Improving the Performance of Hypervisor-Based Fault Tolerance

Jun Zhu, Wei Dong, Zhefu Jiang, Xiaogang Shi, Zhen Xiao,
Xiaoming Li

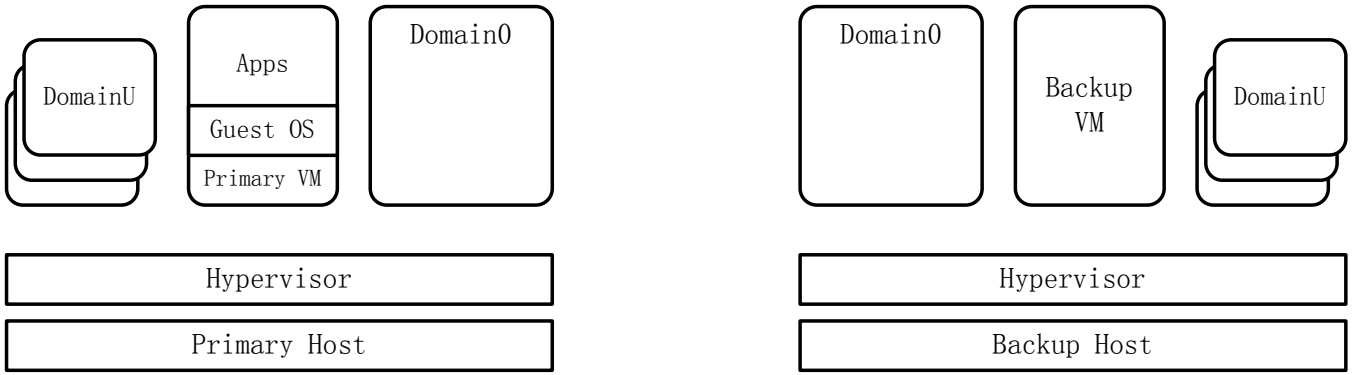
School of Electronics Engineering and Computer Science
Peking University, China

Tuesday - 20 April 2010

Background

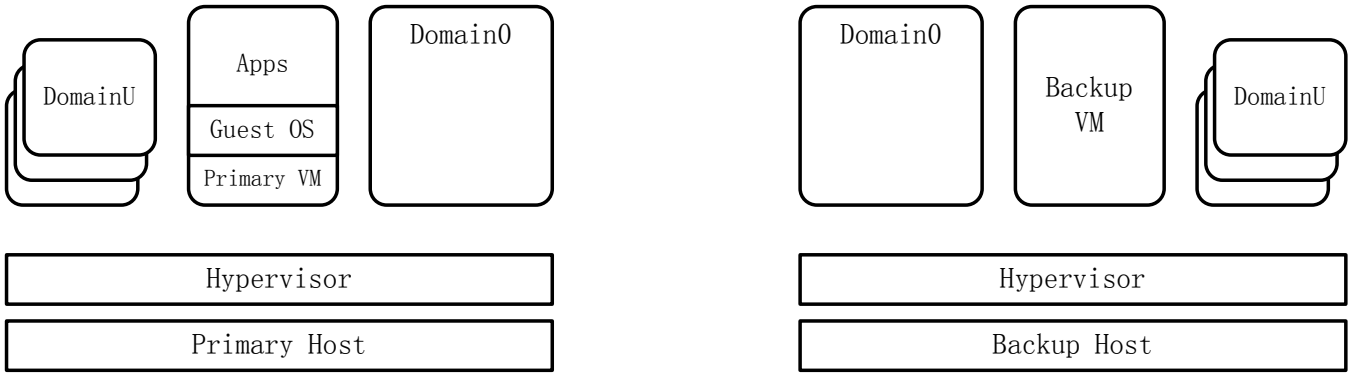
- System failures exist inevitably, e.g. power loss, OS halting.
- Expensive to mask failures transparently.
  - Redundant software or redundant hardware component
  - e.g. HP non-stop server
- Laborious effort on application development.
  - Applications based on fault tolerant protocols
  - e.g. Google Chubby based on Paxos

Hypervisor-Based Fault Tolerance



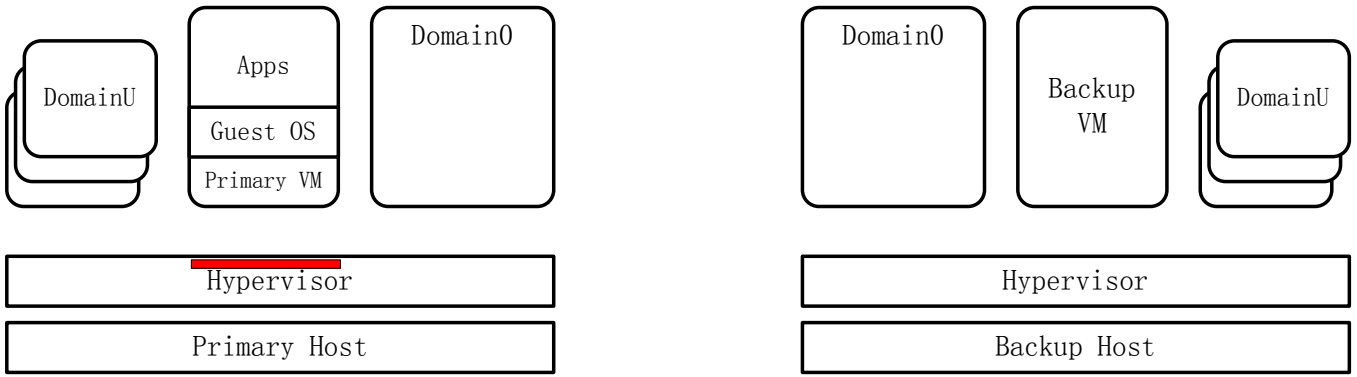
- The primary VM and the backup VM reside in different physical hosts.

Hypervisor-Based Fault Tolerance



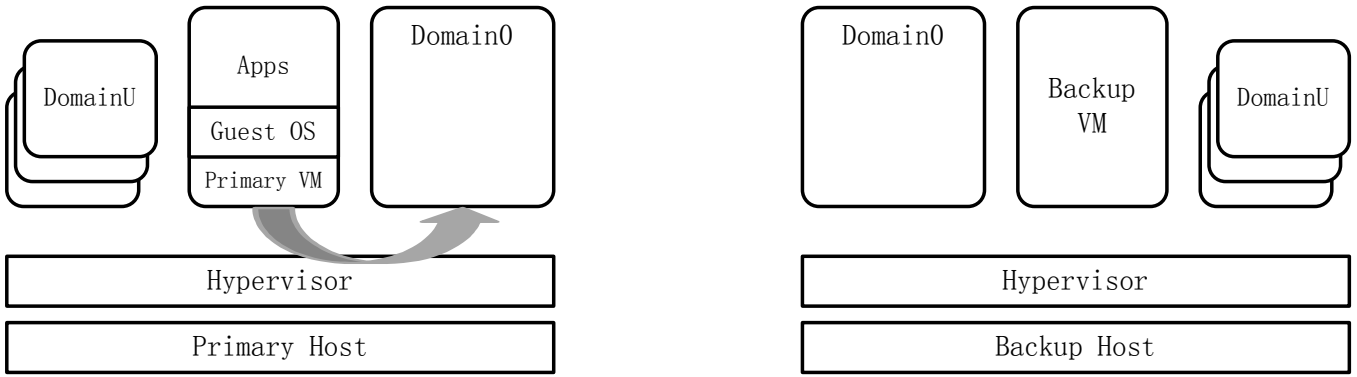
- The primary VM and the backup VM reside in different physical hosts.
- The execution of the primary VM is divided into epochs (e.g. 20msec).

Hypervisor-Based Fault Tolerance



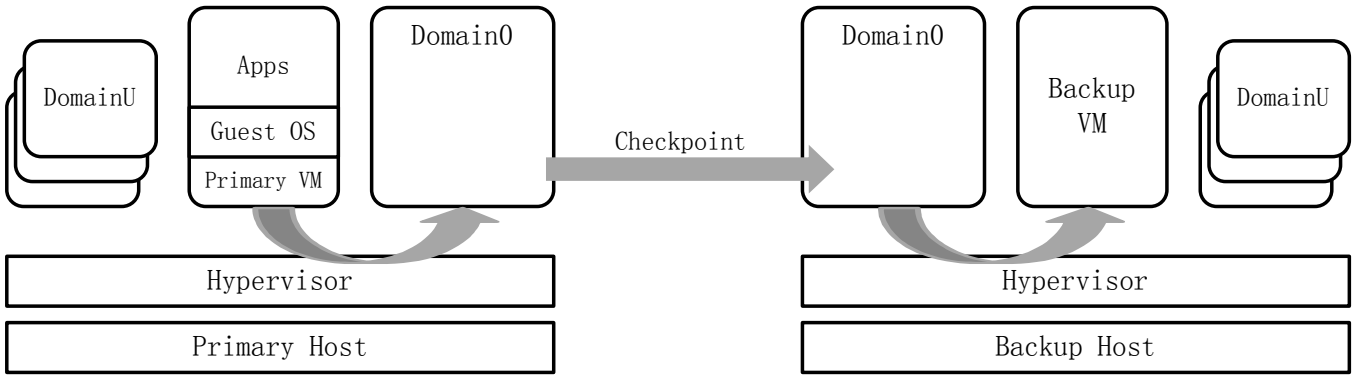
- The primary VM and the backup VM reside in different physical hosts.
- The execution of the primary VM is divided into epochs (e.g. 20msec).
- The hypervisor records modified pages in each epoch. (Expensive)

Hypervisor-Based Fault Tolerance



- The primary VM and the backup VM reside in different physical hosts.
- The execution of the primary VM is divided into epochs (e.g. 20msec).
- The hypervisor records modified pages in each epoch. (Expensive)
- Modified pages are mapped into Domain0's address space. (Expensive)

Hypervisor-Based Fault Tolerance



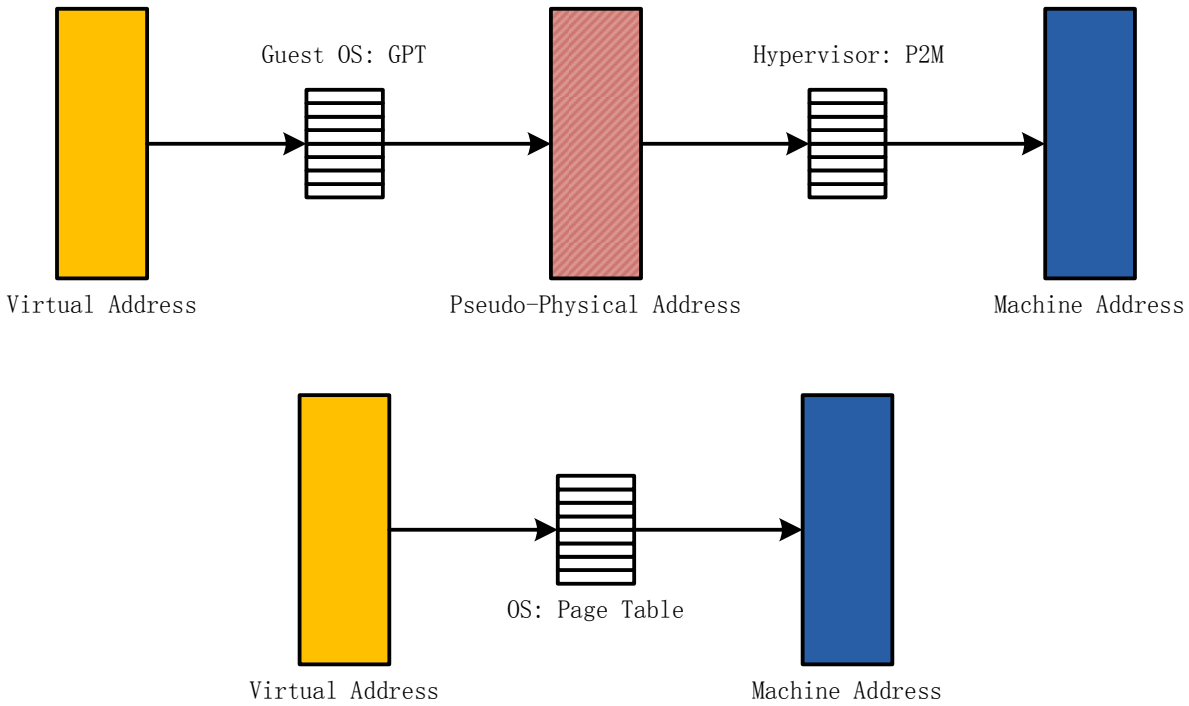
- The primary VM and the backup VM reside in different physical hosts.
- The execution of the primary VM is divided into epochs (e.g. 20msec).
- The hypervisor records modified pages in each epoch. (Expensive)
- Modified pages are mapped into Domain0's address space. (Expensive)
- Checkpoint is sent to the backup VM.

Objective

- Efficient memory tracking mechanism
  - Read fault reduction
  - Write fault prediction
- Efficient memory mapping machanism
  - Software superpage

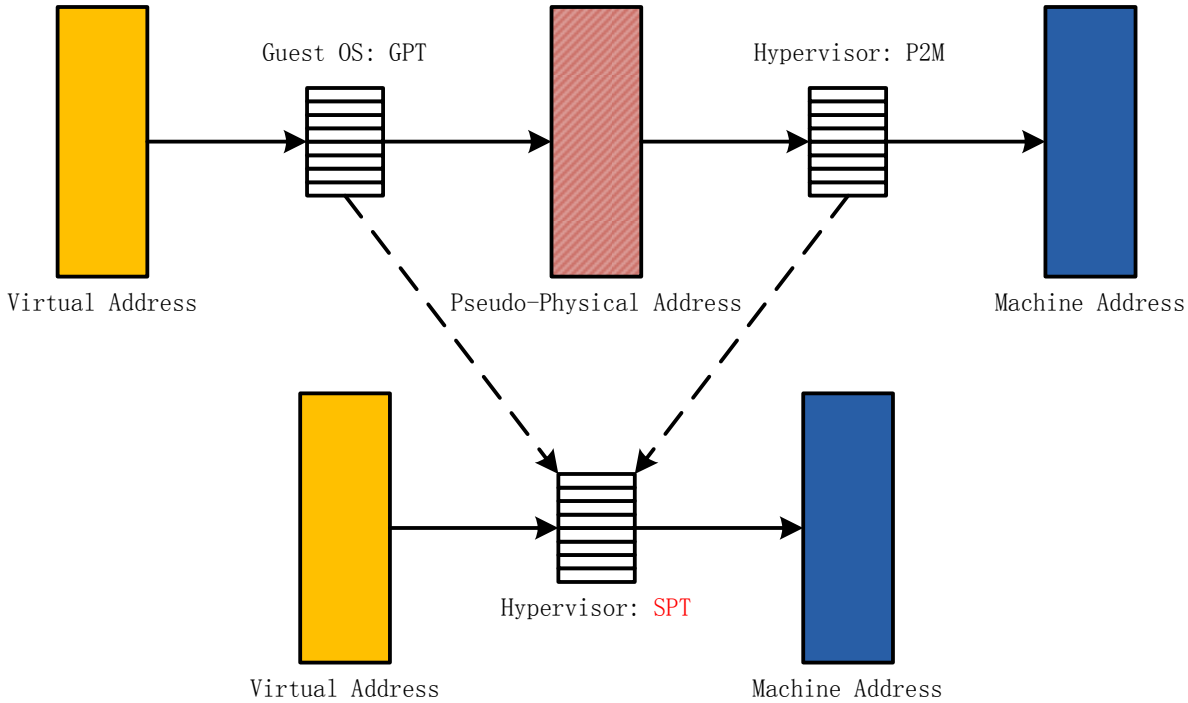
Outline

Shadow Page Table (SPT)



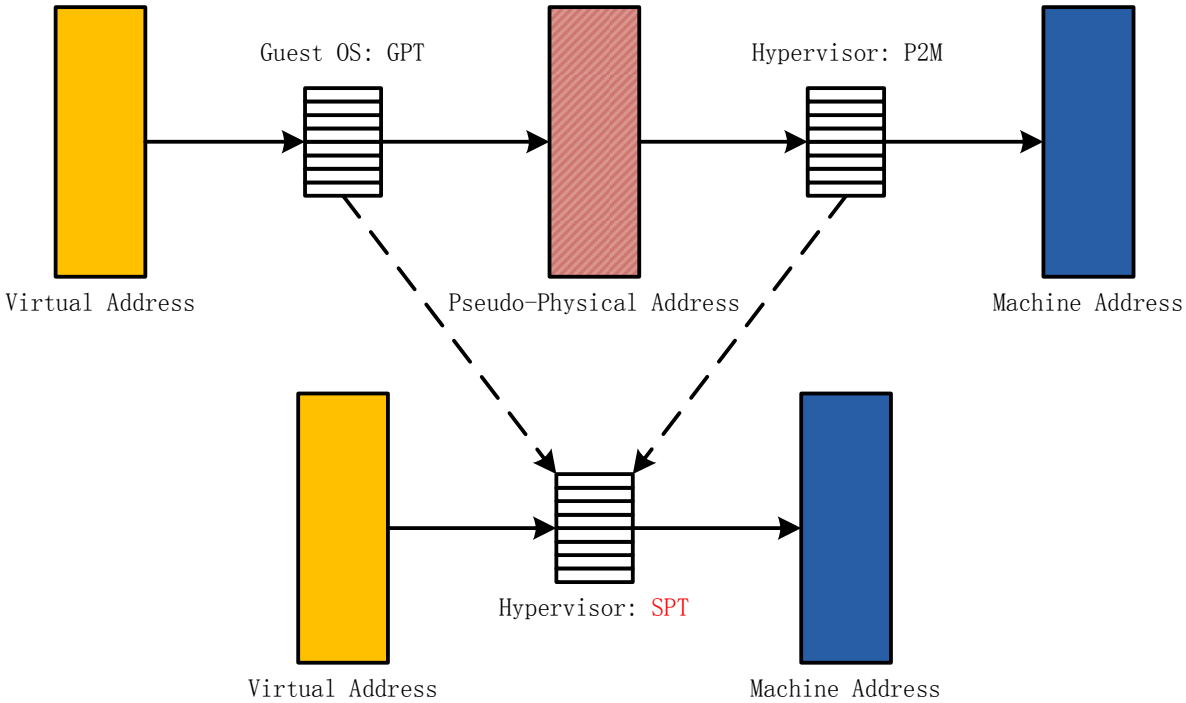
- Virtualization: virtual, pseudo-physical and machine address space; Traditional OS: virtual and machine address space.

Shadow Page Table (SPT)



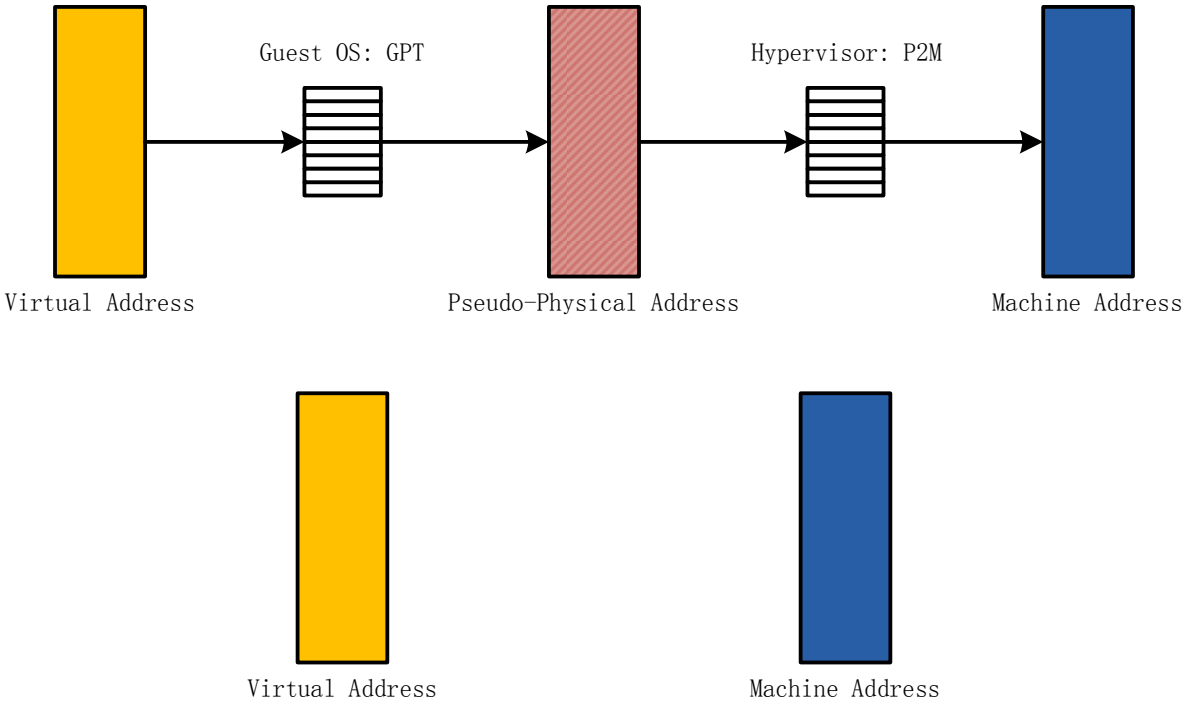
- Virtualization: virtual, pseudo-physical and machine address space; Traditional OS: virtual and machine address space.
- SPT is created on demand according to the guest page table (GPT) and pseudo-physical to machine table (P2M).

Current Memory Tracking Approach



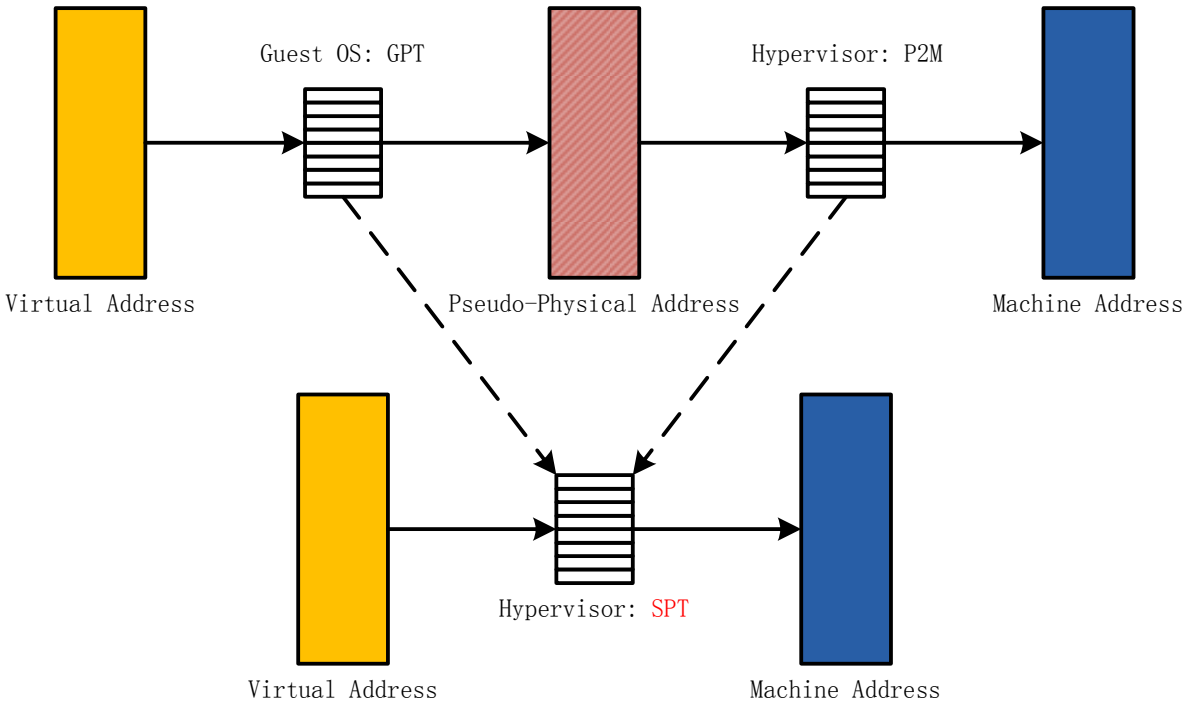
- Developed for live migration, it is not suitable for frequent checkpointing in hypervisor-based fault tolerance.

Current Memory Tracking Approach



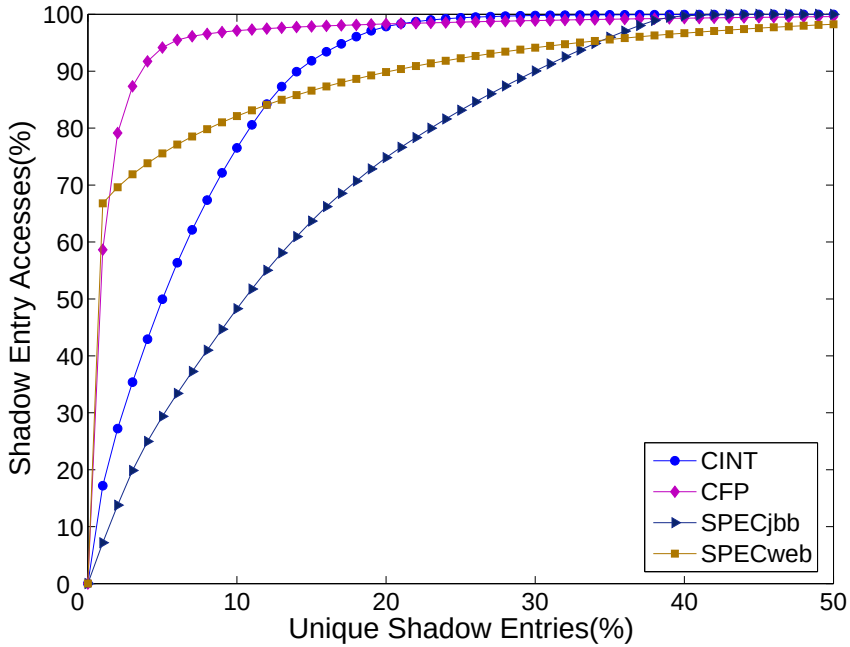
- Developed for live migration, it is not suitable for frequent checkpointing in hypervisor-based fault tolerance.
- At the beginning of each epoch, all SPTs are destroyed.

Current Memory Tracking Approach



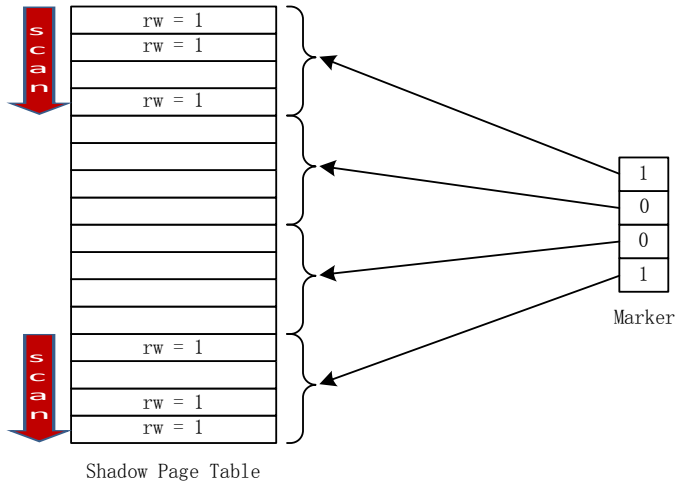
- Developed for live migration, it is not suitable for frequent checkpointing in hypervisor-based fault tolerance.
- At the beginning of each epoch, all SPTs are destroyed.
- During the epoch, any memory access induces a page fault and write accesses can be identified.

Observation: Shadow Entry Reuse



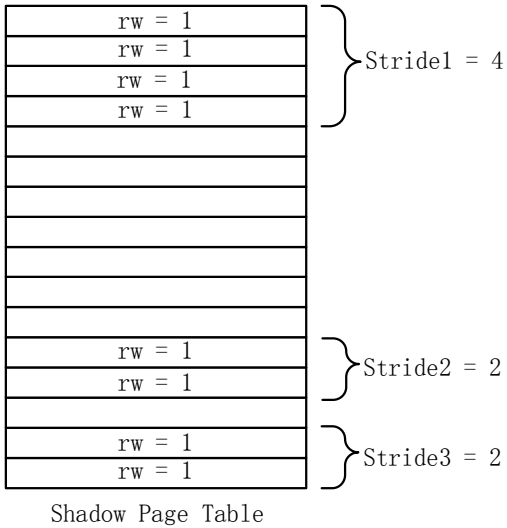
- Shadow Entry Reuse: If a shadow entry is accessed in an epoch, it will likely be accessed in future epochs.
- Reuse Degree: The percentage of unique shadow entries required to account for a given percentage of page accesses.

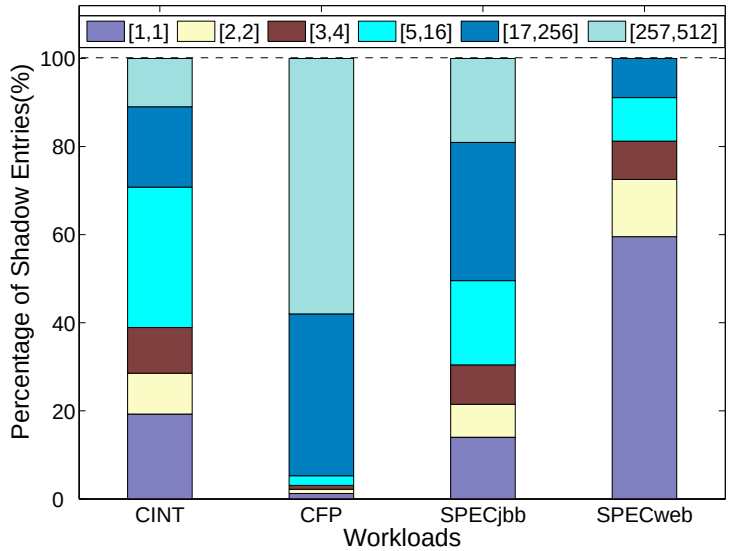
Our Approach: Scanning Shadow Entries



- Our approach: Preserve SPTs and revoke their write permission.
- During the epoch, the marker records which parts have dirty pages.
- At the end of the epoch, shadow entries are selectively checked based on the marker.
- During checking, we record the dirty pages and revoke write permission for the next epoch.

Observation: Spatial Locality

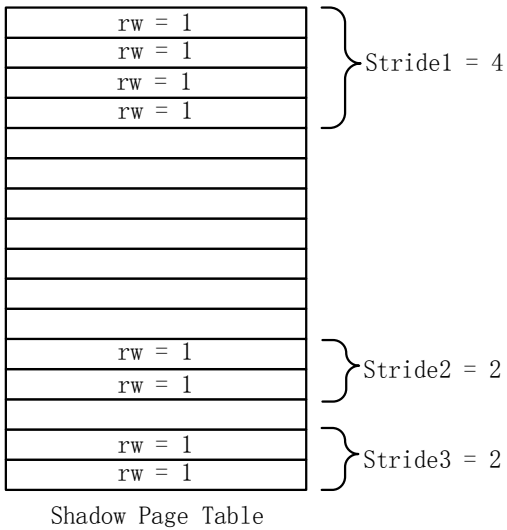


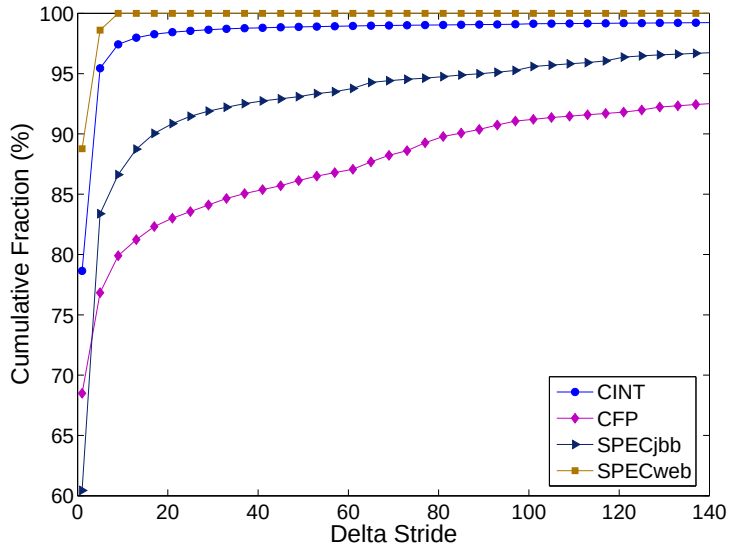


- Stride: Consecutive shadow entries with rw permission in the same epoch.
- Ave_stride: Average length of strides for each SPT.

- Spatial Locality: The shadow entries with rw set are inclined to cluster together.

Observation: History Similarity





- Stride: Consecutive shadow entries with rw set in the same epoch.
- Ave_stride: Average length of strides for each SPT.

- History Similarity: For a particular SPT, the behavior of spatial locality is inclined to be similar among adjacent epochs.

Our Approach: Predicting Write Accesses

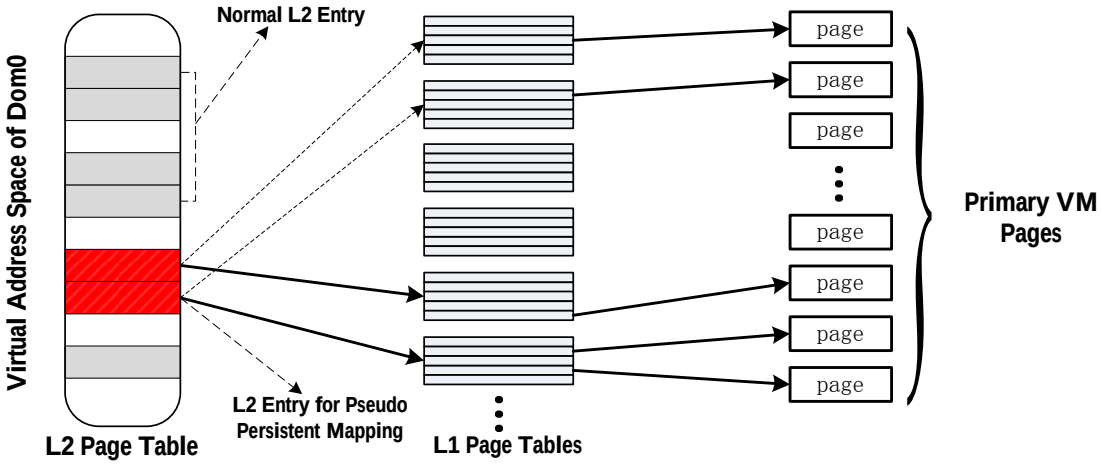
- When a write fault occurs, the adjacent shadow entries will likely be referenced for write accesses. (Spatial Locality)
- How many shadow entries are predicted is decided by the historical ave_strides. (History Similarity)
- $$his\_stride = his\_stride * \alpha + ave\_stride * (1 - \alpha)$$
- 1/3 his_stride backwards and his_stride afterwards shadow entries are predicted (heuristically).
- When a shadow entry is predicted, we set rw in advance with *Prediction* bit tagged.
- At the end of each epoch, we rectify wrong predictions by checking *Prediction* and *Dirty* bits.

Outline

Observation: Memory Mapping is Expensive

- At the end of each epoch, modified pages are mapped into Domain0's address space.
- Memory mapping and umapping are expensive, resulting in the primary VM being stalled too long.
- Observation: Because of locality, the mappings can be reused, without mapping and umapping frequently.

Our Approach: Software Superpage



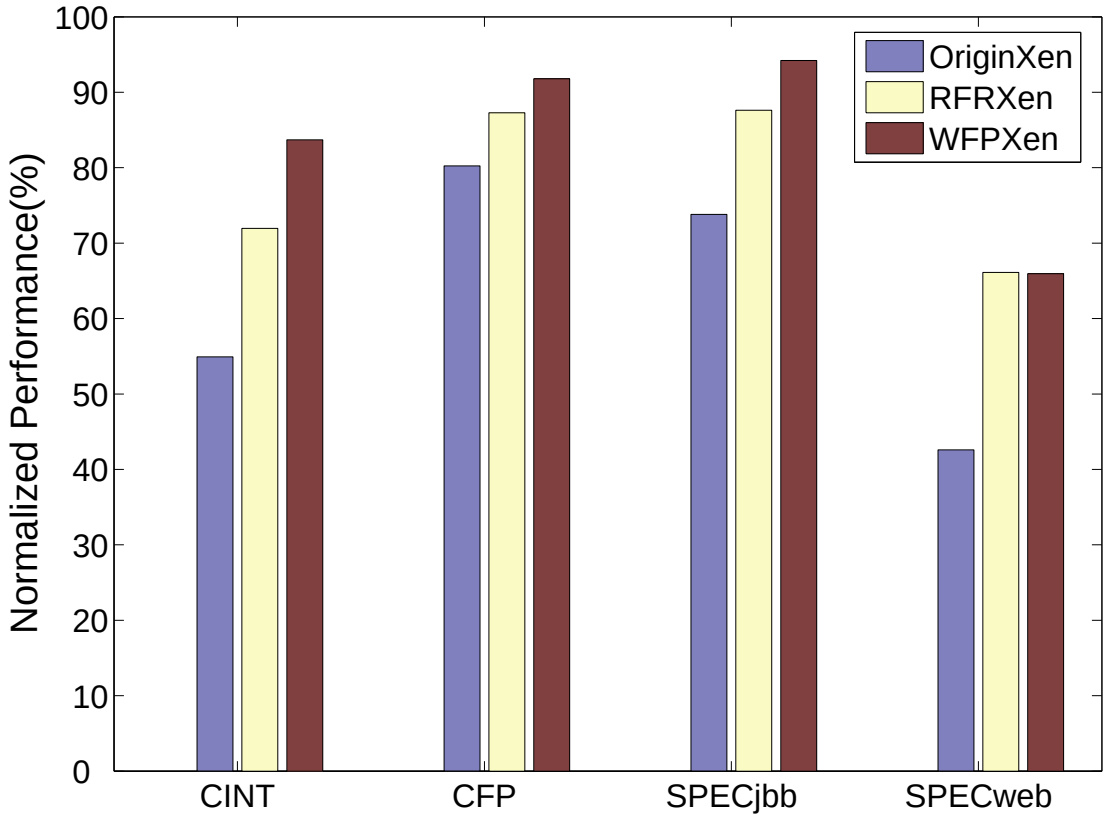
- L1 page tables are allocated to point to the primary VM's entire memory pages, but limited L2 page table entries.
- L1 page tables are installed into these limited L2 page table entries on demand.
- LRU algorithm is employed to decide which L1 page tables are actually pointed to.

Outline

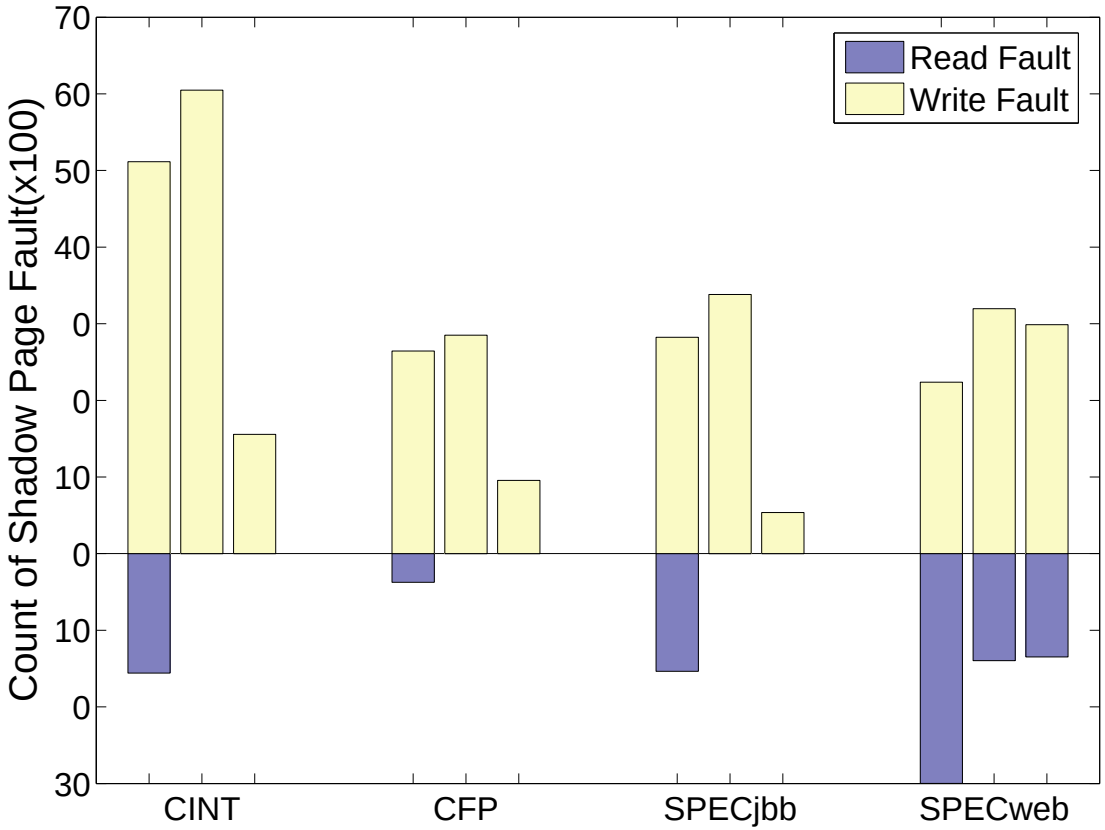
Evaluation Setup

- Two HP ProLiant DL180 servers with 12G memory and two quad-core CPUs, connected via switched Gigabit Ethernet.
- The primary VM: 2G memory and one vCPU.
- Workloads: SPEC Int, SPEC Fp, SPEC Jbb and SPEC Web.
- Epoch length: 20 msec as default.
- Improving two sources of overhead.
  - Memory tracking mechanism: read fault reduction and write fault prediction.
  - Memory mapping mechanism: software superpage.

Memory Tracking: Performance Improvement



Memory Tracking: Page Faults Reduction



From left to right, current approach, our read fault reduction and write fault prediction.

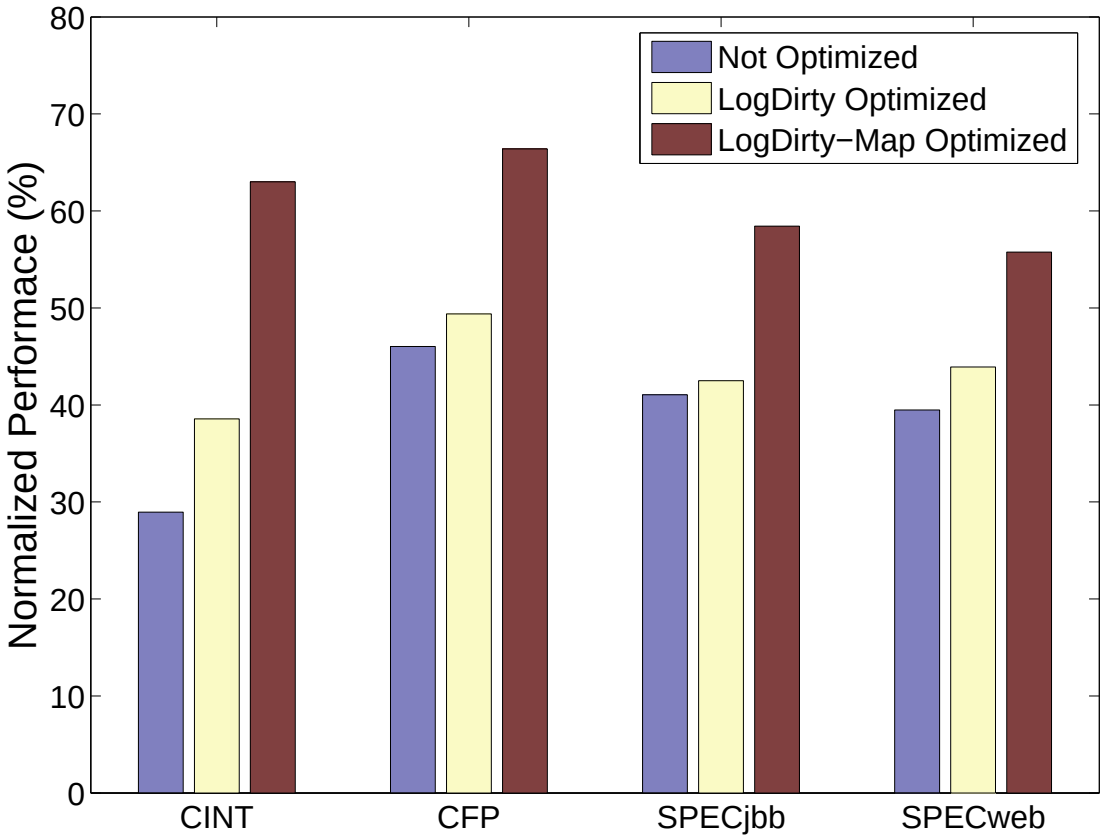
Software Superpage: Mapping Hit Ratio.

We allocate a fixed 64M virtual address space in Dom0 to map all the memory pages of the primary VM (2G).

Table: Software superpage mapping hit ratio.

Workload	*CINT*	*CFP*	*SPECjbb*	*SPECweb*
Hit Ratio	97.27%	97.25%	97.80%	79.45%

Overall Improvement



Conclusion

- Memory tracking and memory mapping are two sources of overhead in hypervisor-based fault tolerance.
- *Read fault reduction* eliminates those unnecessary page faults from read accesses.
- *Write fault prediction* reduces page faults by predicting the pages that will likely be modified.
- *Software superpage* improves memory mapping with limited virtual address space.

Future Work

- We will evaluate our experiments with different epochs and different number of vCPUs.
- Port our approach to nested page table (another memory virtualization mechanism).
- Improve our source code and contribute it to Xen open source project.

Q & A

Thank You!