ReVive: Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors

MilosPrvulovic, Zheng Zhang, JosepTorrellas
University of Illinois at Urbana-Champaign
Hewlett-Packard Laboratories

Isaac Liu

Introduction

- Targeting large scale applications that provide services (need high *availability*)
- Improvements in silicon technology make modern integrated circuits prone to transient and permanent faults
- FER vs. BER
  - Hardware redundancy vs. recovery

ReVive design

- Goal: Cost-effective general-purpose rollback recovery
  - Modest amount of hardware (cost-effective)
  - Recovery from a wide class of errors (General-purpose)
  - Short system downtime due to error (high availability)
  - Low overhead when error-free (high performance)

Hardware Modifications

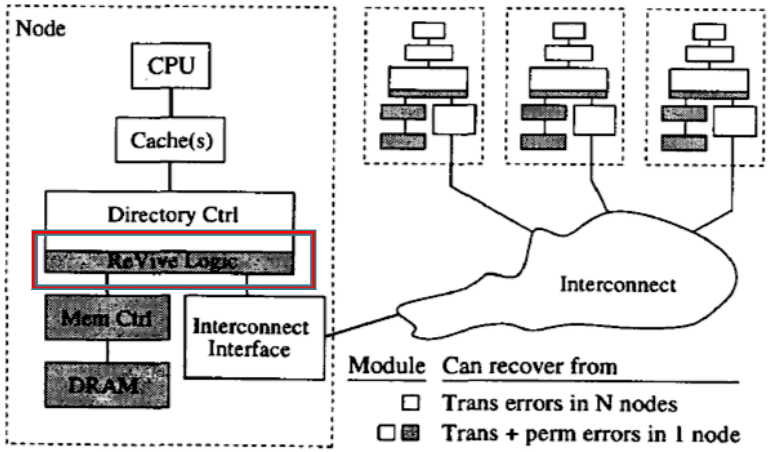


Figure 2. Scalable shared-memory multiprocessor with ReVive.

Design Choices

- Checkpoint Storage:
  - **Safe Internal Storage with Distributed parity**
  - Safe External
  - Specialized fault class
- Checkpoint Separation:
  - **Partial separation with Logging**
  - Full separation
  - Partial separation with buffering (renaming)
- Checkpoint Consistency:
  - **Global**
  - (Un) Coordinated Local

Overview

- Periodically establish checkpoint
- Between checkpoints, whenever main memory written to, log the data to maintain checkpoint state.
- If error is detected, then use the logs to roll back state.

Design Choices

- **Checkpoint Storage:**
  - **Safe Internal Storage with Distributed parity**
- Checkpoint Separation:
  - Partial separation with Logging
- Checkpoint Consistency:
  - Global

Distributed Parity

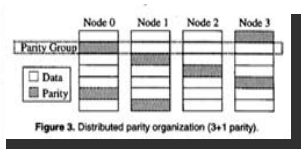


Figure 3. Distributed parity organization (3+1 parity).

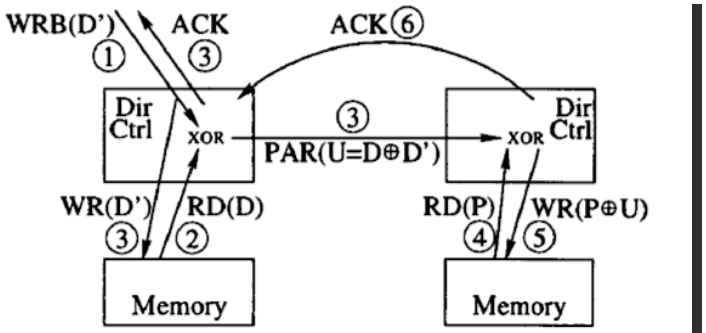


Figure 4. Distributed parity update on a write-back. Messages are numbered in the chronological order.

Design Choices

- Checkpoint Storage:
  - Safe Internal Storage with Distributed parity
- **Checkpoint Separation:**
  - **Partial separation with Logging**
- Checkpoint Consistency:
  - Global

Logging

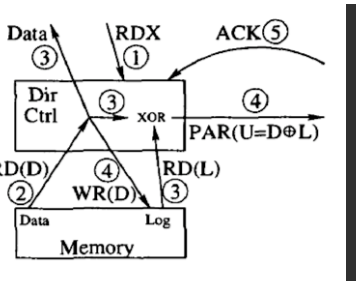
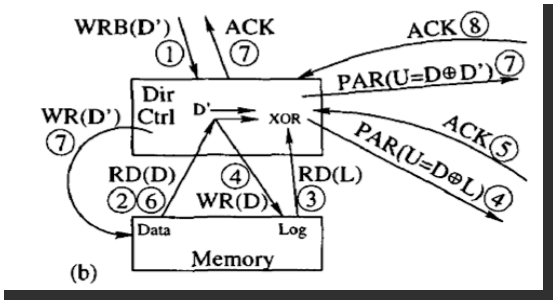


Figure 5. Logging and parity updates for (a) read-exclusive and (b) write-back access to a line that has not already been logged since the last checkpoint. Only the home node of both the data and the log is shown.



Design Choices

- Checkpoint Storage:
  - Safe Internal Storage with Distributed parity
- Checkpoint Separation:
  - Partial separation with Logging
- **Checkpoint Consistency:**
  - **Global Checkpoint**

Global checkpoint

- Commit all work and states to main memory.
- Two phase commit protocol, first sync is tentative commit, and then sync again to fully commit.
- Keeps two most recent checkpoints.

Global Checkpoint

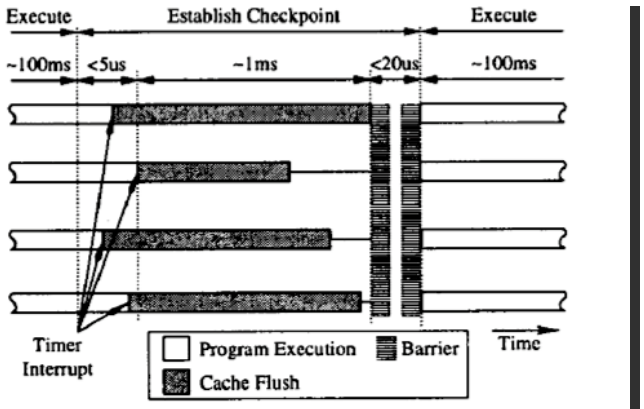


Figure 6. Time-line of establishing a global checkpoint.

Implementation issues

- Extra L bit for each directory entry
- New states in directory protocol, new messages (parity update/ack)
- Race Conditions
  - Log-Data Update race
  - Atomic Log Update Race
  - Log-Parity Update Race
  - Data-Parity Update Race
  - Checkpoint commit Race

Rollback

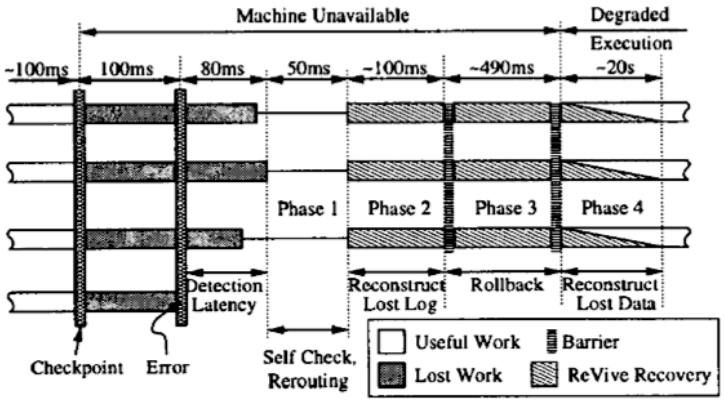


Figure 7. Time-line of recovering from node loss.

Overhead

- Logging and parity maintenance
  - Depends on application
- Global Checkpoint
  - cross-processor interrupt
  - Write dirty data to memory
- Rollback
  - Recovery + Lost work + Rebuild lost memory pages

Evaluation environment

- CC-NUMA multiprocessor with 16 nodes
- Non-blocking and write-back cache
- Full-map directory and cache coherent protocol similar to DASH.
- Cache size:
  - 16KB for L1, 128kB for L2
- *Applications run on smaller problems sizes and shorter periods

Evaluation Results

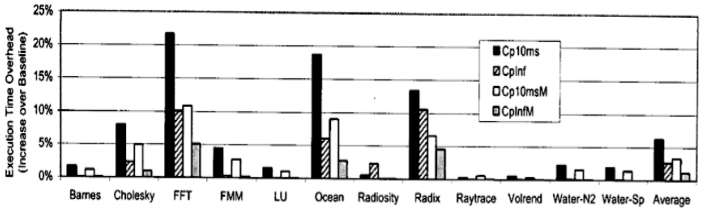


Figure 8. Performance overhead of ReVive in error-free execution.

•Cp10ms – Parity and checkpoint every 10ms
•CpInf – Parity and checkpoint with infinite interval
•Cp10msM – Mirror and checkpoint every 10ms
•CpInfM –Mirror and checkpoint with infinite interval

Traffic

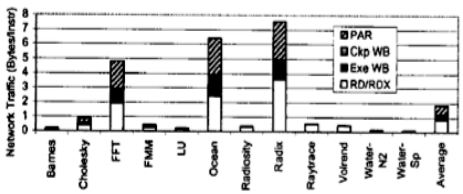


Figure 9. Breakdown of network traffic in Cp10ms.

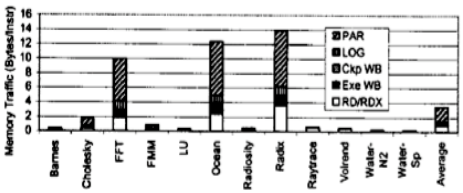


Figure 10. Breakdown of memory traffic in Cp10ms.

•Par – parity updates
•Ckp – checkpoint
•WB – writeback
•RD/RDX- cache miss
•LOG – writing to logs

Overhead

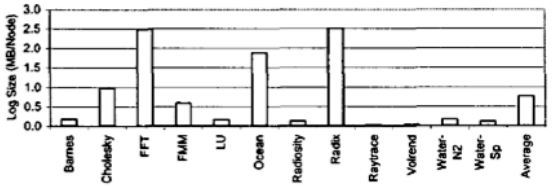


Figure 11. Maximum log size in the Cp10ms configuration.

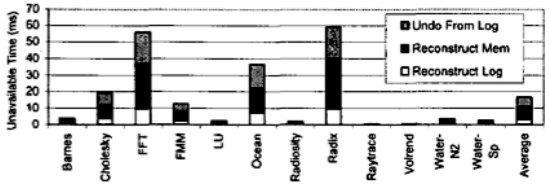


Figure 12. Breakdown of the unavailable time due to an error in the Cp10ms configuration.

ReVive vs. SafetyNet

- Both use log-based rollback mechanisms
- ReVive enables recovery from a permanent node
- ReVive does not need to change processor's cache
- ReVive is more general, so it may result in larger performance overhead.

Conclusion

- ReVive provides:
  - Modest amount of hardware (cost-effective)
  - Recovery from a wide class of errors (General-purpose)
  - Short system downtime due to error (high availability)
  - Low overhead when error-free (high performance)