*, which enables efficient and flexible checkpoint/restore of split-memory VMs. Upon checkpointing, D-CRES independently saves the memory of a split-memory VM at each host. It completely avoids remote paging between the main host and sub-hosts for checkpointing. It also achieves fast checkpointing by saving memory data in parallel at all the hosts. In addition, it supports live checkpointing, which saves the state of a VM without stopping it. It considers remote paging caused by a running VM itself during checkpointing and consistently saves the memory of the VM. Upon restoring, D-CRES enables a split-memory VM to be restored across multiple hosts. It restores the state of a VM in parallel from the checkpoints taken at multiple hosts. At this time, it is also possible to change how to split the memory of a VM according to available hosts.

We have implemented the checkpoint/restore mechanism of D-CRES in KVM. D-CRES saves the memory of a VM on the basis of the network page table and the page sub-tables, both of which are used for running a split-memory VM. Upon a failure, it restores not only the memory but also these tables. For live checkpointing, it minimizes the total size of checkpoint files by overwriting updated memory data, instead of appending them as done in the traditional mechanism. Experimental results show that the performance of checkpoint/restore in D-CRES was significantly improved, compared with when the traditional mechanism was applied to a split-memory VM.

978-1-5386-5541-2/18/$31.00 ©2020 IEEE

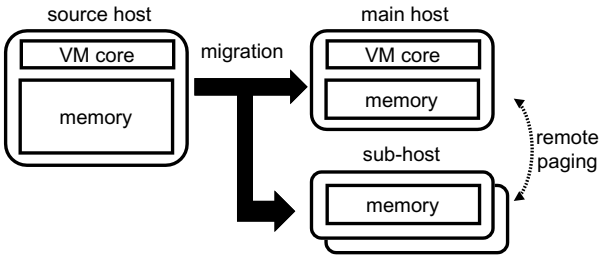

Figure 1. Split migration and remote paging.

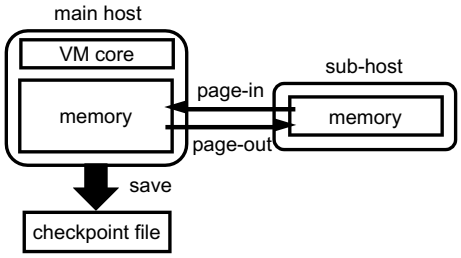

Figure 2. Checkpointing a split-memory VM using the traditional mechanism.

The rest of this paper is organized as follows. Section 2 describes issues of checkpoint/restore of a split-memory VM. Section 3 proposes D-CRES for efficient and flexible checkpoint/restore of a split-memory VM and Section 4 describes its implementation. Section 5 shows experimental results of the performance of checkpoint/restore. Section 6 describes related work and Section 7 concludes this paper.

2. Checkpoint/Restore of Split-memory VMs

IaaS clouds migrate VMs in various reasons, e.g., host maintenance and load balancing. To accommodate the entire memory of a migrated VM, the destination host has to have sufficient free memory. Since the memory size of a VM is increasing recently [1], this constraint on VM migration and the cost for always preserving such a destination host are being concerns of cloud providers. Split migration [6] enables a large-memory VM to be divided into several pieces and be transferred to multiple small hosts, as shown in Fig. 1. It transfers the state of the VM core such as virtual CPUs and devices and the memory likely accessed in the near future to one main host. It transfers the memory that cannot be accommodated in the main host to sub-hosts. The future memory access of a migrated VM is predicted on the basis of LRU.

After split migration, the main host runs the migrated VM, while the sub-hosts provide part of the memory of the VM to the main host. Such a VM across multiple hosts is called a *split-memory VM*. It exchanges memory data between hosts by *remote paging* as needed. When it requires the memory existing in a sub-host, the memory is *paged in* to the main host via the network. At the same time, unlikely accessed memory in the main host is *paged out* to that sub-host to balance the amount of memory between the two hosts. As such, remote paging enables a large-memory VM to run on small hosts with insufficient free memory.

Since a split-memory VM needs network communication among hosts, it is more likely to be affected by host and network failures, compared with a normal VM running on one host. For example, if one of the sub-hosts suffers from a failure and stops, part of the memory of a split-memory VM is lost and the execution of the VM cannot continue. If a network failure occurs, a split-memory VM cannot be executed until the network is recovered because the memory data required by remote paging cannot be transferred to the main host.

Traditionally, *checkpoint/restore* of VMs has been used as a countermeasure against such failures. The checkpoint mechanism periodically saves the state of a VM to a file as a backup called a checkpoint. The state of a VM consists of those of virtual CPUs and devices, memory data, disk data, and so on. When a failure occurs, the restore mechanism creates a new VM on another host and restores the saved state of a VM from the latest checkpoint. Compared with booting a new VM from scratch, this mechanism can shorten the recovery time and minimize lost data due to failures.

However, two issues arise if this traditional checkpoint/restore is applied to split-memory VMs as it is. The first issue is that remote paging is frequently caused by checkpointing and largely degrades checkpointing performance. The memory existing in the main host is saved at the main host as usual, but that existing in a sub-host is first paged in to the main host and is then saved, as shown in Fig. 2. This is because the state of a VM is saved only at one host, where the checkpoint mechanism runs. At the same time, unlikely accessed memory in the main host is paged out to that sub-host to prepare free memory for the paged-in memory. If the paged-out memory is saved later, it is paged in to the main host again.

The second issue is that the traditional checkpoint/restore cannot restore a VM as a split-memory VM. It always restores a normal VM running on one host even if a split-memory VM is checkpointed. As mentioned above, the traditional checkpoint mechanism saves the entire memory of a VM to one checkpoint file at the main host. This means that the checkpoint file cannot be distinguished from that for a normal VM. As a result, the traditional restore mechanism requires one large host with sufficient free memory to accommodate the entire memory of a VM. This is often difficult for a large-memory VM.

3. D-CRES

This paper proposes *D-CRES*, which enables efficient and flexible checkpoint/restore of split-memory VMs. For checkpointing, D-CRES saves the memory of a split-memory VM in parallel at multiple hosts, as shown in Fig. 3. This prevents remote paging between the main host and sub-hosts from being caused by checkpointing and achieves fast checkpointing of a split-memory VM. Upon a host or network failure, D-CRES restores a split-memory VM from the latest checkpoint in parallel at multiple hosts. For flexibility, it can use an arbitrary set of hosts whose free memory is different from that on checkpointing.

When D-CRES takes a checkpoint of a split-memory VM, it saves part of the memory of the VM existing in the main host to a checkpoint file at the main host.

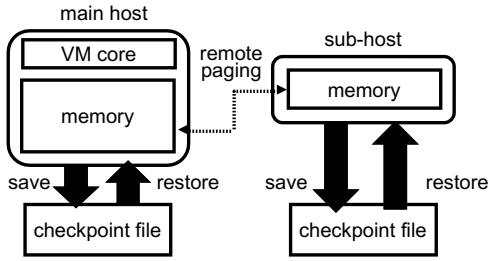

Figure 3. Checkpoint/restore in D-CRES.

It also saves the state of the VM core such as virtual CPUs and devices. In addition, the snapshot of the virtual disk is taken like that of a normal VM. At each sub-host, on the other hand, D-CRES saves only part of the memory of the VM existing in that sub-host to a different checkpoint file. Checkpointing is completed when all the states are saved at the main host and all the sub-hosts. In preparation for host or network failures, the saved checkpoint is transferred to redundant network storage.

D-CRES consistently performs live checkpointing, which can take a checkpoint of a split-memory VM without stopping the VM. It saves the memory of a VM while the VM is running. Then, it additionally saves the memory updated by the VM itself during checkpointing. When the amount of updated memory becomes small enough, D-CRES temporarily stops the VM and saves the remaining memory and the state of the VM core. During live checkpointing, it saves the memory without excess or deficiency by considering remote paging between hosts, which is still caused by a running VM. It saves the memory paged in to the main host at the main host. If that memory has been already saved at a sub-host, it is removed from the checkpoint file created at that host. Similarly, D-CRES saves the memory paged out to a sub-host at that sub-host. If that memory has been already saved at the main host, it is removed from the checkpoint file created at the main host.

When a failure occurs, D-CRES first finds new main host and sub-hosts that have sufficient free memory for a restored VM in total. Then, it creates a new VM on the main host and restores a split-memory VM using the latest checkpoint files. It restores part of the memory of a VM at each host. At the main host, the states of the VM core and the virtual disk are also restored. When D-CRES completes the restoration of the state of a VM at all the hosts, it establishes a network connection for remote paging between the main host and each sub-host. Finally, it restarts the execution of the split-memory VM.

Upon restoring a split-memory VM, D-CRES can change how to split the memory of the VM into new hosts. This makes it possible to restore a split-memory VM flexibly even if there is no available set of hosts that have exactly the same free memory as those on checkpointing. As an extreme example, if there is one host with sufficient free memory, D-CRES can restore a VM as a normal VM running only on that host. In general, it re-splits the memory of a VM on the basis of the saved memory access history so that likely accessed memory is placed in the main host as much as possible.

4. Implementation

We have implemented D-CRES in KVM with support for split migration and remote paging.

4.1. Memory Management of a Split-memory VM

The memory of a split-memory VM is managed differently at the main host and sub-hosts. At the main host, QEMU-KVM runs and manages part of the memory of a VM using the *network page table*. The network page table is a table for searching for a host ID from a memory page number. QEMU-KVM obtains information in which host the specified memory page exists by looking up this table. At each sub-host, a memory server runs and manages part of the memory of a VM using the *page sub-table*. The page sub-table is a table for searching for memory data from a memory page number.

When QEMU-KVM in the main host performs a page-in from a sub-host, it first searches the network page table for the sub-host where the page exists and sends a page-in request to that sub-host. The memory server in the sub-host searches the page sub-table for the memory data of the requested page and sends the data back to the main host. At this time, it removes the found entry from the page sub-table. When QEMU-KVM receives that data, it inserts a new entry to the network page table. After that, QEMU-KVM selects the least recently used page and sends a page-out request to that sub-host. Then, it modifies the corresponding entry in the network page table. The memory server inserts a new entry for the received page to the page sub-table.

4.2. Checkpointing at the Main Host

When QEMU-KVM in the main host receives the checkpoint command, it first forwards that command to all the sub-hosts. Then, it creates a checkpoint file for saving the state of a VM and saves the memory data on the basis of the network page table. For a page existing in the main host, it saves the memory address, data, and access history. For a page existing in a sub-host, it saves only the memory address and the sub-host ID. When it completes to save all the pages and the other states of the VM, it waits for the completion of checkpointing at the sub-hosts and then finishes checkpointing.

The original QEMU-KVM supports live checkpointing of normal VMs, which takes a checkpoint without stopping a VM. However, that implementation is not suitable for large split-memory VMs. First, the memory data updated during checkpointing is appended to the end of a checkpoint file. Since a large amount of memory tends to be updated in a large-memory VM, this increases the total size of checkpoint files. Second, a taken checkpoint can be inconsistent because traditional live checkpointing does not consider remote paging. When memory is paged in to the main host during checkpointing, it may not be saved at the main host correctly. Even when memory is paged out and no longer exists in the main host, it is not removed from a checkpoint file.

To address these issues, D-CRES saves the memory of a VM to an independent sparse file called a *memory file*, as shown in Fig. 4. Since the offset in a memory file

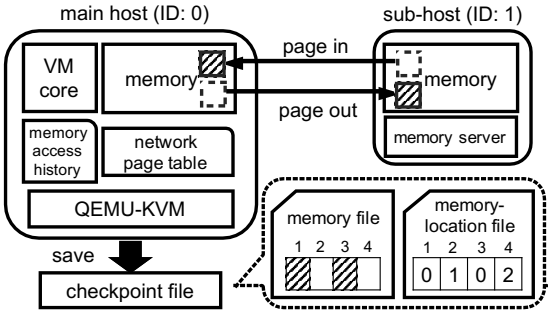

Figure 4. Live checkpointing at the main host.

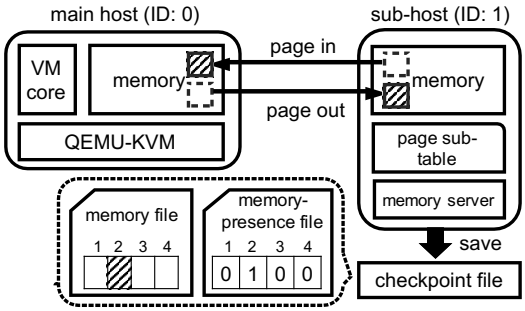

Figure 5. Live checkpointing at a sub-host.

corresponds to the memory address one-to-one, D-CRES can easily overwrite memory data in this file by updated one. Therefore, the maximum size of this file is limited to the memory size of a VM even if updated memory is saved. In a sparse file, the file block in which data is not saved is empty, which is called a *hole*. As a result, the total of the real sizes of the checkpoint files saved in all the hosts is equal to the memory size of a VM.

When a page-in is caused by a split-memory VM during checkpointing, QEMU-KVM sets the corresponding bit to the *dirty bitmap*, which is used for identifying memory to be saved. Later, it saves the paged-in page to the corresponding block of the memory file. In contrast, when a page-out is caused, QEMU-KVM makes the corresponding block of the memory file a hole and removes memory data from the file. In addition, the information on the host where each page exists is stored in a separate file called a *memory-location file*. This file is also updated whenever remote paging occurs.

When the number of remaining pages to be saved becomes small enough, QEMU-KVM sends synchronization messages to the sub-host. Then, it waits until all the sub-hosts reach the same state and send back the responses. When all the hosts are synchronized, QEMU-KVM waits for the completion of pending remote paging, sends a finalization message to each sub-host, and pauses the VM. After that, it saves the remaining memory and the memory access history. In addition, it creates a snapshot of the virtual disk efficiently using the feature of the QCOW2 disk format. Finally, it saves the states of virtual CPUs and devices. Then, it waits for completion messages from all the sub-hosts and restarts the VM.

4.3. Checkpointing at a Sub-host

When the memory server receives the checkpoint command from the main host, it creates a checkpoint file for saving part of the memory of a VM. On the basis of the page sub-table, it saves the address and data of the memory existing in the sub-host. To support live checkpointing, the memory server also uses a memory file, as shown in Fig. 5. When it receives a page-out request during checkpointing, it saves the paged-out page to the corresponding block of the memory file. In contrast, when it receives a page-in request, it makes the corresponding block of the memory file a hole. Unlike at the main host, a VM does not directly access the memory existing in sub-hosts. Therefore, the memory server updates the memory file only when remote paging occurs. In addition, it saves information on whether pages exist in the sub-host to a separate file called a *memory-presence* file.

When the memory server receives a synchronization message from the main host, it returns a response if the remaining memory to be saved becomes small enough. Then, it waits for a finalization message from the main host and then saves the remaining memory. When it completes to save all the memory, it returns a completion message to the main host.

4.4. Restoring a Split-memory VM

D-CRES first transfers the obtained checkpoint files to a new set of hosts and then sends the restore command to QEMU-KVM at the new main host. When QEMU-KVM receives this command, it forwards the command to all of the new sub-hosts. Next, QEMU-KVM restores part of the memory at the new main host from the memory file saved at the old main host. In addition, it restores the network page table from the memory-location file. Then, it restores the memory access history, the states of virtual CPUs and devices, and the state of the virtual disk. When it receives completion messages from all the sub-hosts, it establishes a network connection for remote paging between the new main host and each new sub-host and restarts the restored split-memory VM.

When the memory server at a new sub-host receives the restore command from the main host, it restores part of the memory that existed in one of the old sub-hosts from the memory file saved at that old sub-host. In addition, it restores the page sub-table from the memory-presence file. Then, it sends a completion message to the main host. After it establishes a network connection with the main host, it waits for requests for remote paging.

If D-CRES relocates the memory of a split-memory VM according to the amount of free memory of newly available hosts, it moves memory data among the memory files before restoration. In addition, it modifies the memory-location file and the memory-presence files. Currently, D-CRES supports only the relocation of the entire memory from all the sub-hosts to the main host.

5. Experiments

We conducted several experiments to examine the performance of checkpoint/restore of a split-memory VM using D-CRES. For a main host and a sub-host, we used

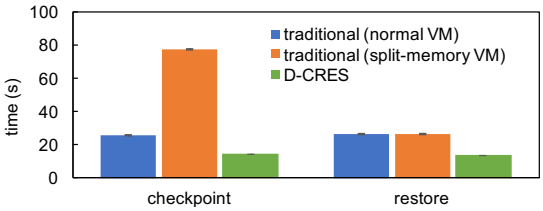

Figure 6. The time for checkpoint/restore.

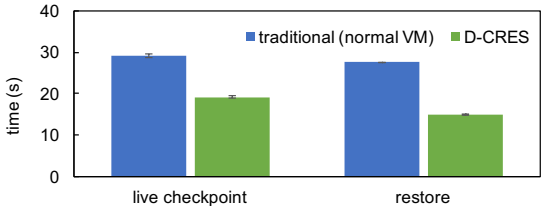

Figure 7. The time for live checkpoint/restore.

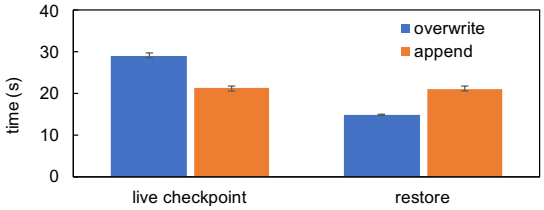

Figure 8. The checkpoint/restore time for different checkpoint file formats.

two PCs with an Intel Core i7-7700 processor, 8 GB of DDR4 DRAM, and 1 TB of SATA HDD. These PCs were connected with Gigabit Ethernet. We ran Linux 4.4.169 and QEMU-KVM 2.4.1. For a split-memory VM, we assigned one virtual CPU and 4 GB of memory and equally split the memory into two. We measured the performance 5 times for each experiment.

5.1. Checkpoint/Restore Time

We first measured the time for checkpoint/restore of the split-memory VM using D-CRES. For comparison, we measured the time when we used traditional checkpoint/restore for the split-memory VM. We also applied the traditional mechanism to a normal VM running on one host. In this experiment, we performed non-live checkpointing and stopped a VM during checkpointing to exclude the impact of memory updates by the VM.

Fig. 6 shows the time for checkpointing and restoring a VM. When we took a checkpoint of the split-memory VM, D-CRES was 5.4 times faster than the traditional mechanism. This is because D-CRES could avoid remote paging caused by checkpointing. Compared with traditional checkpointing of the normal VM, D-CRES was even 77% faster thanks to parallel checkpointing at the two hosts. However, the checkpoint time in D-CRES was more than the half of that for the normal VM although the amount of memory to be saved was reduced to the half at each host. This is due to saving the other states as well at the main host.

For restoring, D-CRES was 89% faster than the traditional mechanism. The restore time in D-CRES was also not reduced to the half of that in the traditional mechanism. The reason is the same as that for checkpointing. Since the traditional mechanism always restores a normal VM, the restore time was almost the same between when we used a checkpoint taken from the normal VM and when we used that from the split-memory VM.

Next, we measured the time for live checkpoint/restore of a split-memory VM in D-CRES when the VM itself caused remote paging during checkpointing. For comparison, we performed traditional live checkpointing of a normal VM. We ran a program that allocated 2 GB of memory and modified it in the VM. As shown in Fig. 7, live checkpointing in D-CRES was 51% faster than the traditional mechanism, while restoring in D-CRES was 84% faster.

5.2. Impact of Checkpoint File Formats

To examine the impact of the format of checkpoint files in live checkpointing, we first measured the total size of the checkpoint files in D-CRES. We ran a program that allocated only 1 GB of memory and modified it in the VM. D-CRES used memory files for saving memory data and overwrote memory data in the files by updated one. For comparison, we measured the total size when D-CRES appended updated memory data to the end of checkpoint files, as done in the traditional live checkpointing. The total size in D-CRES was always 4.0 GB regardless of the amount of updated memory data. In contrast, when D-CRES appended the data to the checkpoint files, the total size increased to 5.3 GB.

Next, we compared the time needed for live checkpoint/restore between the two formats of checkpoint files. When D-CRES used memory files, the checkpoint time was 37% longer, as shown in Fig. 8. This is because many seek operations occurred to overwrite specific blocks in the files. In contrast, the restore time was 30% shorter. As such, a trade-off exists among the size of checkpoint files, the checkpoint time, and the restore time.

5.3. Impact of Memory-split Ratios

We measured the time for checkpoint/restore when we changed the ratio of the amount of memory of a VM allocated to the two hosts. Fig. 9 shows the time for each memory-split ratio. Both the checkpoint and restore times were minimized when we assigned 70% of the memory of a VM to the sub-host. This is because QEMU-KVM in the main host saved and restored not only the memory but also the state of the VM core and a snapshot of the virtual disk.

6. Related Work

To achieve high availability of VMs, Remus [7] prepares an active VM and a backup VM at different hosts. It transfers only the difference of the state of the active VM to the backup VM. Network transmission and disk writes are buffered until the synchronization is completed. Even if the active VM stops due to a failure, the execution can be transparently continued by switching to the backup VM. Kemari [8] reduces the frequency of

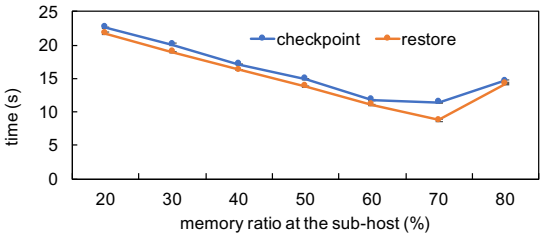

Figure 9. The checkpoint/restore time for different memory-split ratios.

the synchronization by synchronizing only when network transmission and disk writes are performed by a VM. COLO [9] transfers the request packet received by an active VM to a backup VM as well and waits until the response from both VMs matches. Since these methods require two VMs, it is often difficult to apply them to a large-memory VM.

For distributed checkpointing, Emulab [10] saves the states of multiple VMs in a closed distributed system together with that of the network. It modifies the OS kernel in the VM to stop the execution and the clocks on checkpointing. It synchronizes the clocks and then stops all the VMs at the same time to avoid packet delay and new in-flight packets due to some of the VMs being stopped earlier. In addition, it takes the checkpoints of delay nodes on the network to prevent in-flight packets from being lost due to network delay. D-CRES can take a checkpoint without in-flight packets by finally synchronizing remote paging because remote paging is only performed synchronously between hosts.

VM migration has a similarity to checkpoint/restore. It takes a checkpoint of a VM at the source host, transfers it to the destination host, and restores the VM from the checkpoint. In particular, the implementation of subst migration [11] is similar to checkpoint/restore of D-CRES. Subst migration moves the entire state of a split-memory VM in either the main host or one of the sub-hosts to a new host. However, D-CRES requires large modification to the implementation of subst migration. For example, it uses a sparse file to reduce the total size of checkpoint files. It needs a custom mechanism for preserving integrity when remote paging is caused during checkpointing.

VM co-migration requires synchronization between two dependent VMs. VMCoupler [12] migrates a VM to be monitored with a VM running intrusion detection systems at the same time. It synchronizes the pause and termination of the two VMs at the source host and the creation and restart of the VMs at the destination host so that monitoring can continue. D-MORE [13] migrates a VM to be managed with a VM used for remote management together. It synchronizes the migration of the two VMs at more points. D-CRES needs less synchronization points because it synchronizes checkpointing only between a VM and memory servers, not between VMs.

For parallel VM migration, PMigrate [14] uses data parallel and pipeline parallel with surplus CPUs and NICs. To improve the scalability of the mmap and munmap system calls executed in parallel, a method called range lock has been proposed. Even though D-CRES currently parallelizes checkpoint/restore at a host granularity, it is more effective for a large-memory VM to use parallelism within a host together.

7. Conclusion

This paper proposed D-CRES for efficient and flexible checkpoint/restore of split-memory VMs across multiple hosts. D-CRES saves the memory of a VM in parallel at multiple hosts and avoids remote paging caused by checkpointing. For live checkpointing, it consistently saves the memory of a running VM by considering remote paging caused by the VM itself. Upon a host or network failure, it restores a split-memory VM in parallel at multiple hosts. At this time, it can relocate the memory of a split-memory VM among hosts to use an arbitrary set of hosts. We have implemented D-CRES in KVM and conducted several experiments to show the efficiency of checkpoint/restore of a split-memory VM in D-CRES.

Our future work is supporting incremental checkpointing of split-memory VMs to reduce the overhead of checkpointing. Since remote paging always changes how the memory is split in a split-memory VM, it is necessary to efficiently detect memory differences across multiple hosts.

Acknowledgements

The research results have been achieved by the "Resilient Edge Cloud Designed Network (19304)," the Commissioned Research of National Institute of Information and Communications Technology (NICT), Japan.

References

[1] Amazon Web Services, Inc. Amazon EC2 High Memory Instances. https://aws.amazon.com/ec2/instance-types/high-memory/, 2019.

[2] Apache Software Foundation. Apache Spark – Lightning-Fast Cluster Computing. http://spark.apache.org/.

[3] Facebook, Inc. Presto: Distributed SQL Query Engine for Big Data. https://prestodb.io/.

[4] SAP SE. What is SAP HANA? An Unrivaled Data Platform for the Digital Age. https://www.sap.com/products/hana.html.

[5] Microsoft Corporation. SQL Server 2017 on Windows and Linux. https://www.microsoft.com/en-us/sql-server/sql-server-2017.

[6] M. Suetake, T. Kashiwagi, H. Kizu, and K. Kourai. S-memV: Split Migration of Large-Memory Virtual Machines in IaaS Clouds. In *Proc. Int. Conf. Cloud Computing*, pages 285–293, 2018.

[7] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *Proc. Symp. Networked Systems Design & Implementation*, pages 161–174, 2008.

[8] Y. Tamura. Kemari: Virtual Machine Synchronization for Fault Tolerance using DomT. In *Xen Summit Boston 2008*, 2008.

[9] Y. Dong, W. Ye, Y. Jiang, I. Pratt, S. Ma, J. Li, and H. Guan. COLO: COarse-grained LOck-stepping Virtual Machines for Non-stop Service. In *Proc. Annual Symp. Cloud Computing*, 2013.

[10] A. Burtsev, P. Radhakrishnan, M. Hibler, and J. Lepreau. Transparent Checkpoints of Closed Distributed Systems in Emulab. In *Proc. European Conf. Computer Systems*, 2009.

[11] T. Kashiwagi and K. Kourai. Flexible and Efficient Partial Migration of Split-memory VMs. In *Proc. Int. Conf. Cloud Computing*, 2020.

[12] K. Kourai and H. Utsunomiya. Synchronized Comigration of Virtual Machines for IDS Offloading in Clouds. In *Proc. Int. Conf. Cloud Computing Technology and Science*, pages 120–129, 2013.

[13] S. Kawahara and K. Kourai. The Continuity of Out-of-band Remote Management across Virtual Machine Migration in Clouds. In *Proc. Int. Conf. Utility and Cloud Computing*, pages 176–185, 2014.

[14] X. Song, J. Shi, R. Liu, J.Yang, and H. Chen. Parallelizing Live Migration of Virtual Machines. In *Proc. Int. Conf. Virtual Execution Environments*, pages 85–96, 2013.