

An Efficient Virtual Machine Checkpointing Mechanism for Hypervisor-based HPC systems



K. Chanchio

C. Leangsuksun, H. Ong, V. Ratanasamoot, and A. Shafi

Thammasat University,
Louisiana Tech University,
Oak Ridge National Laboratory,
NUST Institute of Information Technology





Outline

- Motivation
- Related works
- CEVM Architecture
- Protocols
- Implementation and Analysis
- Progress and Future works



Motivation

- System-level virtualization uses VMM (hypervisor) to manage resources and to provide VM abstraction
 - VM's state can be saved and restored
- VM services can be “out of order” during
 - Recording of VM memory and hardware state
 - Storing of a snapshot of VM disk image; copying a subset of a stack file system
- *The more frequent that we do checkpointing, the higher the application turnaround time*



CEVM

- We propose the “Checkpointing-Enabled Virtual Machine (CEVM)” system
- CEVM aims:
 - To enable implicit system-level fault tolerance without modifying existing OS or applications, and
 - To minimize space and time overheads needed to execute software that cannot tolerate fault

This talk will focus on the fundamental mechanisms of CEVM and leaving advanced features such as Coordinated checkpointing to future work



Paper Contributions

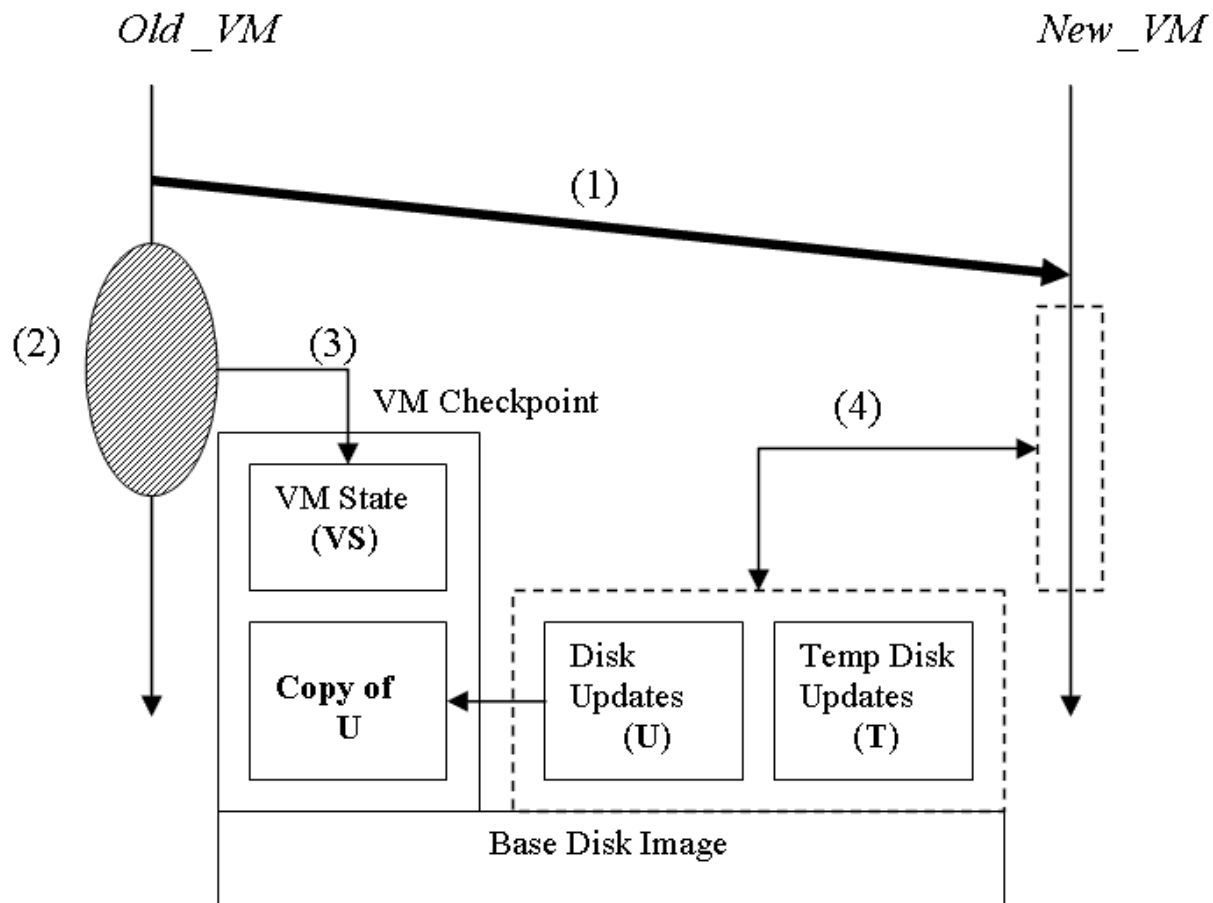
- A novel approach to checkpoint/restart mechanism
 - Uses hypervisor-based VM migration capability
- Efficient Protocols
 - A Checkpointing Protocol
 - Uses VM migration mechanism to move a VM to a new processor/node while letting original VM checkpoint
 - *Allows computation and Checkpointing to overlap*
 - A Disk Image Manipulation Protocol
 - Uses overlay disk images to handle disk update during checkpointing
 - *Provides correct concurrent disk accesses*



Related Works

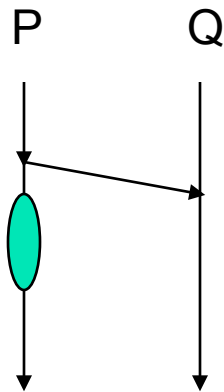
- Application-level Checkpointing
 - Source code is modified or relinked
- Kernel-level Checkpointing
 - OS dependent
- VM checkpoint-restart system
 - Qemu, KVM, Xen, etc.
 - “Out of order” during checkpointing

CEVM Architecture Overview

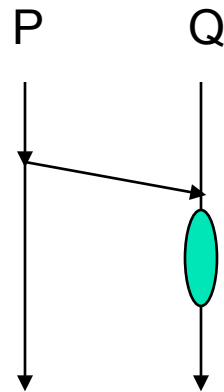


- CEVM approach to implement an efficient checkpointing mechanism

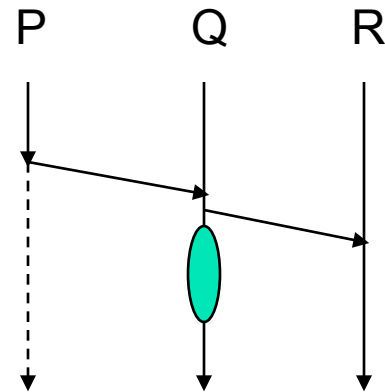
Scheduling Possibilities



(a)



(b)

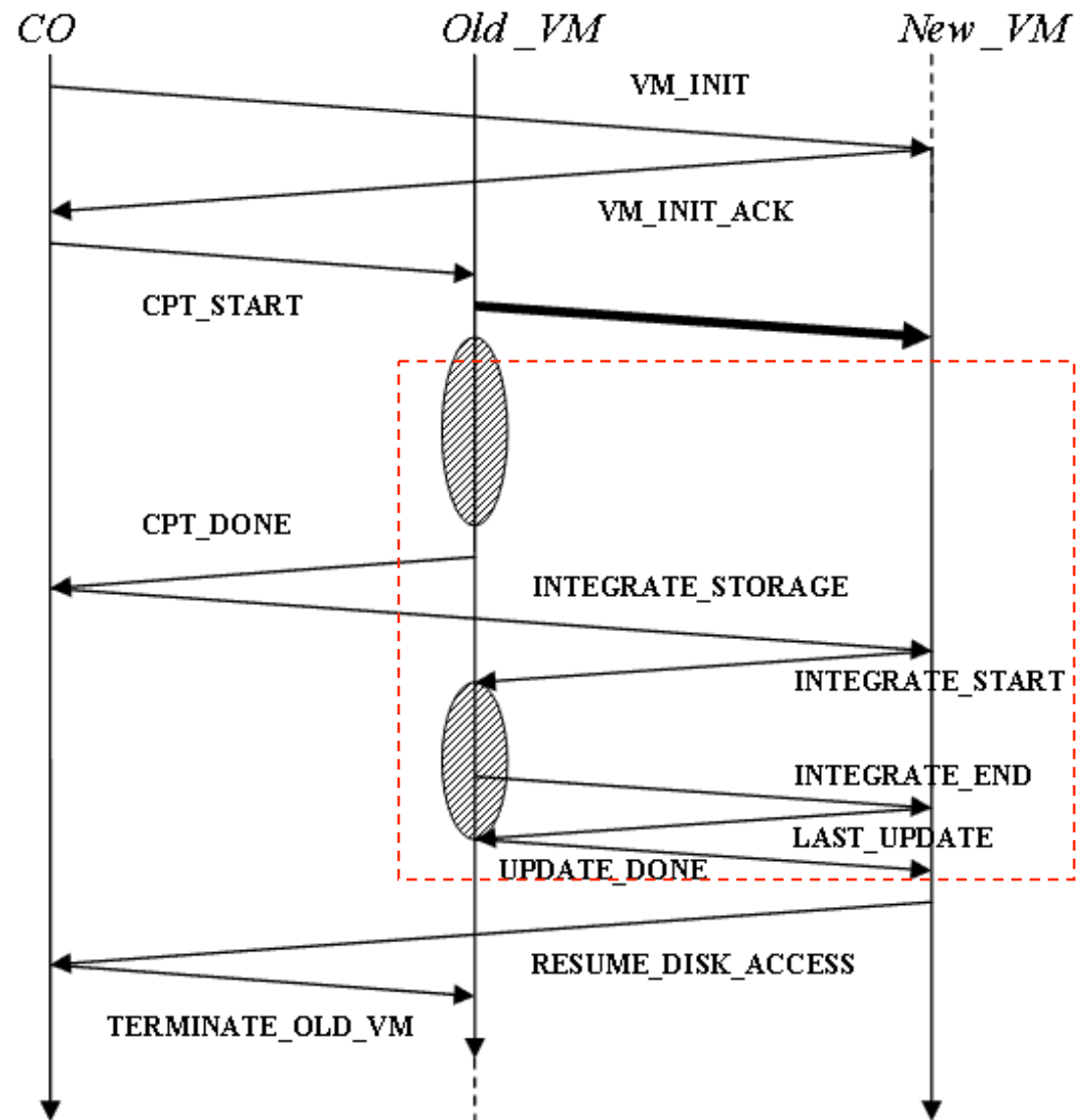


(c)

- This approach allows us to separate checkpointing tasks from original computation
- They can be assigned to processors in various ways

Protocols

- The Checkpointing Protocol
 - Use CO to coordinate migration
- The Disk Image Manipulation protocol
 - Handle disk access requests





Checkpointing Protocol

1. While $\neg DONE$ do
2. if $\neg EOF_on_T$ then
3. if nextCluster(T , &content, &Taddress) $\neq EOF$ then
4. updateCluster(U , content, Taddress)
5. else
6. send INTEGRATE_END to New_VM
7. $EOF_on_T = true$
8. end if
9. end if
10. nrecv m from New_VM
11. if ($m \neq LAST_UPDATE \wedge m \neq NULL$) then
12. accessDisk(m)
13. else if ($m \neq NULL$) then
14. send UPDATE_DONE to New_VM
15. $DONE = true$
16. end if
17. end while

Disk Image Manipulation Protocol

AccessDisk(*m*)

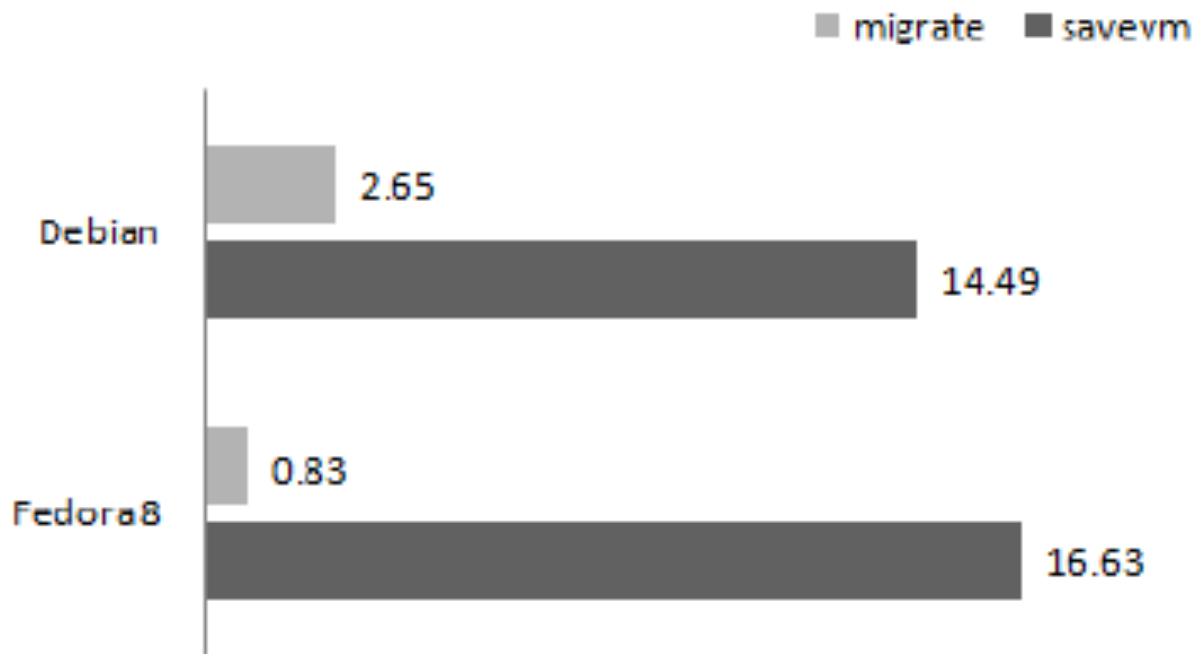
1. if *m* is a disk update request then
2. if (*m.address* < *T.address* + *cluster_size*) ∨ *EOF_on_T* then
3. updateCluster(U, *m.content*, *m.address*)
4. else
5. updateCluster(T, *m.content*, *m.address*)
6. end if
7. else // *m* is a read request
8. if (*m.address* < *T.address* + *cluster_size*) ∨ *EOF_on_T* then
9. read from U or base image, and send a cluster to New_VM
10. else
11. read from T or U or base image, and send a cluster
12. end if
13. end if



Implementation

- Thread-base vs. Process-base
 - Each VM can be assigned to processors using the Linux “set affinity” system call
- Thread-base VM is efficient
 - Pro: Threads can share VM memory data structure
 - Con: May require substantial VM modification
- Process-base VM is portable
 - Pro: Minimal modification to VM code
 - Con: May be less efficient as compare to the thread-base approach

Feasibility Study



- Comparisons of KVM migration and save state performance on a notebook computer with Centrino Duo CPU with 2 GB RAM



Performance Discussion

Turnaround time with Traditional Checkpointing:	$T_{vm} = T_{comp} + nT_{cpt}$
with migration:	$T_{vm} = T_{comp} + nT_{mg}$
with CEVM:	$T_{vm} = T_{comp} + nT_{cevm} + T_o$

$$T_{mg} \leq T_{cevm} + T_o/n \leq T_{cpt}$$



Progress and Future Works

- Theoretical work
 - Sounds architecture design, and protocols
 - Conducted feasibility study
- Implementation is still at the embryonic stage
 - Currently, we are prototyping the CEVM system on a KVM (process-base)
- Future work
 - Incorporate VM coordinated checkpointing mechanisms
 - Performance evaluation: throughput, turn-around time, scheduling and scalability
 - Thread-base implementation



Acknowledgements

- Thanks to everyone in our team and our affiliated institutions that make this collaboration possible. We also thanks the reviewer for good comments and suggestions.
- Thank you
- Questions/Comments ?