
Guanpeng Li, Karthik Pattabiraman, Chen-Yong Cher and Pradip Bose
Soft Errors

Bauman, T-DMR[1]
Traditional Checkpoint

Roll Back to Checkpoint

Every 2 days

T0

T1

Fault Occurrence

T2

Failure
Checkpoint at High Frequency

System is unrecoverable!

Every tens of thousand instructions

Fault Occurrence

Failure

T1

T2

ReVive[2]
SafetyNet[3]
Fault Model

• Transient errors occurred in computation components
• Memory and cache protected with ECC
• Single-bit flip
• Crash-causing faults
Checkpoint

• Periodic checkpoint system

• Saves all architectural states
Checkpoint Corruption

Checkpoint CorrupNon Rate

Checkpoint Interval (\# of dynamic Instructions)

Checkpoint CorrupNon Rate

Checkpoint Interval (\# of dynamic Instructions)
Traditional Method: DMR

Dual Modular Redundancy (DMR)

• Run 2 copies in program
• Compare for divergence

Too much energy consumption!
Traditional Method: Dual-checkpoint Scheme

- Checkpoints still corrupted at high frequency
- Takes additional memory space

Wang et.al. at TDSC [4]
Aupy et.al. at PRDC [5]
Our Goal

• Keep single checkpoint

• Minimize checkpoint corruptions
Challenges

• Fault propagations are application-specific

• Difficult to reason about error propagation (Huge state space)
Our Approach

- **Static & Dynamic analysis to identify patterns of crashes**
- **Strategically place checkpoints at Quiescent States**
- **Use low-cost duplication technique to protect**

ReCov
Crashes Leading to Checkpoint Corruptions

- Long-latency Crashes (LLC)
- Short-latency Crashes (SLC)

1000 Dynamic Instructions

Propagation Latency

Execution Frequency
SLC Leading to Checkpoint Corruption

a = calc_addr index, 1
...

... = load a

Loop

B1

B2

B3
LLC Leading to Checkpoint Corruption

Our work at DSN[4]

```c
static unsigned int state[N+1];
static unsigned int *next;
...
unsigned int reloadMT(void)
{
    ...
    register unsigned int *p0 = state;
    next = state+1;
    ...
    *p0++ = *pM++ ^ ...;
    ...
}
...
unsigned int randomMT(void)
{
    unsigned int y;
    ...
    y = *next++;
    ...
}
...
```

[From sjeng program]
Our Approach

Static & Dynamic analysis to identify patterns of crashes

Strategically place checkpoints at Quiescent States

Use low-cost duplication technique to protect

ReCov
Quiescent States for SLCs

\[a = \text{calc_addr} \text{ index}, 1\]

\[\ldots\]

\[\ldots = \text{load a}\]

B1

B2

B3

Quiescent State

Loop
What We Do

- Static & Dynamic analysis to identify patterns of crashes
- Strategically place checkpoints at Quiescent States
- Use low-cost duplication technique to protect

ReCov
Protection of LLCs

Duplicate backward slices of chosen instructions and insert a checker at the end
ReCov

Download: https://github.com/DependableSystemsLab/ReCov

LLVM Compiler Infrastructure

Static & Dynamic Analysis → Duplication of LLCs → Identification of Quiescent States

Program Source Code → Program Executable
Research Questions

RQ1: How much does ReCov reduce the checkpoint corruption?

RQ2: What are the performance overheads incurred by ReCov?

RQ3: How much reduction in unavailability does ReCov provide?
Experiment

• Benchmarks
  • 8 applications from 4 suites: Parsec, Parboil, SPLESH-2 & SPEC
  • 2 open source applications: PureMD and Hercules

• 5 applications for our initial study, 10 in total for evaluation

• Periodic single checkpoint scheme as baseline

• 3000 Fault Injections per checkpoint interval (Error Bar: 0.06% - 0.6%)

• LLVM Fault Injector (LLFI) -> https://github.com/DependableSystemsLab/LLFI
ReCov: Minimize Checkpoint Corruption

SLC Coverage: 87% on average
LLC Coverage: 96% on average
RQs

RQ1:
How much does ReCov reduce the checkpoint corruption?

RQ2:
What are the performance overheads incurred by ReCov?

RQ3:
How much reduction in unavailability does ReCov provide?
Protection of LLCs

Amount of dynamic instructions protected: 9.44% on average

Average Runtime Overhead: 5.03%
RQs

RQ1: How much does ReCov reduce the checkpoint corruption?

RQ2: What are the performance overheads incurred by ReCov?

RQ3: How much reduction in unavailability does ReCov provide?
Unavailability

\[
\text{Availability} = \frac{MTTF}{MTTF + MTTR}
\]

Unavailability = 1 - Availability

- 8.25 times reduction compared to baseline
- 6.2 times reduction compared to dual-checkpoint
Summary

• Checkpoint corruptions are non-negligible at high-frequency checkpointing

• 2 patterns leading to checkpoint corruptions: SLC & LLC

• Quiescent states to place SLC to avoid checkpoint corruptions

• Protection of LLCs: ~5% overhead

• ReCov: Single checkpoint scheme that reduces ~8 times unavailability

  gpli@ece.ubc.ca

  https://github.com/DependableSystemsLab/ReCov