ROOT: Replaying Multithreaded Traces with Resource-Oriented Ordering

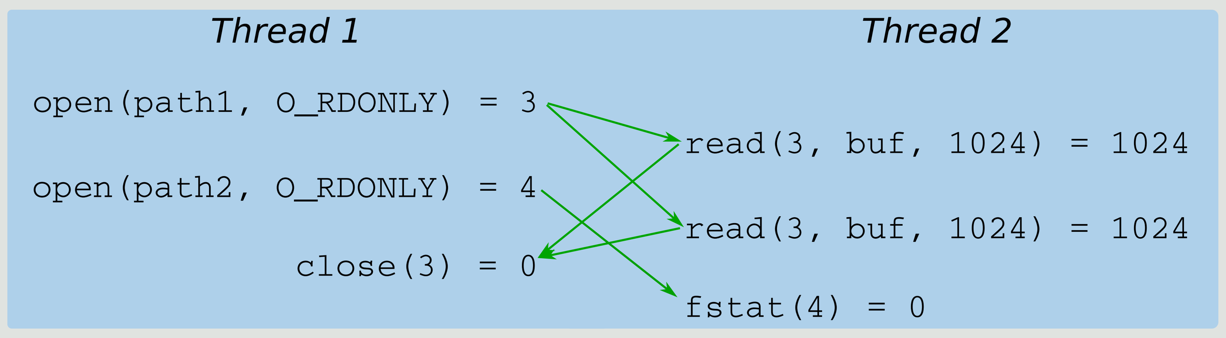
Zev Weiss, Tyler Harter, Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau
{zev,harter,dusseau,remzi}@cs.wisc.edu
University of Wisconsin-Madison, Department of Computer Sciences

Motivation

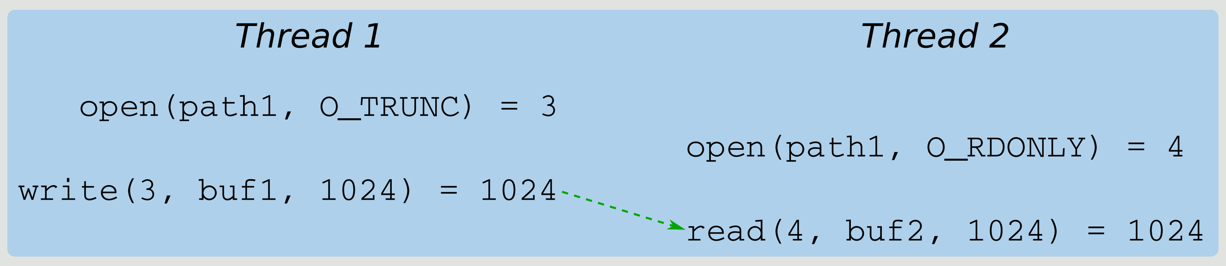
- Why I/O trace replay?
  - A useful tool for storage benchmarking
  - Real workloads offer a more informative benchmark than synthetic ones
- Starting point: SOSP11, "A File is not a File..."
  - iBench: 34 Apple desktop application system-call traces
  - Interesting challenge: how to replay?

Challenges

- Heavy use of **threads** in modern applications makes trace replay much trickier
- Interactions between threads have major effects on the performance and correctness of replay attempts



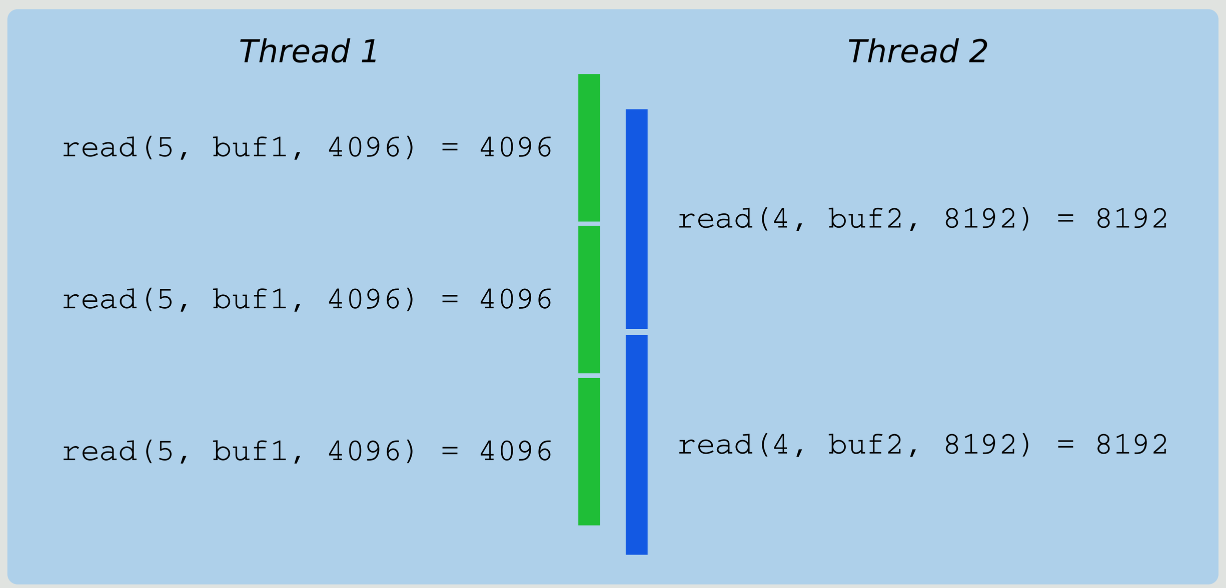
Green arrows indicate ordering dependencies that replay must preserve for correctness.



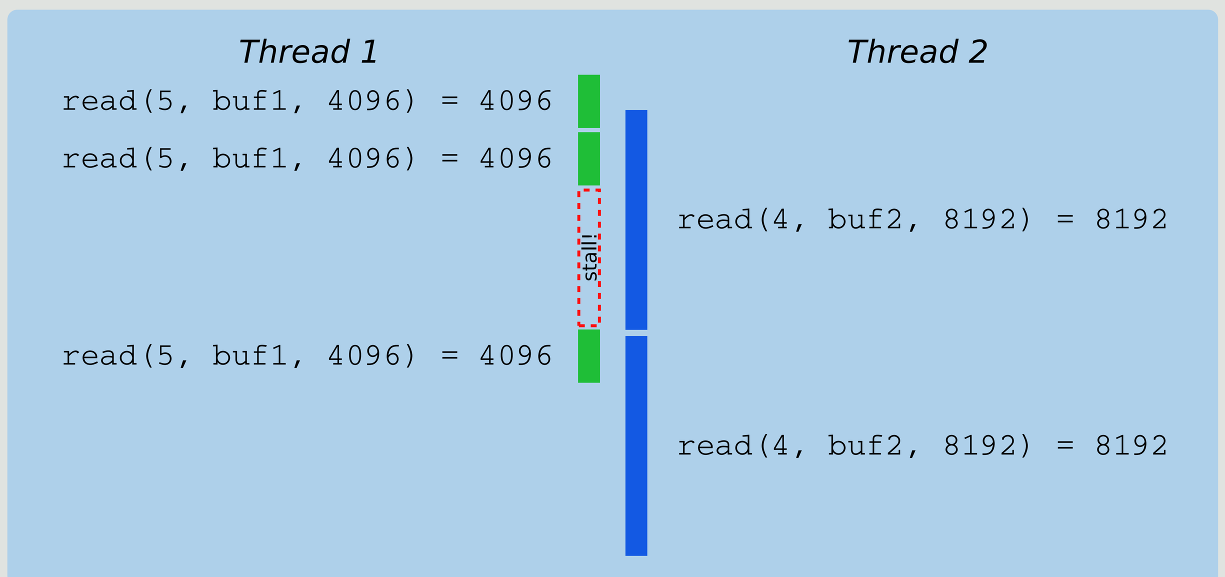
Here there is a more subtle dependency between Thread 1's `write` and Thread 2's `read`.

A Simple Solution

- One easy answer to ordering problems: simply preserve the ordering of the original trace.
- This is very pessimistic; it assumes dependencies *are* present everywhere they *could* be.



A trace with system call overlap.

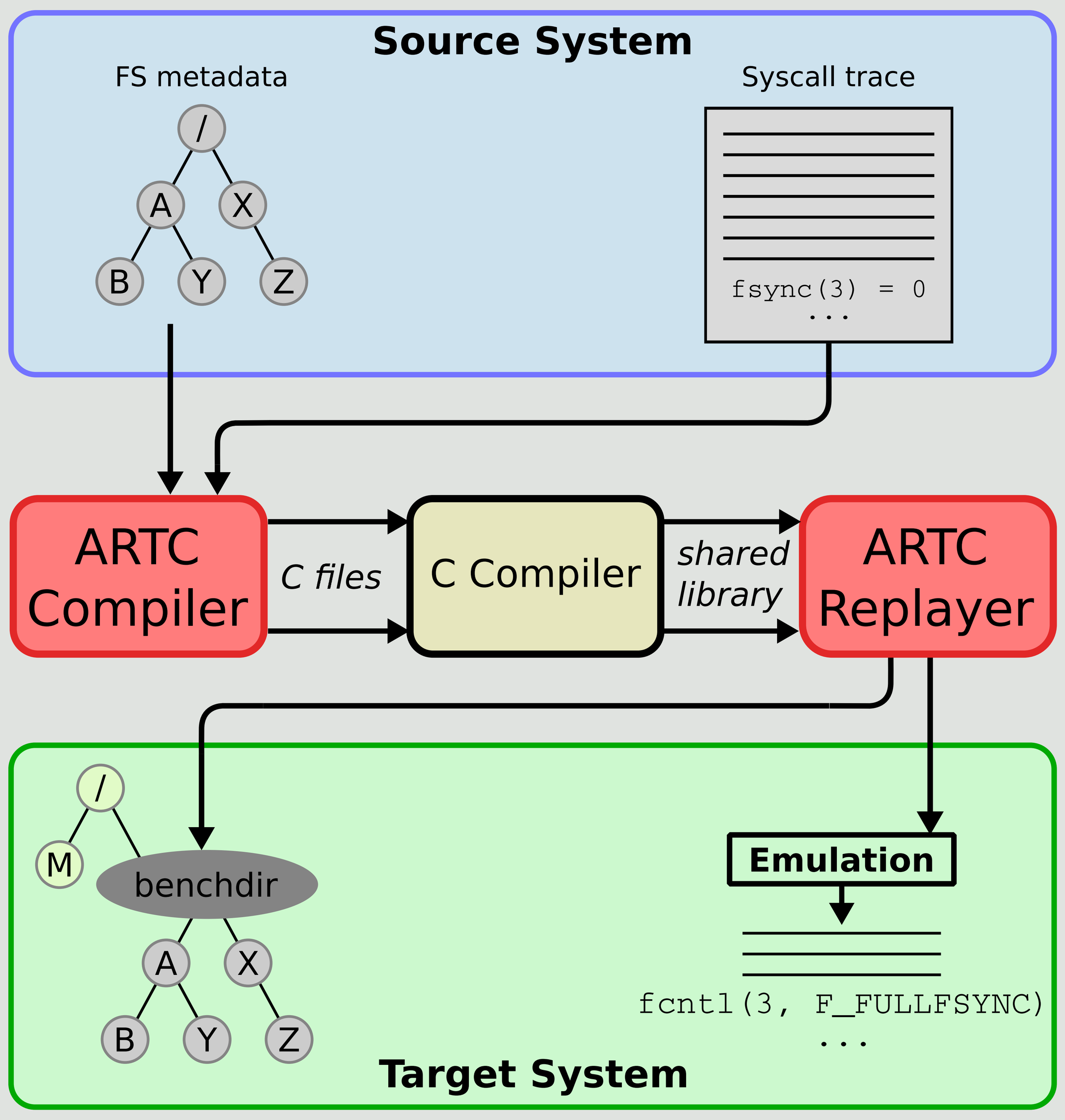


On a system with different performance characteristics, an overly simplistic, order-preserving replay may be forced to insert artificial stalls.

ROOT

- **R**esource-**O**riented **O**rdering for **T**race replay
- Basic idea:
  - Observe ordering of subset of trace events involving each resource
  - Resources: paths, file descriptors, files, AIO control blocks
  - Constrain replay to preserve those *partial* orderings
- Allows flexible, nondeterministic reordering

ARTC: ROOT Implemented



ARTC: an Approximate-Replay Trace Compiler.

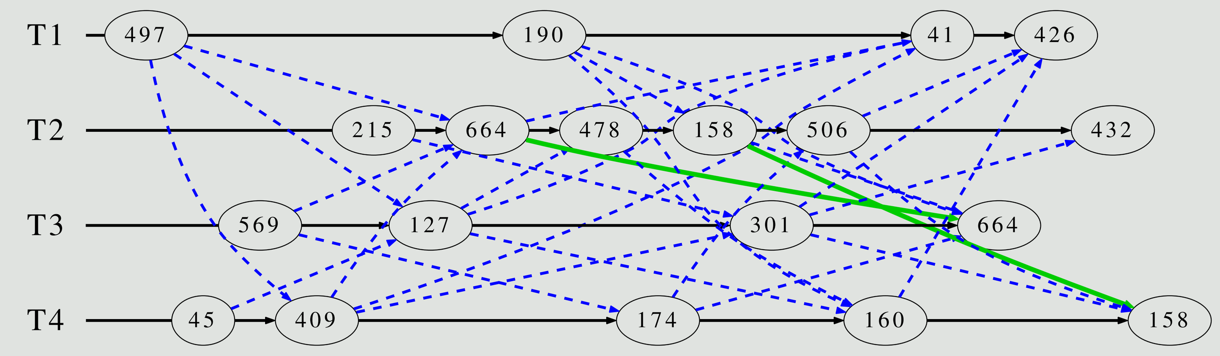
- Compiles a trace and initial FS metadata state snapshot into replayable benchmark
- ~16KLoC (C, Bison, Flex)
- Over 80 system calls supported
- Cross-platform: 
- Emulates non-standard system-calls where necessary
- Reports detailed timing statistics

Evaluation

- Two criteria: semantic correctness and performance accuracy across systems with different performance characteristics
- Workloads:
  - Correctness: Magritte (compiled iBench suite)
  - Performance: four synthetic microbenchmarks, two LevelDB macrobenchmarks
- Alternate strategies:
  - Unconstrained: free-running multithreaded replay
  - Single-threaded: one replay thread for all trace threads
  - Temporally-ordered: multithreaded; constrained to replay in the same order as the original trace

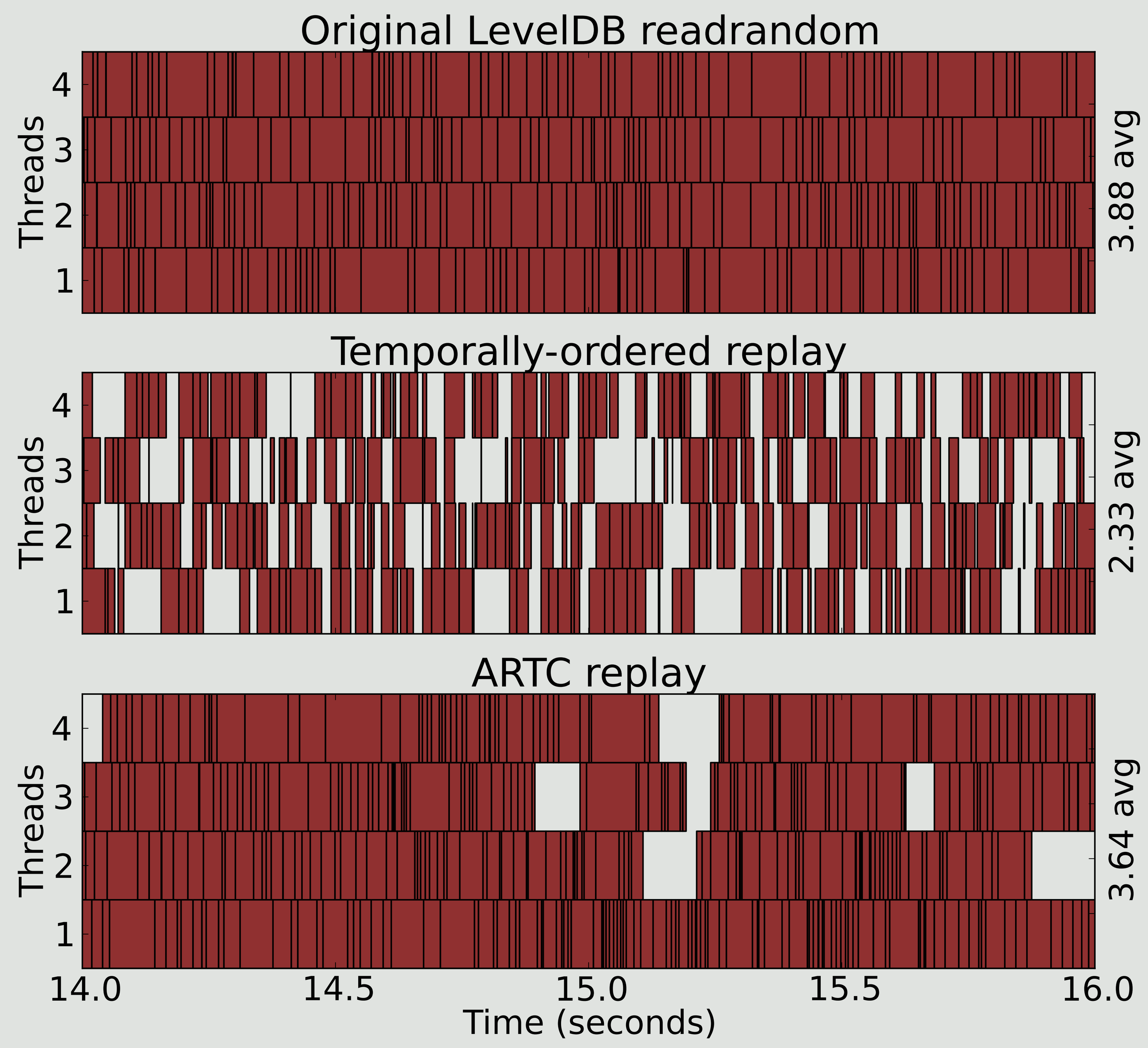
Results

Across workloads, ARTC achieves good replay correctness and much better performance accuracy than simpler replay methods.

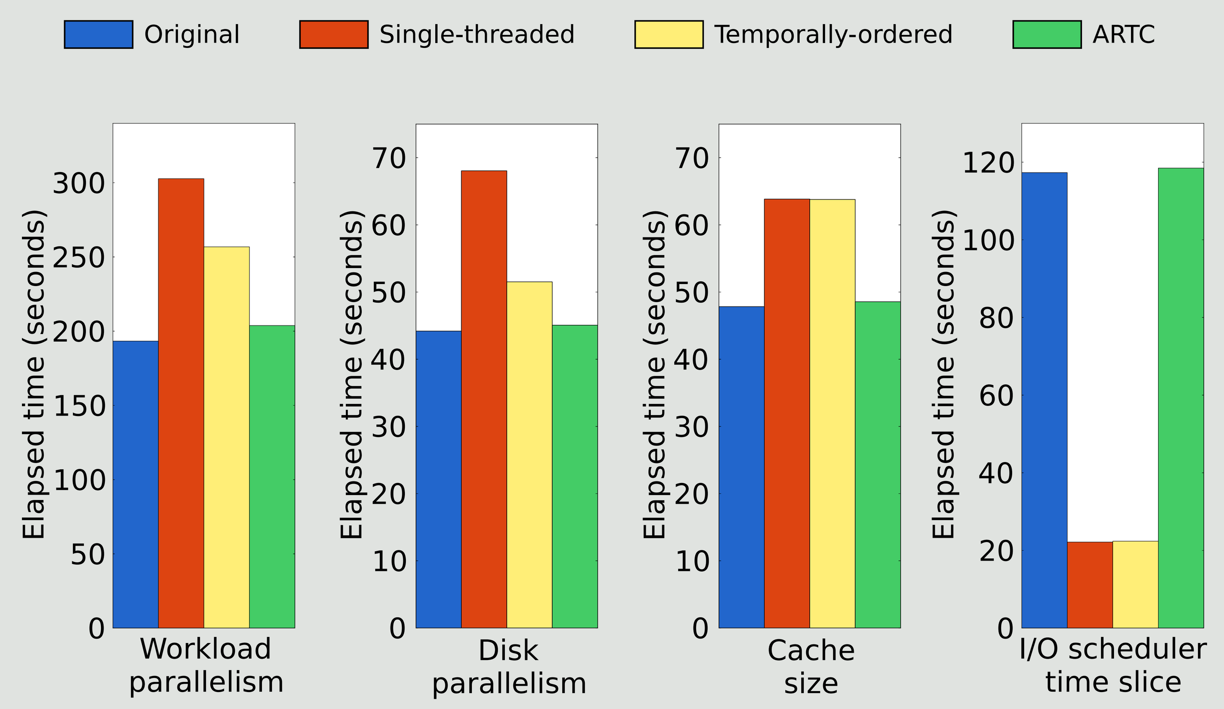


A dependency graph for a four-thread LevelDB random-read trace. The dependencies enforced by ARTC (solid green) are much less restrictive than those necessary to enforce the ordering of the original trace (dashed blue).

ARTC's looser ordering constraints allow it to achieve much more system-call overlap than simpler replay methods:



System call overlap of the original four-thread LevelDB random-read workload and two replays. Temporally-ordered replay achieves only 60% of the original application's system call concurrency, whereas ARTC's replay achieves 90%.



ARTC achieves substantially more accurate performance than simpler replay methods across a variety of workloads and differing system configurations.

Conclusions

ARTC's ROOT-ordered replay provides much more useful replay performance than simpler replay strategies, while preserving correctness.

ARTC and Magritte are available for download at `https://research.cs.wisc.edu/adsl/Software/artc`