//TRACE: Parallel trace replay with approximate causal events

MICHAEL P. MESNIER, MATTHEW WACHS, RAJA R. SAMBASIVAN, JULIO LOPEZ,

JAMES HENDRICKS, GREGORY R. GANGER, DAVID O'HALLARON

CARNEGIE MELLON UNIVERSITY

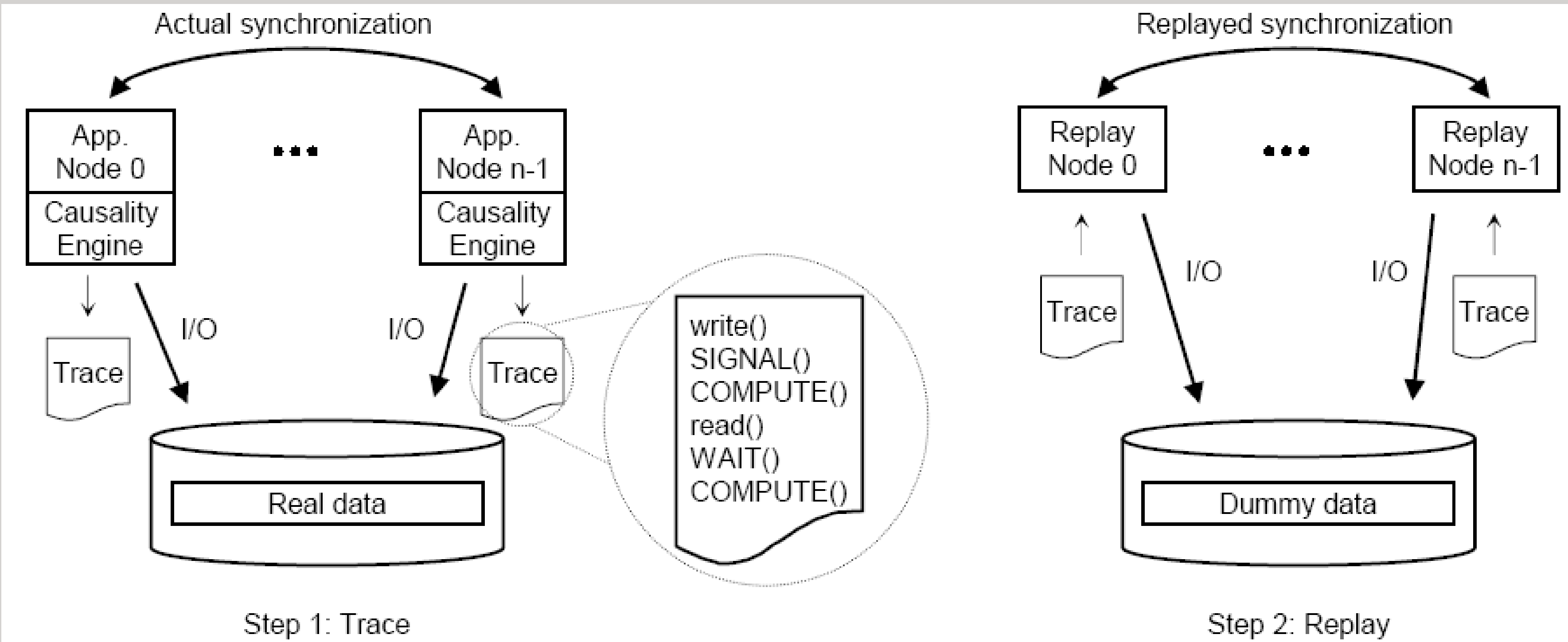
Introduction

- Extract/replay parallel applications to recreate I/O behaviour.
- Discover inter-node data dependencies, inter-I/O compute times per node
- Mimic application behaviour across storage systems

//TRACE

- Black box – no modification to app/storage system
  - Library calls interposed/delayed (LD_PRELOAD) by tracing engine
- Application executed multiple times with artificial delays in "throttled" node
  - Exposes data dependencies
- Execution manager
  - different node throttled on each execution
- Post processing tools – merge traces from multiple runs

High level View



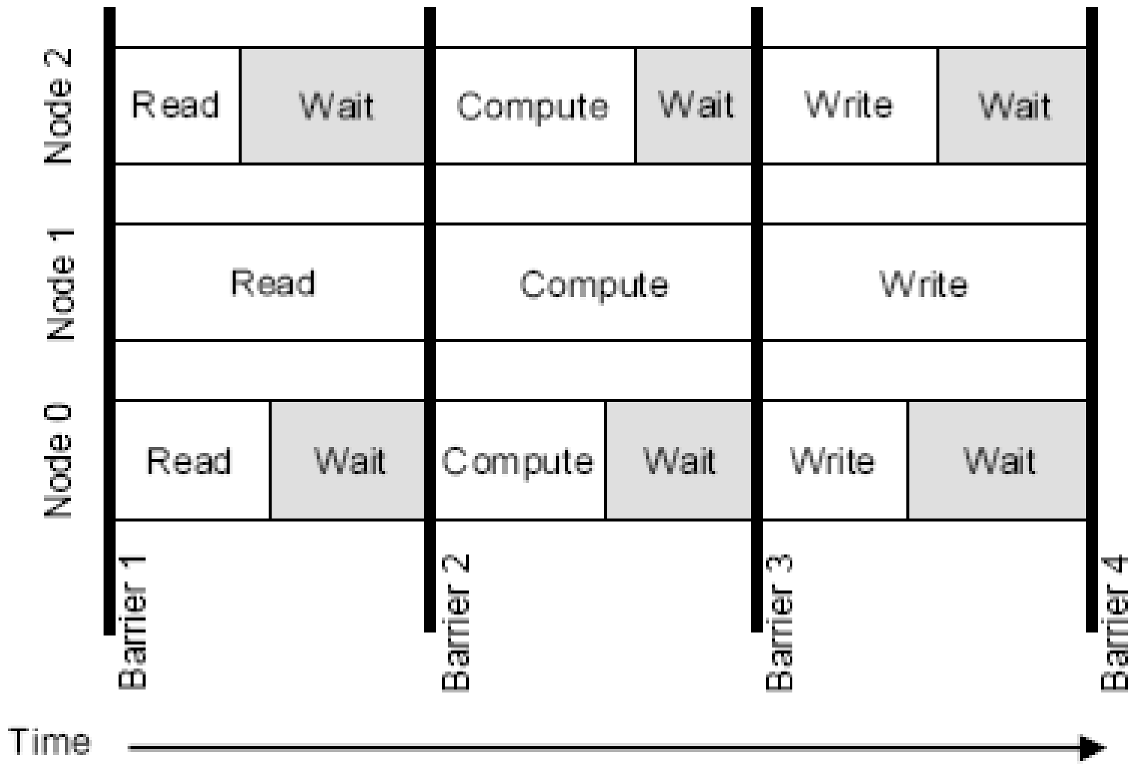
Trace replay models

- Closed model
  - I/O arrivals are dependent upon I/O completions
  - Replay rate determined by think (compute + sync) time and service time of I/O
  - Replay rate dependent on storage system
- Open model
  - Replay rate unaffected by storage system

Synchronization and I/O

I/O is only a fraction of total running time

Wait time depends upon storage system



Design requirements

- adjust with the speed of the storage system
  - traces must be replayed with a *closed model*.
- enforce data dependencies
  - annotated with the inter-node synchronization calls.
- model computation
  - the inter-I/O compute time reflected in the traces.
- evaluate different file systems
  - the traces must be *file-level traces*,

I/O throttling

- Slow down I/O , wait till other nodes exit or block
- Detect whether other nodes are blocked based on CPU, I/O activity
  - throttling node adds a SIGNAL() to its trace
  - blocking node adds a WAIT()
- I/O Sampling, Node sampling
  - Trade-off between accuracy and tracing time

Discover compute time

- Approach 1: Sync time is zero for throttled node.
  - Compute time = think time
  - I/O Sampling can affect calculation
- Approach 2: record time of library/system calls (sync time)
  - Not applicable for "untraceable" synchronization (e.g shared memory)
  - No throttling required
  - Unaffected by sampling

* assumption: I/O synchronous in a thread

Detailed design (contd)

- Causality engine
  - Throttled mode – exactly one node in this mode
  - Unthrottled mode
  - Intercepts and stores computation + signaling/waiting information
    - COMPUTE <seconds>
    - I/O op args
    - SIGNAL/WAIT info (as per sampling period)
  - Delay I/O – RPC sent to watchdog task on unthrottled node
    - Resume I/O on receiving message from each watchdog

Detailed design (contd)

- When is a node blocked?
  - Watchdog checks with causality engine if the node is in computation or synchronization
  - Determine time spent in synchronization system call
  - Considered blocked if time spent exceeds a predetermined maximum
    - System call time is recorded on previous run and increased by a few factors
    - Too small ‘maximum’ can lead to error
    - Too big ‘maximum’ increases tracing time

Trace Replay

- Preparing for replay
  - After m runs – m traces must be merged
  - After merge, each I/O has
    - m - 1 preceding WAIT() calls
    - m – 1 succeeding SIGNAL() calls
    - one COMPUTE() call
- Replay is straightforward
  - file operations replayed as-is on dummy files
  - synchronization – **MPI**, Java, CORBA
  - computation – spinning rather than sleeping (induce CPU load)

Baseline for comparison

- as fast as possible (AFAP) replay
  - ignore think time
- replay think time (*think limited*)
  - more accurate than AFAP
- timing-accurate
  - has identical running time to the application
  - Running time fixed – independent of storage system
- Replay accuracy measure
  - (AppTime – ReplayTime) * 100 / AppTime

Evaluation

- **Hypothesis 1**
  - Data dependencies and computation must be independently modeled during replay, otherwise the replay may differ from the traced application.
- **Hypothesis 2**
  - By throttling every node and delaying every I/O, the I/O dependencies and compute time can be discovered and accurately replayed.
- **Hypothesis 3**
  - Not every I/O necessarily needs to be delayed in order to achieve good replay accuracy. (I/O sampling)
- **Hypothesis 4**
  - Not every node necessarily needs to be throttled in order to achieve good replay accuracy. (node sampling)

Experiment

- Experiment 1 (Hypothesis 1)
  - think-limited vs. application.
    - think limited assumes a fixed synchronization time,
    - We expect high replay error for an application with significant synchronization time
- Experiment 2 (Hypothesis 2)
  - uses the causality engine to create annotated I/O traces. The traces are replayed and compared against think-limited.
- Experiment 3 (Hypothesis 3)
  - uses I/O sampling to explore the trade-off between tracing time and replay accuracy.
- Experiment 4 (Hypothesis 4)
  - uses node sampling to illustrate that not all nodes necessarily need to be throttled in order to achieve a good replay accuracy.

Setup

- VendorA
  - 14-disk (400GB 7K RPM Hitachi Deskstar SATA) RAID-50 array with 1GB of RAM;
- VendorB
  - 6-disk (250GB 7K RPM Seagate Barracuda SATA) RAID-0 with 512 MB of RAM; and
- VendorC
  - 8-disk (250GB 7K RPM Seagate Barracuda SATA) RAID-10 with 512 MB of RAM

Benchmarks

- Pseudo
  - simulates checkpointing of a large-scale computation
  - N processes write a checkpoint file (with interleaved access), synchronize, and then read back the file
  - Pseudo : without any flags specified (),
  - PseudoSync : barrier synchronization after every write I/O
  - PseudoSyncDat2 : sync + computation between every I/O
- Fitness
  - nodes write to file one after the other
- Quake –
  - computation is interleaved with the I/O, and the state of the simulated region is periodically written to disk by all nodes

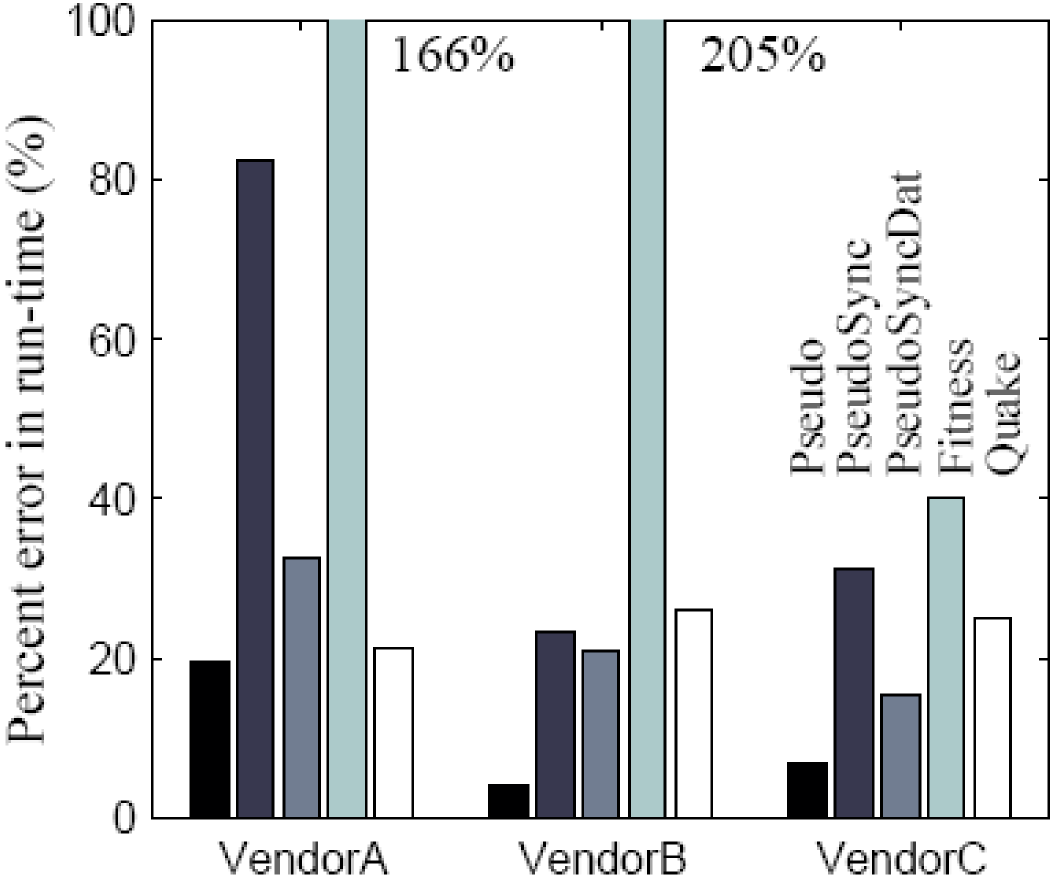
Think-limited (Experiment 1)

Fixed amount of think time between I/O

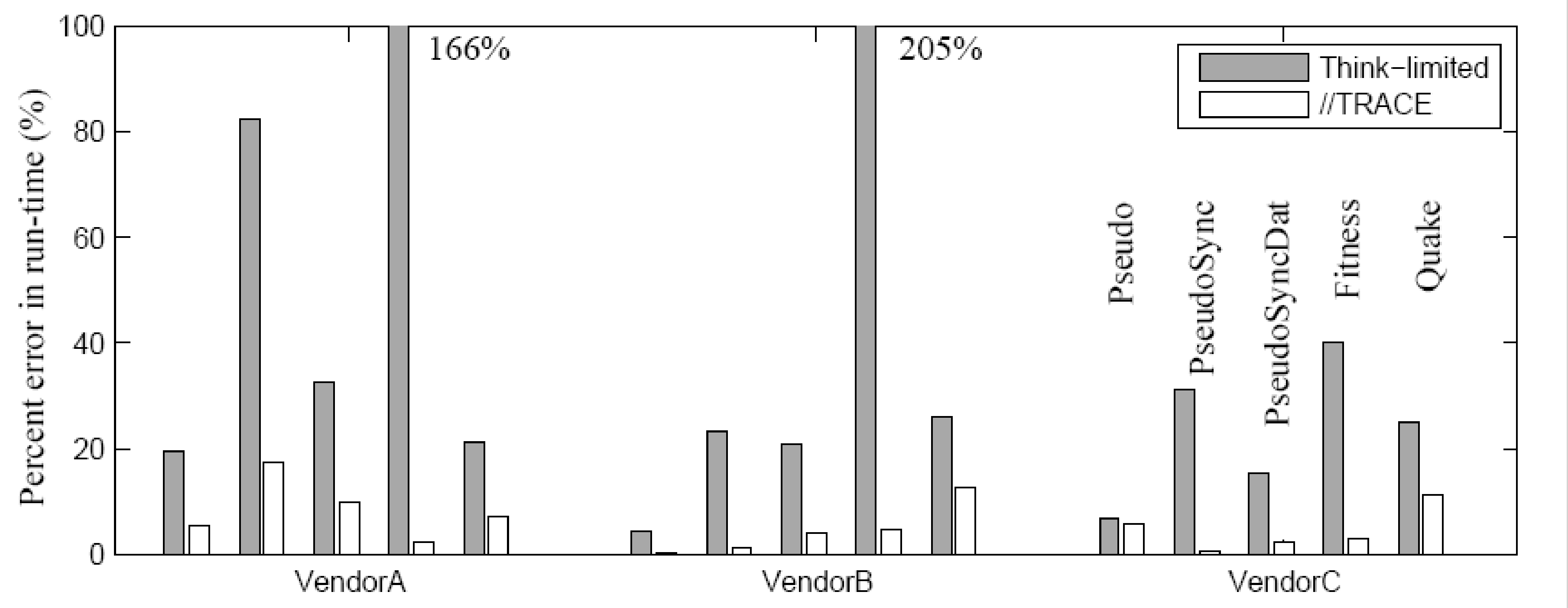
Pseudo has little synchronization, best result

PseudoSync PseudoSyncDat affected due to synchronization. Computation affects result

Fitness – worst affected if synchronization ignored

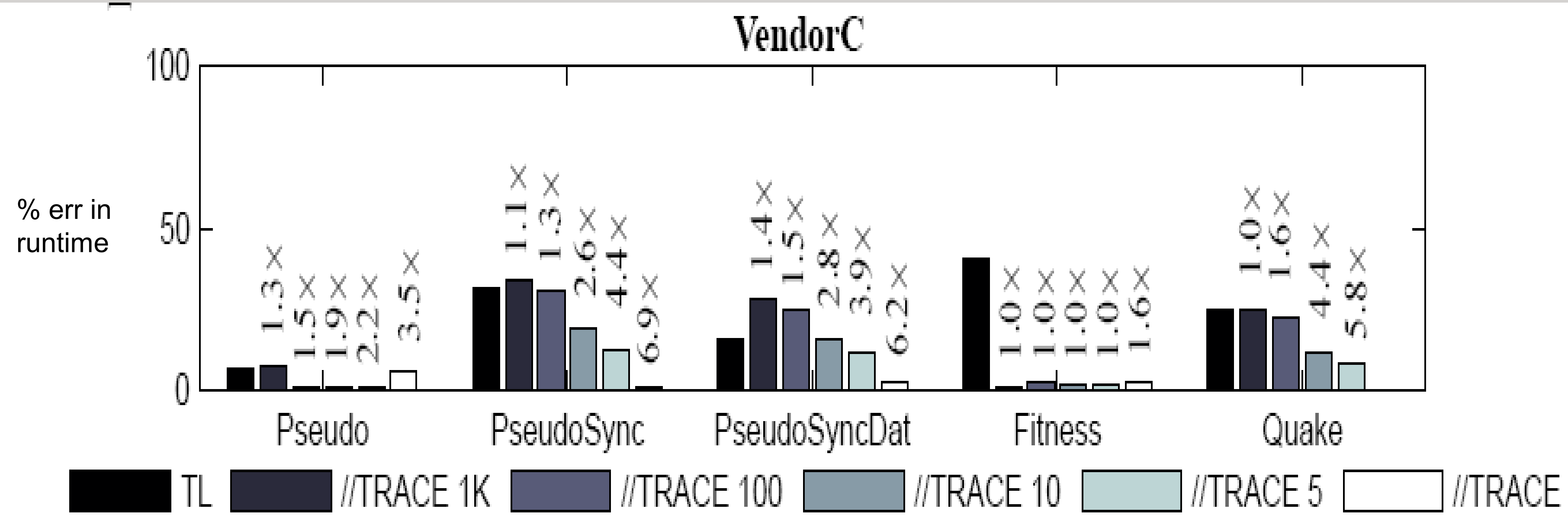


I/O throttling(Experiment 2)



- substantial gain for PseudoSync/Dat , Fitness, Quake(due to replayed synchronization)
- Pseudo not affected as much due to lack of data dependencies

I/O sampling (Experiment 3)



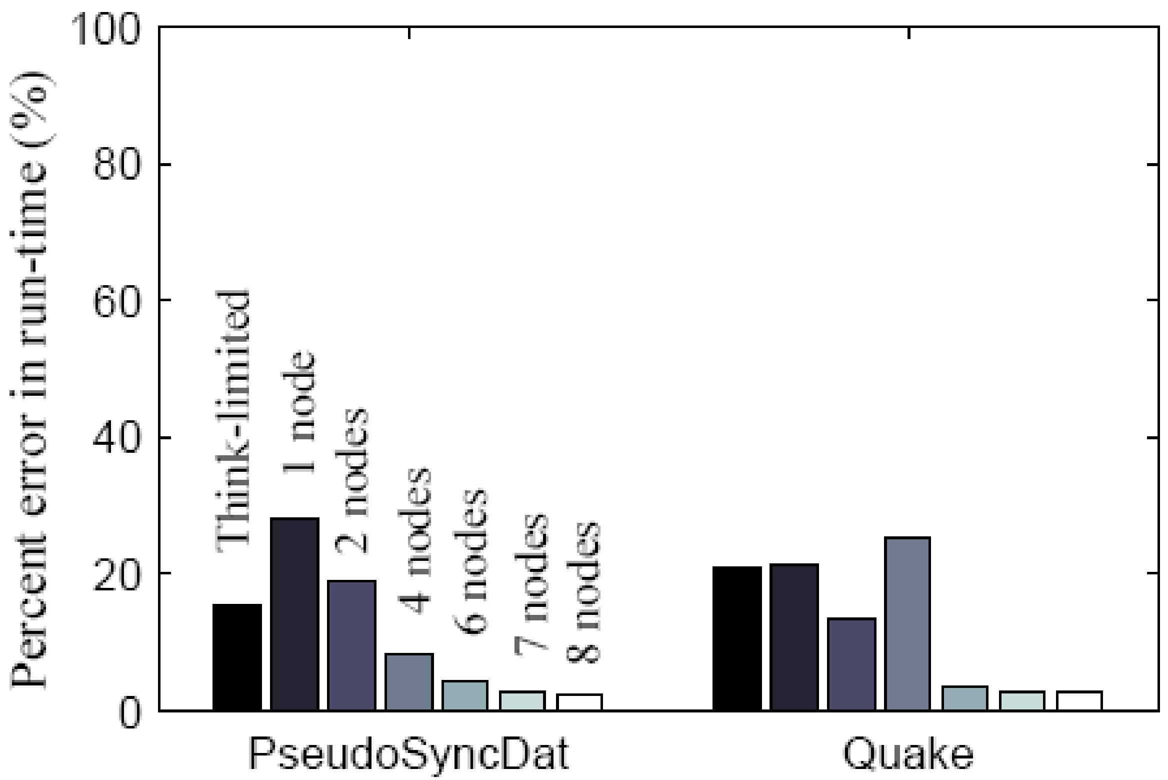
- Pseudo, Fitness – few data dependencies
- higher sampling rate discovers more data dependencies

Experiment 4 (Node sampling)

low replay error can be achieved without having to throttle every node

As with I/O sampling, one can sample nodes iteratively until a desired accuracy is achieved

heuristics for intelligent node sampling are required to more effectively guide the trace collection process and further reduce tracing time



Conclusion

- presents a technique for accurately extracting and replaying I/O traces from parallel applications
- By selectively delaying I/O while tracing an application, computation time and inter-node dependencies can be discovered and approximated in trace annotations
- average replay error is below 6%.