Time-Travel in Closed Distributed Systems
Anton Burtsev, Prashanth Radhakrishnan, Mike Hibler, Jay Lepreau
University of Utah, School of Computing

Motivation and Goals
- Distributed systems are complex:
  - Non-deterministic
  - Long-running
  - Asynchronous
  - Intricate component interaction
- We need a tool to analyze and debug long-running distributed systems
- Our Goal: Time-travel a network of thousands of virtual machines spread across multiple physical machines in the Emulab testbed environment

What’s New?
- Existing solutions employ only deterministic time-travel and operate on:
  - A single VM [King et al., USENIX05] or
  - Multiple VMs on one physical machine [Ho et al., GRID04] or
  - Specific user applications across physical machines [Geels et al., USENIX06]
- Our work differs in the following ways:
  - It is designed to be a practical system that time-travels multi-node experiments in “Emulab Classic”
  - We emphasize system scalability. Some design decisions towards this end include:
    - VMs distributed across physical machines
    - “Closed world” assumption
    - Relaxed determinism
    - Cooperative replay
    - We allow state mutations during replay by non-determinism

Emulab – The “Closed World”
- Experiments in Emulab Classic are typically “closed” - nodes within an experiment tend to communicate only with each other
- “Closed world” assumption helps improve scalability and enables exploring relaxed determinism
- Emulab’s reliable, low-latency network fabric for control plane helps simplify the overall system by bounding clock skews to microsecond range [Veitch et al., IMC04]
- Emulab automation makes it easy to setup large-scale experiments

Why Relax Determinism?
- Mutating debugging operations are incompatible with deterministic replay
- Non-deterministic time-travel reduces logging overhead
- Some applications may not need determinism for debugging

Implementation
- Uses existing Xen VMM technology
- Implements both non-deterministic and deterministic time-travel in an attempt to compare them
- Logs sources of non-determinism during original execution
  - Timer Interrupts, Disk I/O, Network I/O, Physical Time
- Employs cooperative logging
  - Obviates logging of packet contents
- Takes periodic consistent distributed checkpoints for efficient time-travel
  - Tag all network packets with checkpoint "epoch-id"
- Replay
  - Deterministic: uses branch counters
  - Non-deterministic: uses VM virtual time

Current Status, Future Work, and Conclusions
- Prototype non-deterministic time-travel is working
- LVM for disk checkpointing
- Xen mechanisms for memory checkpointing
- Future work:
  - Implementing deterministic time-travel
  - Making disk checkpointing efficient by leveraging versioning filesystem research work, like Ventana [Pfaff et al., NSDI06] and Parallax [Warfield et al., HOTOS05]
  - Making memory checkpointing efficient by implementing CoW memory
  - Evaluating deterministic vs. non-deterministic time-travel
- Research questions:
  - Is non-deterministic replay useful for many applications?
  - Does non-deterministic replay have performance advantages?
  - What is the overhead of time-traveling a network of virtual machines?