TreeSLS: A Whole-system Persistent Microkernel with Tree-structured State Checkpoint on NVM

Maria Trakosa

M131 Advanced Operating Systems

Overview

- **TreeSLS Proposal:** TreeSLS is a whole-system persistent microkernel. It simplifies the entire system's state maintenance through a capability tree and a robust checkpoint manager.
- **Exploiting NVM:** TreeSLS takes advantage of the emerging non-volatile memory (NVM) technology, eliminating the need to distinguish between ephemeral and persistent devices.
- **Delayed External Visibility:** ensures transparent external synchrony with minimal overhead.
- **Performance Evaluation**: Microbenchmarks and real-world applications demonstrate that TreeSLS achieves whole-system persistence in approximately 100 µs and can take a checkpoin every 1 ms with reasonable overhead to applications.

Limitations of Existing SLSs

Single-Level Store (SLS)

Uses checkpointing to extend the memory layer down to include disks. This approach manages data, both permanent and ephemeral, and system state together with transparent persistence.

Issues with Existing SLSs

- Two-Tiered Memory-Storage Hierarchy
  - Additional Cache Layers
  - Write-Amplification and Checkpoint Frequency
- External Synchrony Issue

Addressing Challenges with TreeSLS

- NVM as a Single-Level Device
- Careful Relation Maintenance
- Checkpointing Optimized for NVM
- Transparent External Synchrony

Overview of TreeSLS Architecture

- Microkernel Architecture
- NVM Integration
- Addressing Two Key Challenges
  - Efficient Whole-System State Capture
  - Efficient Whole-System Checkpointing

Overview of TreeSLS Architecture

- Checkpoint/Restore Procedure
  1. IPI requests
  2. Checkpoint of runtime capability tree
  3. Speculative copy of pages
  4. Increment global version number
  5. Resume execution
  6. Copy on-write
  7. Restore: rolls back

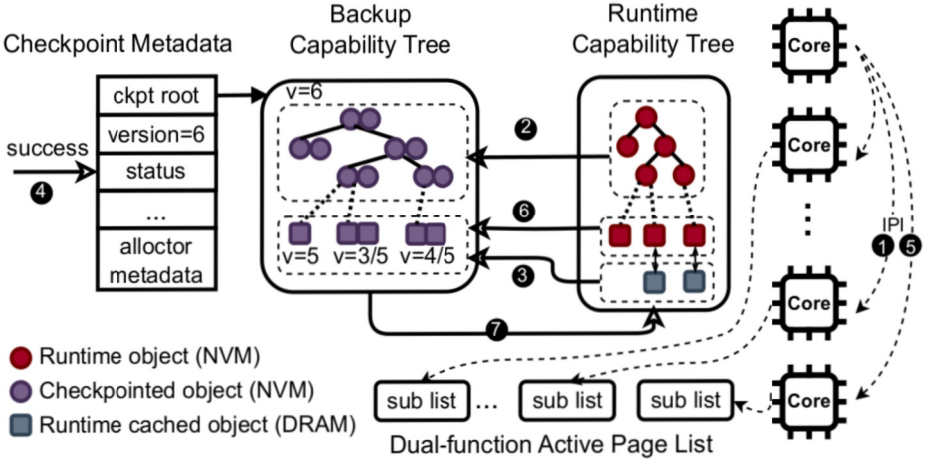


Figure 5. The checkpoint/restore procedure.

- Ensuring Correctness

Checkpointing the Capability Tree

- Capability: A reference to an object and its access rights
- Ensuring Efficiency
- Checkpoint Strategies for Various Objects
  - Cap Group
  - Thread
  - IPC Connection, Notification, and IRQ Notification
  - VM Space and Page Tables
  - PMO and Memory Pages

Copy Methods

- Stop-and-copy
- Speculative stop-and-copy
- Copy-on-write
- Speculative copy-on-write
- Hybrid Copy
  - Speculative Copy of Hot Pages
  - Parallel Stop-and-Copy Operations

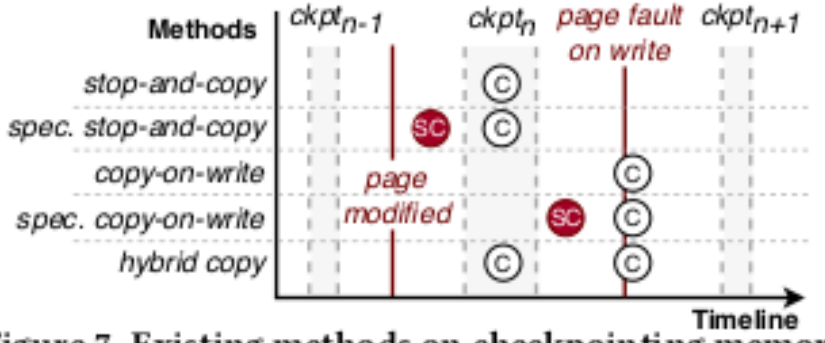


Figure 7. Existing methods on checkpointing memor pages. "C" stands for memory page copy and "SC" represent speculative page copy. The *hybrid copy* in TreeSLS leverage the idea of *speculative copy-on-write* and additionally use *stop-and-copy* for hot pages that are migrated to DRAM.

Extended Versioning for Hybrid Copy

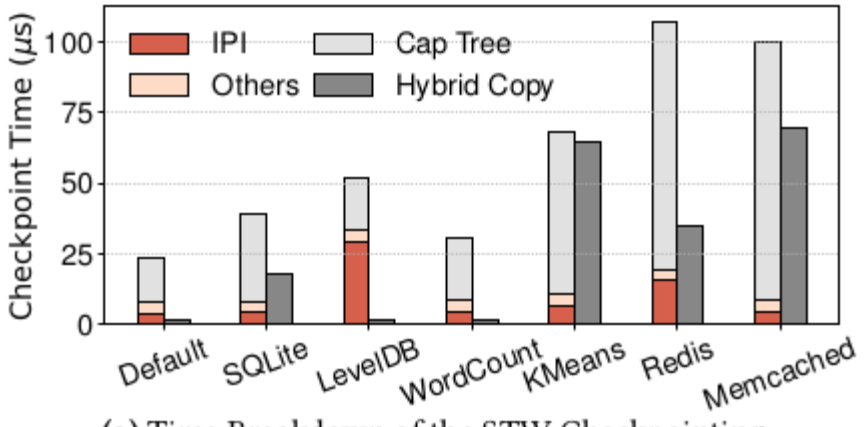
Migration Processes

- NVM-to-DRAM Migration
  - DRAM page allocation, and runtime page's data is copied to it
  - Runtime page table is updated to make the DRAM page the new runtime page
  - Version of the runtime page in NVM is set to the global version
- DRAM-to-NVM Migration
  - Second backup's version is set to zero, and the runtime page table is updated to make the second backup the new runtime page.

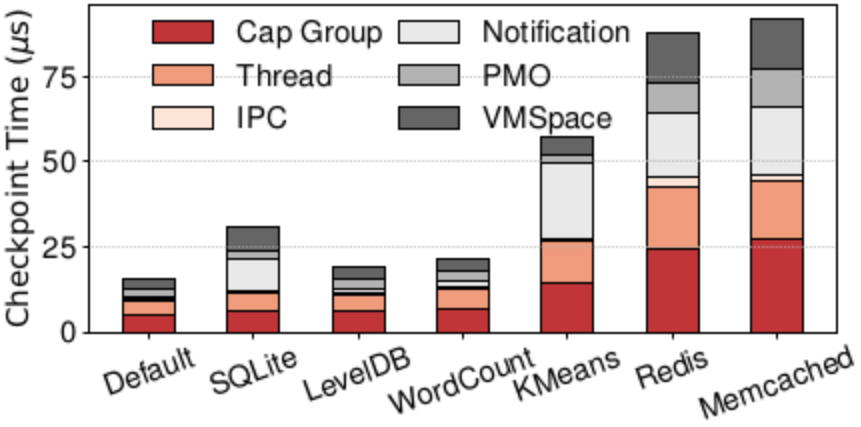
Transparent External Synchrony

- TreeSLS employs high-frequency checkpointing, delaying external visible operations until a checkpoint is taken.
- User-space network drivers, for instance, register a checkpoint callback at the end of each checkpointing.

Evaluation: Stop-the-world Checkpointing



(a) Time Breakdown of the STW Checkpointing



(b) Breakdown of Checkpointing Capability Tree

Evaluation: Runtime Overhead

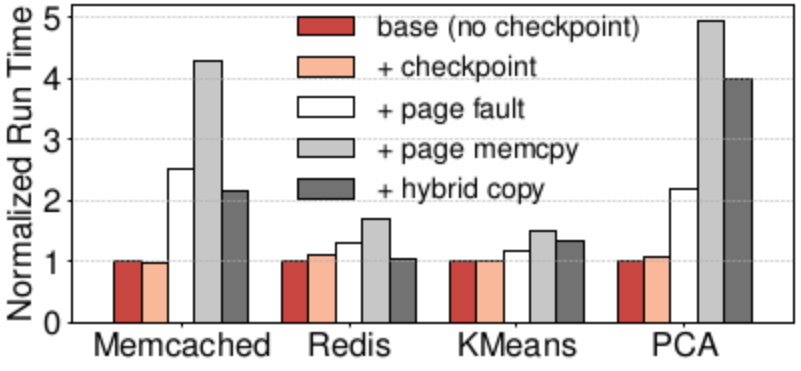
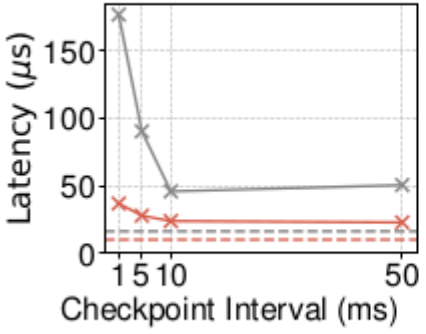
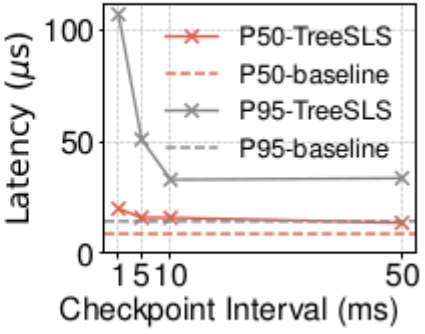


Figure 10. Breakdown of runtime overhead and effect of hybrid memory checkpoint.



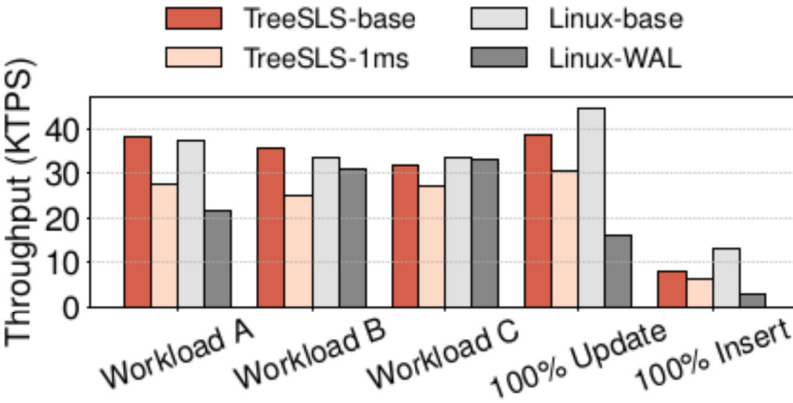
(a) SET



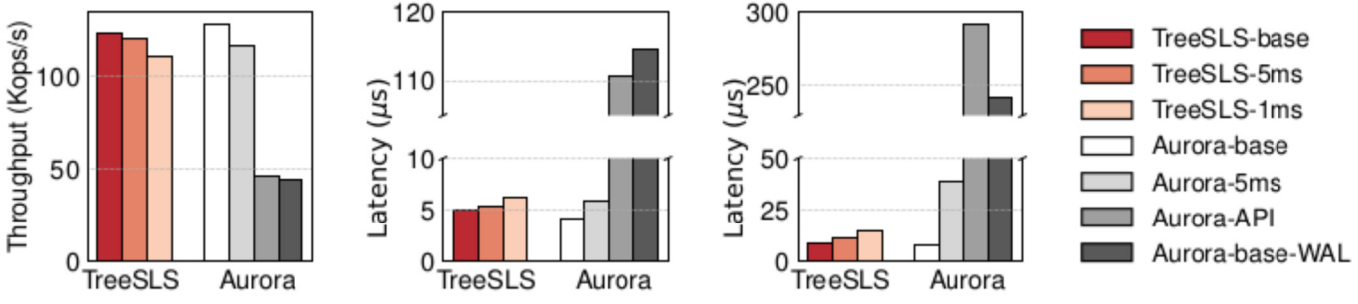
(b) GET

Evaluation: Real World Applications

- In-memory Key-Value Stores



- Persistent Key-Value Stores



(a) Throughput (b) P50 Write Latency (c) P99 Write Latency (d) Legend

Advantages of TreeSLS

- Reduced Recovery Time
- Real-World Applicability
- Reduced Overheads
- External Synchrony Support
- Minimized Disruption to Workflow
- Reduced Development and Maintenance Efforts

Disadvantages of TreeSLS

- Performance Trade-offs
- Dependencies on NVM
- Uniformed Persistence
- Data Reliability
- Memory Over-commitment
- Extension to Eidetic System

Thank you!