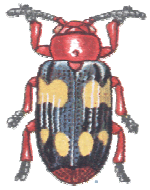
# *Flashback*: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging

Sudarshan Srinivasan, Srikanth Kandula,
Christopher Andrews and Yuanyuan Zhou

# Software Bugs are Expensive!

- Software bugs
  - Inevitable in complex systems
  - Account for 40% of system failures
  - Cost $59.5 billion annually, 0.6% of the US GDP!
  - 50-75% of software development cost

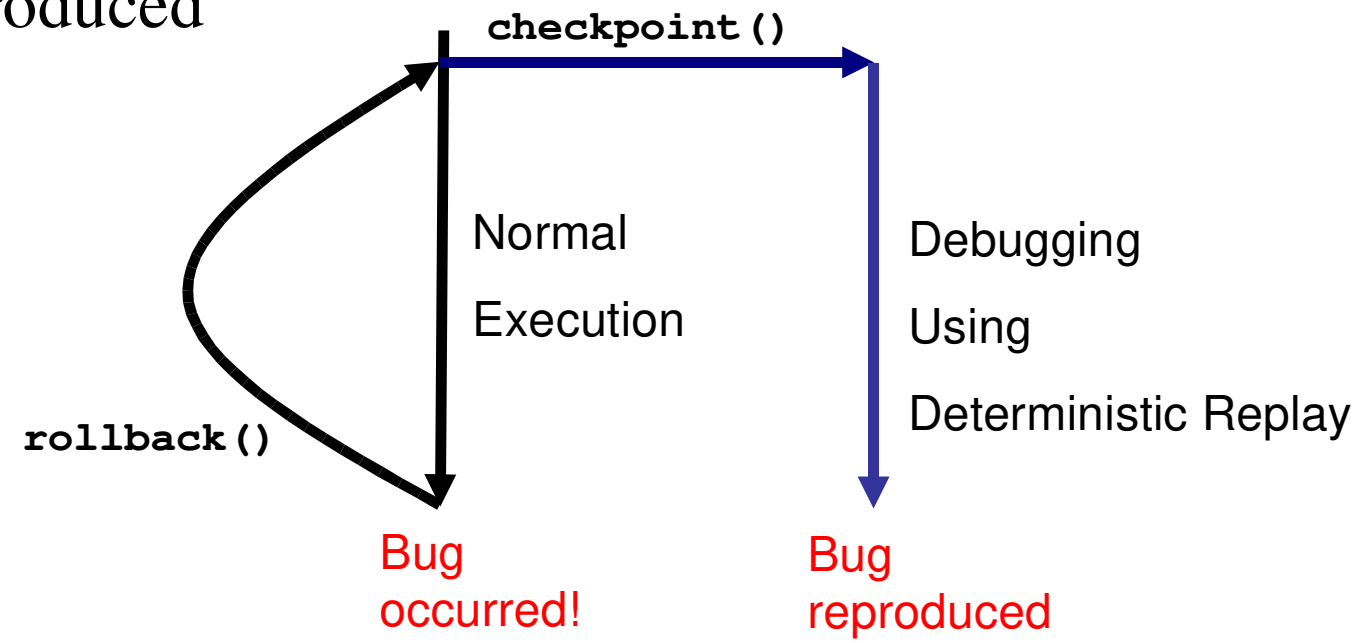


# Software Debugging is Hard!

- Number of bugs is on the rise
  - On average,100 defects per thousand lines of code
- Hard, if not impossible, to test everything
  - Too many  possible execution paths
- Existing debugging tools useful but limited
  - Interactive debugging is still the most widely used method
- Some bugs are dynamic (hard to reproduce)
  - Configuration and running environment dependent
  - Unexpected user interaction

# One Possible Approach – Rollback and Replay

- Roll back the process' execution to a previous point in time
- Replay execution deterministically so that the bug is reproduced



# Current State-of-Art

- Support for rollback
  - Checkpoint systems were designed for system failures, too heavy-weight for debugging
  - Transaction systems: require applications to be transaction-aware
- Support for deterministic replay
  - High-level protocols have been proposed, but never implemented, so many issues are not addressed
  - Most are designed for fault-tolerance (as long as it does not appear different for outside)---- does not work for debugging

# What is *Flashback*?

- Lightweight OS support for rollback and deterministic replay, specifically designed for software debugging

- Implemented as:
  - an extension to the Linux kernel
  - modifications to gdb to support checkpoint and replay

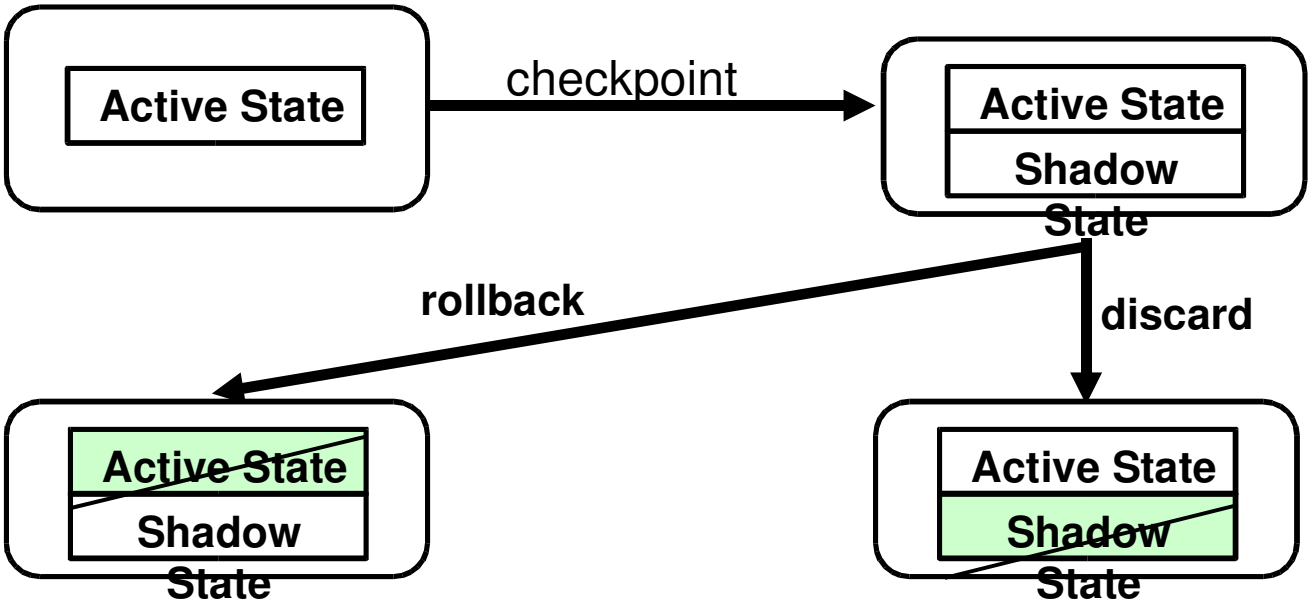
# In this talk…

- Motivation
- Main idea
- Details
- Where we stand
- Performance
- Conclusion

# How Flashback Works – The Main Idea

- Support for rollback
  - Shadow process: Fast in-memory checkpointing of a process' execution state
- Support for deterministic replay
  - Sandboxing: Logging and replaying system calls and events

# Shadow Process

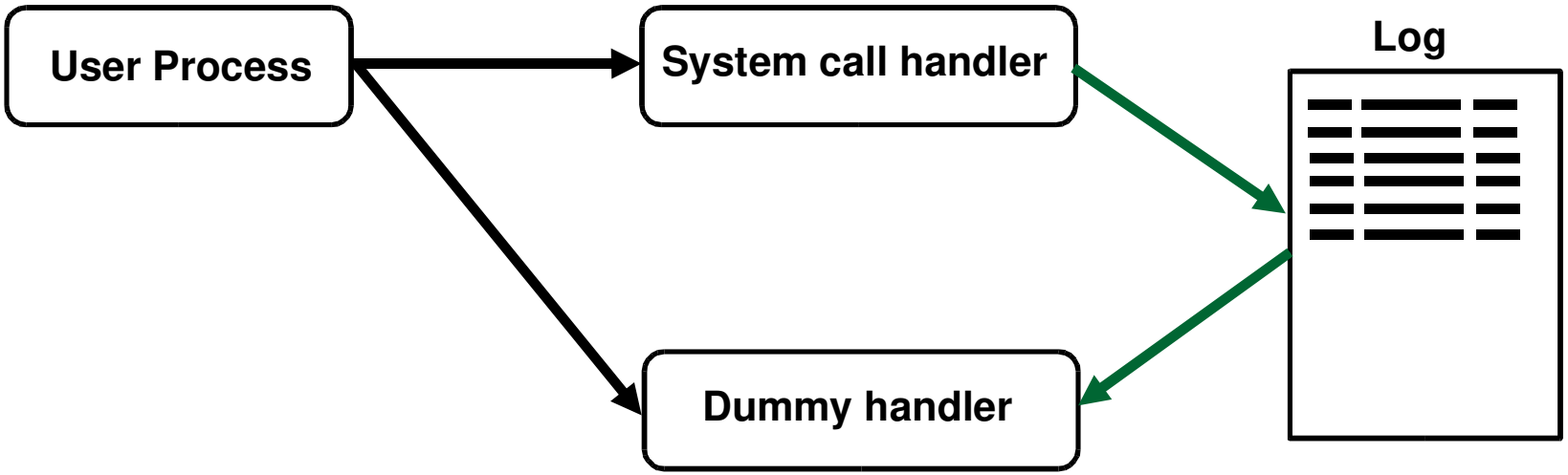



**`Checkpoint:`** capture system state using steps similar to a process fork

**`Discard:`** get rid of the shadow state

**`Rollback:`** reinstate the shadow state by copying the contents of saved process

# Sandbox Mechanism




- Enable logging during original execution
- Use log entries to fake system call replay
- Implemented as an extension to *syscalltrack*

# Challenges

- Rollback
  - Multithreaded applications – inconsistent thread state
- Deterministic replay
  - Memory-mapped files
  - Multithreaded applications - thread interleaving
  - Shared memory accesses
  - Signals and other asynchronous events
  - Asynchronous I/O

# Multithreaded Applications - Rollback

- Problem
  - Choice between single checkpoint for all threads and per-thread checkpointing
  - Some threads may be in the kernel or in sleeping state
- Solution
  - *Flashback* provides single checkpoint functionality
  - Modify process' stack so that system call is re-executed during replay

# Multithreaded Applications – Deterministic Replay

- Problem
  - Need to ensure thread interleaving identical to original execution
- Proposed Solution
  - Record physical thread interleaving at the scheduler and force identical interleaving (similar to Russinovich et al)

# Memory-mapped files

- Problem
  - Interaction between processes outside the system call interface
  - Undetected changes to files
- Possible solutions
  - Per-access bookkeeping
  - File versioning
  - Architecture support
- Current solution
  - Bookkeeping at every page fault to memory-mapped file pages
  - Deterministic replay for private mapping and single writer maps

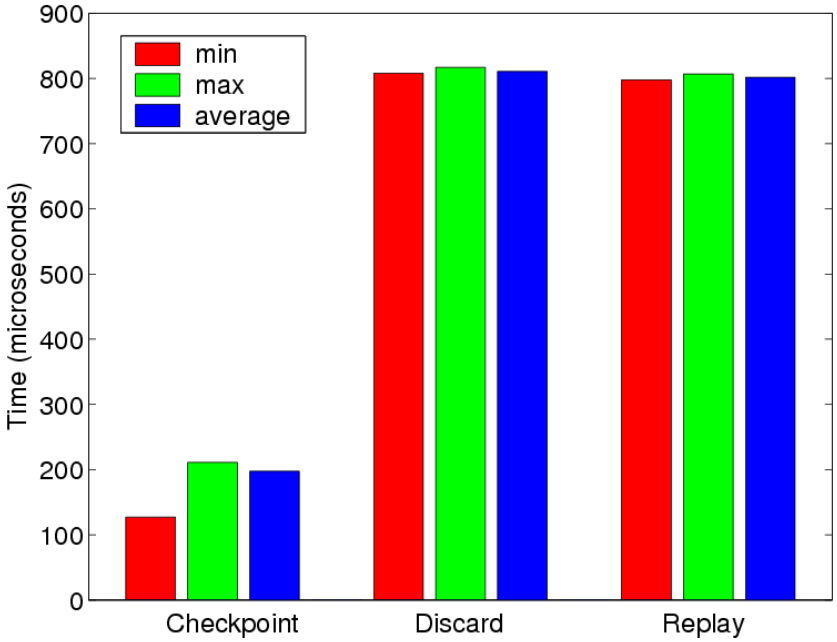
# Changes to gdb

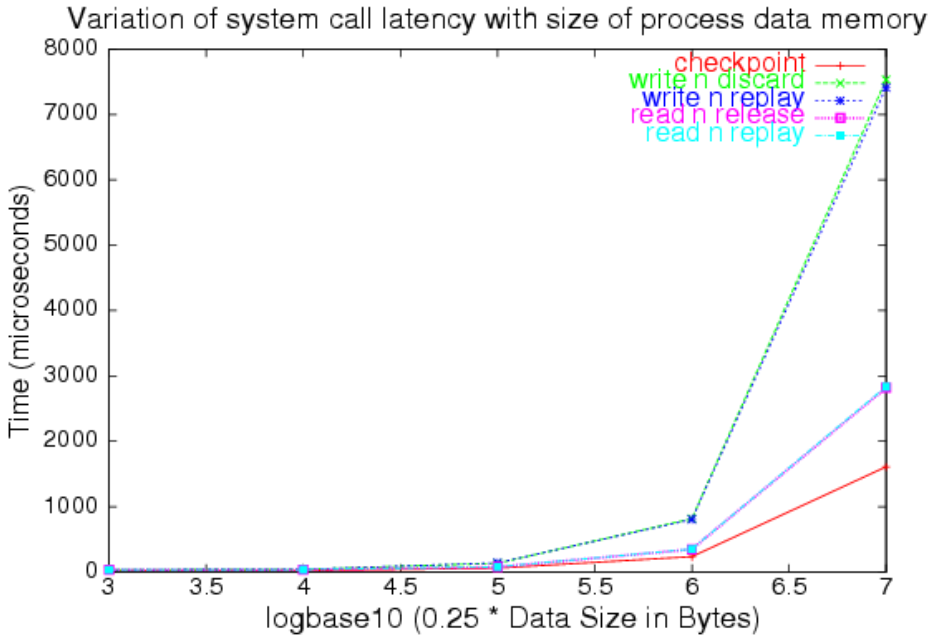
- Frontend
  - New commands – checkpoint, discard, rollback, replay
  - Running program can be rolled back and either *replayed* or *re-executed*
- Backend
  - Invocations to the three primitives made by gdb for checkpoint, commit, rollback
  - Adding and removing of rules for deterministic replay if required

# Experimental Methodology

- Setup
  - Pentium III 1.8 GHz processor system; 100 Mbps network
- Microbenchmark experiments
  - Overhead of system calls
  - Logging overhead
- Application Testing
  - Apache webserver – multiprocess and multithreaded configurations
  - Webstone load generating application
  - Throughput, response time and CPU load-average measured

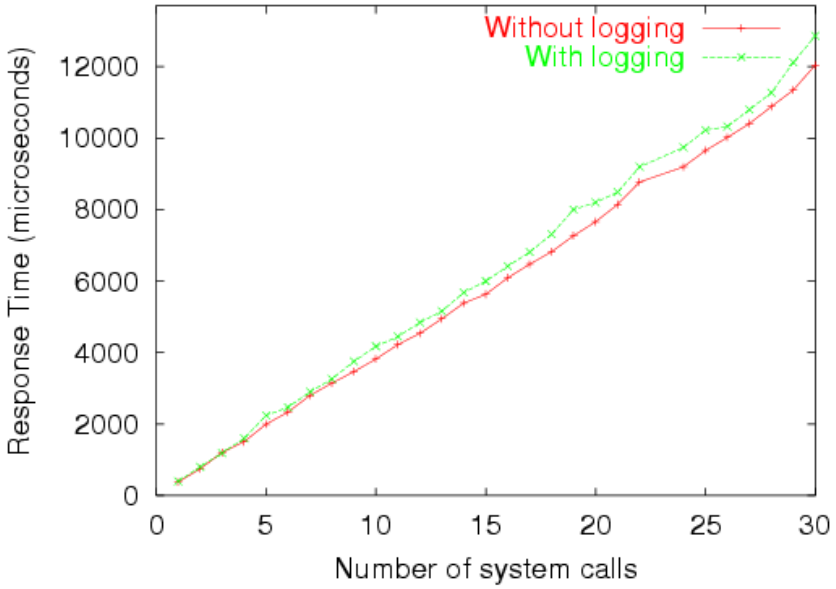
# Overhead of shadow processes



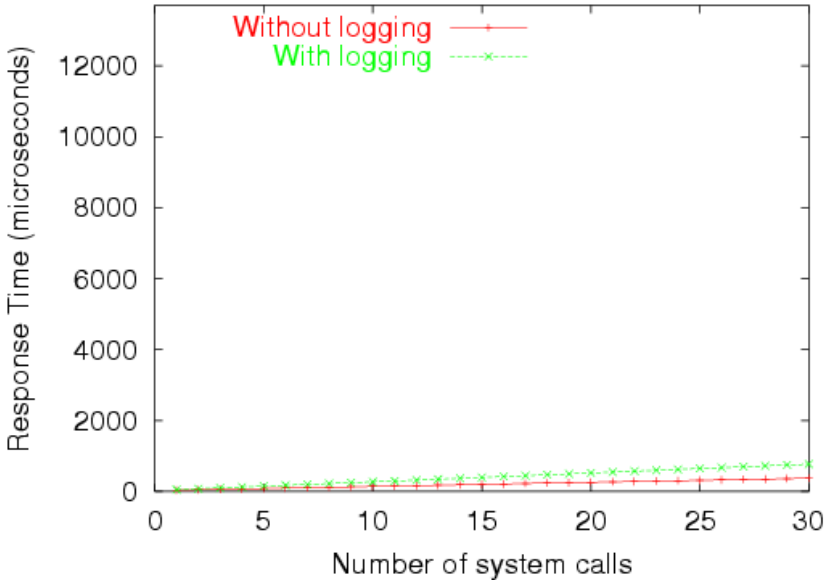




- Overhead depends on footprint of memory operations between checkpoints
- Overheads for 4K memory modifications: `checkpoint take` ~ 25 microseconds, `discard` and `rollback` take ~ 30 microseconds

# Overhead of logging



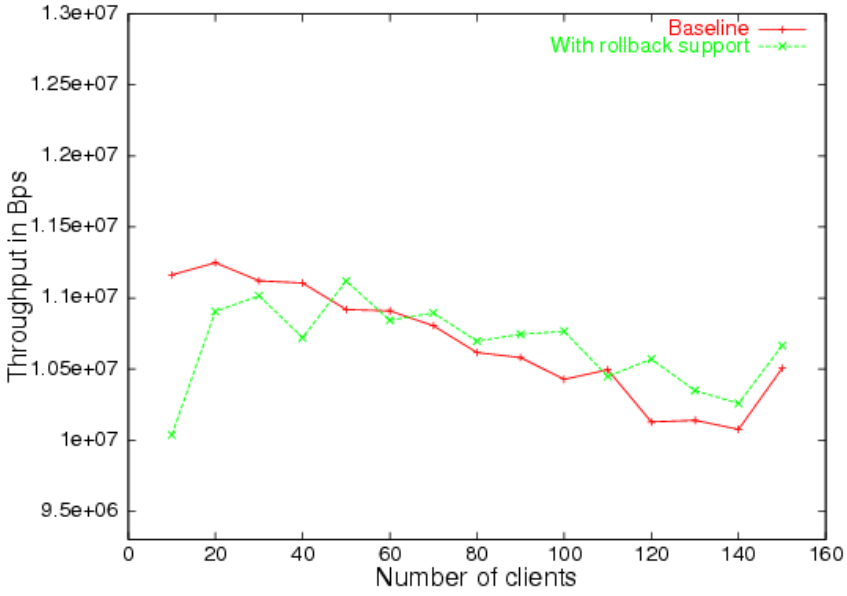

**Buffer copying system calls**



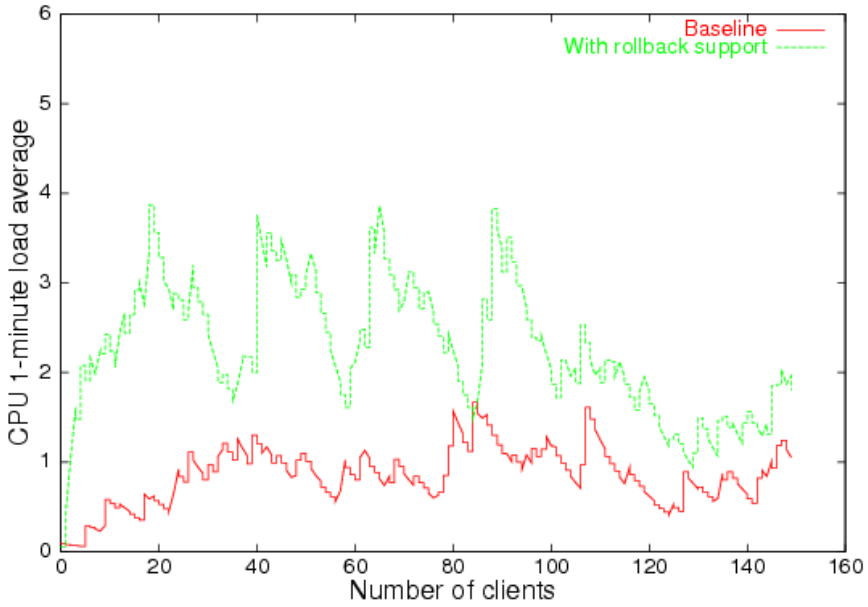

**No-copying system calls**

- Overhead depends on amount of writing involved – varies with system calls
- Size varies linearly with number of system calls

# Application Results




**Throughput for multithreaded mode**




**Load average for multithreaded mode**

- 10% drop in throughput; response time unaffected – reasonably small overhead
- Overhead in multithreaded configuration less than in multiprocess configuration

# Current Status

- Handles rollback (state capture) of in-memory state for single-threaded and multithreaded applications
- Deterministic replay of processes using system calls and memory-mapped files to some extent
- GDB extensions allow checkpoints, rollback, re-execution and replay. Support periodic and on-demand checkpoints

# Concluding Remarks

- *Flashback* provides features that are necessary for cyclic debugging
- Fills a gap in software debugging by addressing the need for deterministic replay
- We will release the code for people to use

# From here…

- Replay of multithreaded applications
- Replay of asynchronous events- signals and I/O
- Extend Flashback for multiprocessor systems
  - Architecture support for replay
- Efficient handling of shared memory using architecture support

# Acknowledgements


- The shepherd, Prof Srini Seshan
- Reviewers for the paper
- Pierre Salverda, Arijit Ghosh, Qing Zhang and Jagadeesan Sundaresan
- Syscalltrack development team


* Code will be available at http://carmen.cs.uiuc.edu

# THE END

# Example Usage Scenarios

- Using a debugger like gdb
  - User turns on checkpointing while running the application in debug mode
  - If execution is buggy, he can go back to an earlier breakpoint and look more carefully at the execution
- For remote debugging
  - User of software notices that program crashes sometimes (non-deterministic because of timing-dependent events); turns on logging
  - Flashback maintains checkpoints and log of system calls
  - Developer gets a bundle of checkpoint + log – can use this to replay non-deterministic crashes

# Issues in Rollback

- How to save the in-memory state of a running process
  - Lightweight
  - Minimum interference with normal execution
  - Simple interface
- How to handle multithreaded applications
  - Rollback of several threads – a distributed snapshot problem?