Scalable Collection of Large MPI Traces on Red Storm

Cray User Group (CUG) Meeting
Seattle, Washington, USA

Rolf Riesen
Sandia National Laboratories
rolf@sandia.gov

May 9, 2006

Overview

Introduction

Seshat

Experiments

Experiments II

Experiments III

Related Work

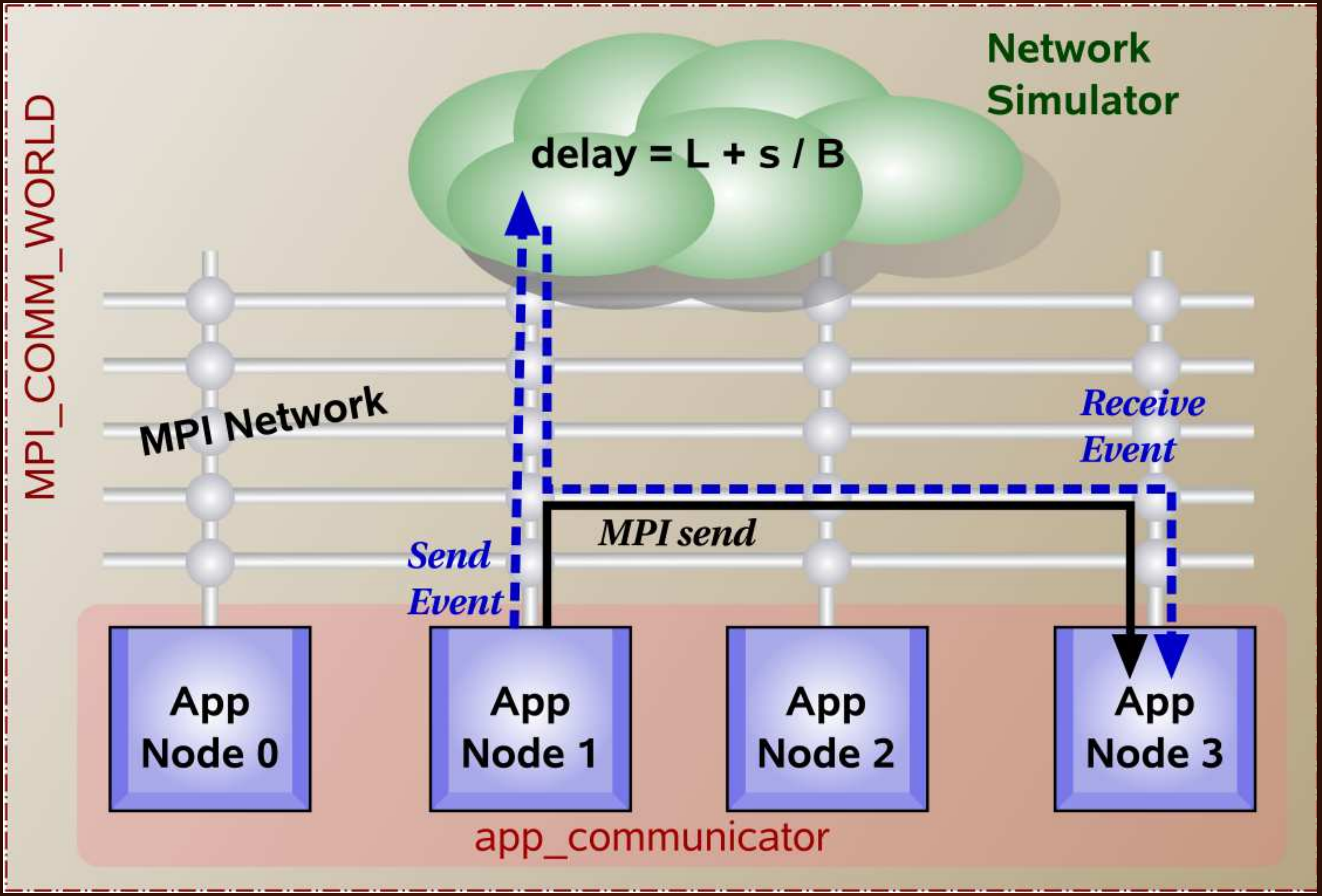
Summary and Future Work

Introduction

Introduction

- Many applications today are so complex (and dynamic) that it is very difficult to predict message passing patterns and behavior
- MPI traces can help analyze applications
- Traces can also be used to feed simulators for next-generation systems
- Problem: Extracting traces changes application behavior
- This talk presents preliminary results for an intrusion free MPI trace collector

Seshat

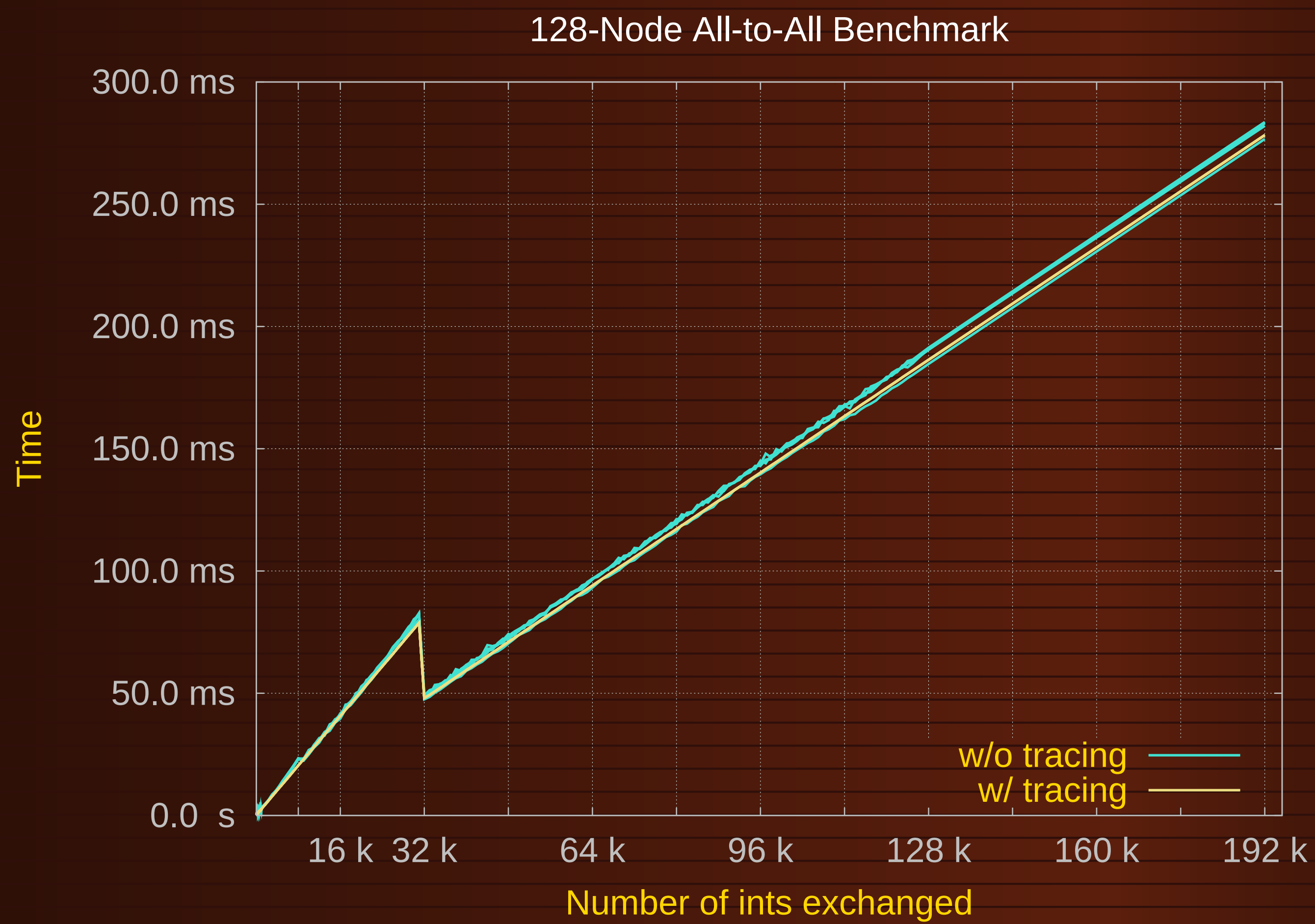

Network Simulator
delay = L + s / B
MPI_COMM_WORLD
MPI Network
Receive Event
Send Event
MPI send
App Node 0
App Node 1
App Node 2
App Node 3
app_communicator

Design

- Execution driven network simulator
  - Current sim is simple; uses Red Storm parameters
  - Plans to make it parallel and topology aware
- Use MPI profiling interface to hook into existing applications
  - No code instrumentation; only re-link needed
- Run each node in virtual time, set by simulator
  - `MPI_Wtime()` returns virtual time
- Network sim collects statistics about ever message in app
- Can write info to a trace file without disturbing virtual time

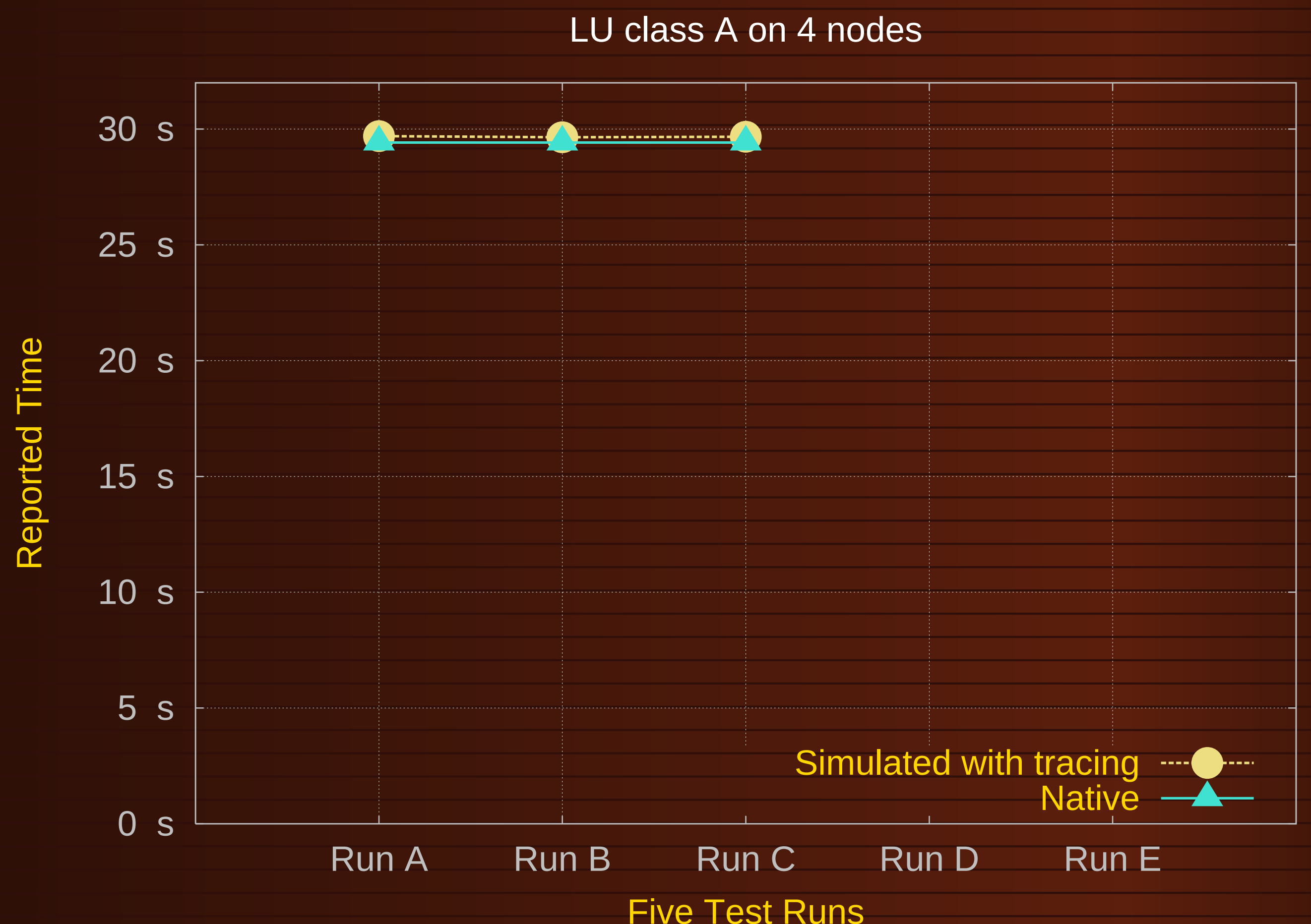
Experiments

All-to-All benchmark on 128 Nodes

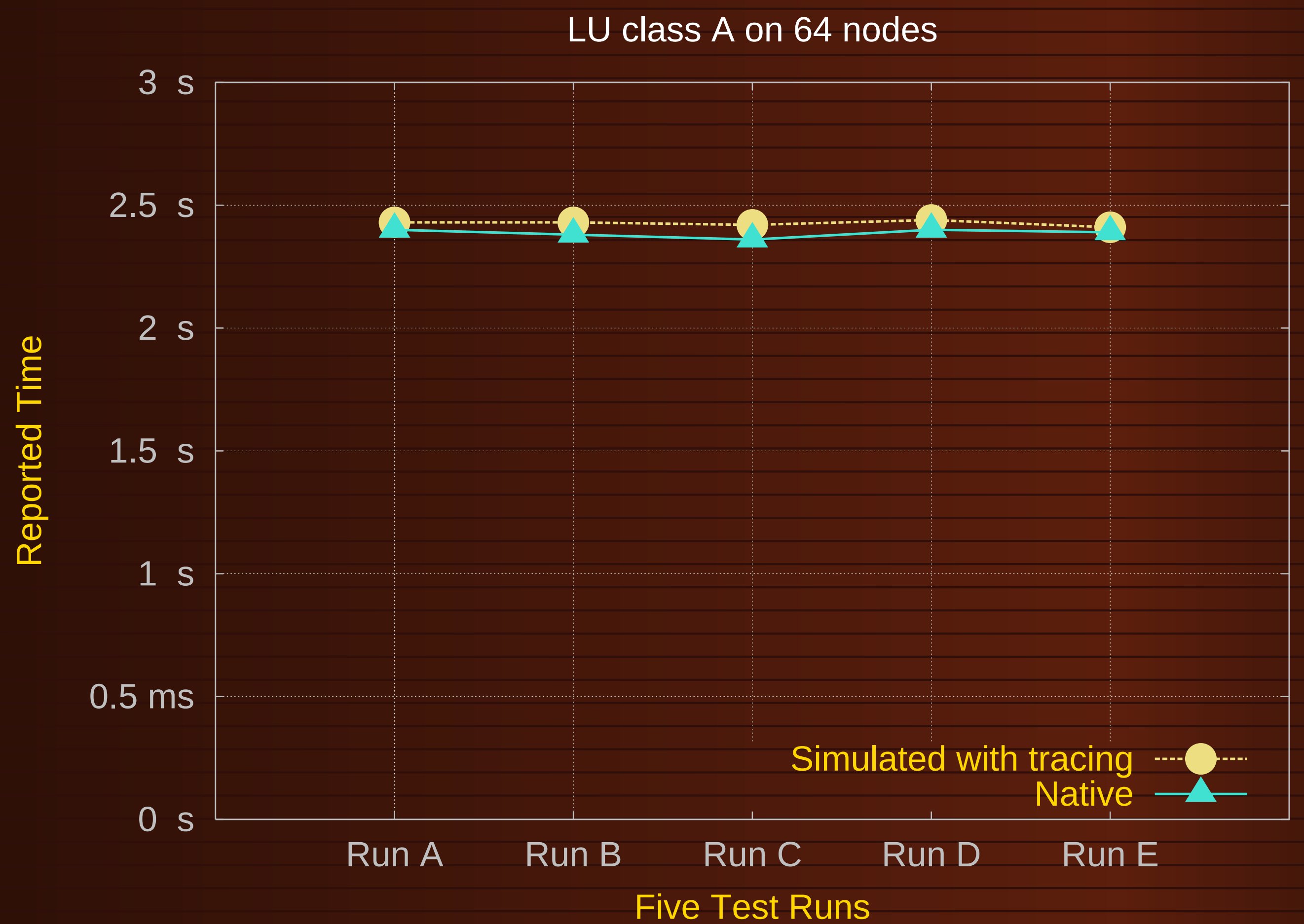


- 10 runs, same nodes, alternate trace on/off

NAS LU Class A on 4 Nodes



NAS LU Class A on 64 Nodes



Trace Format

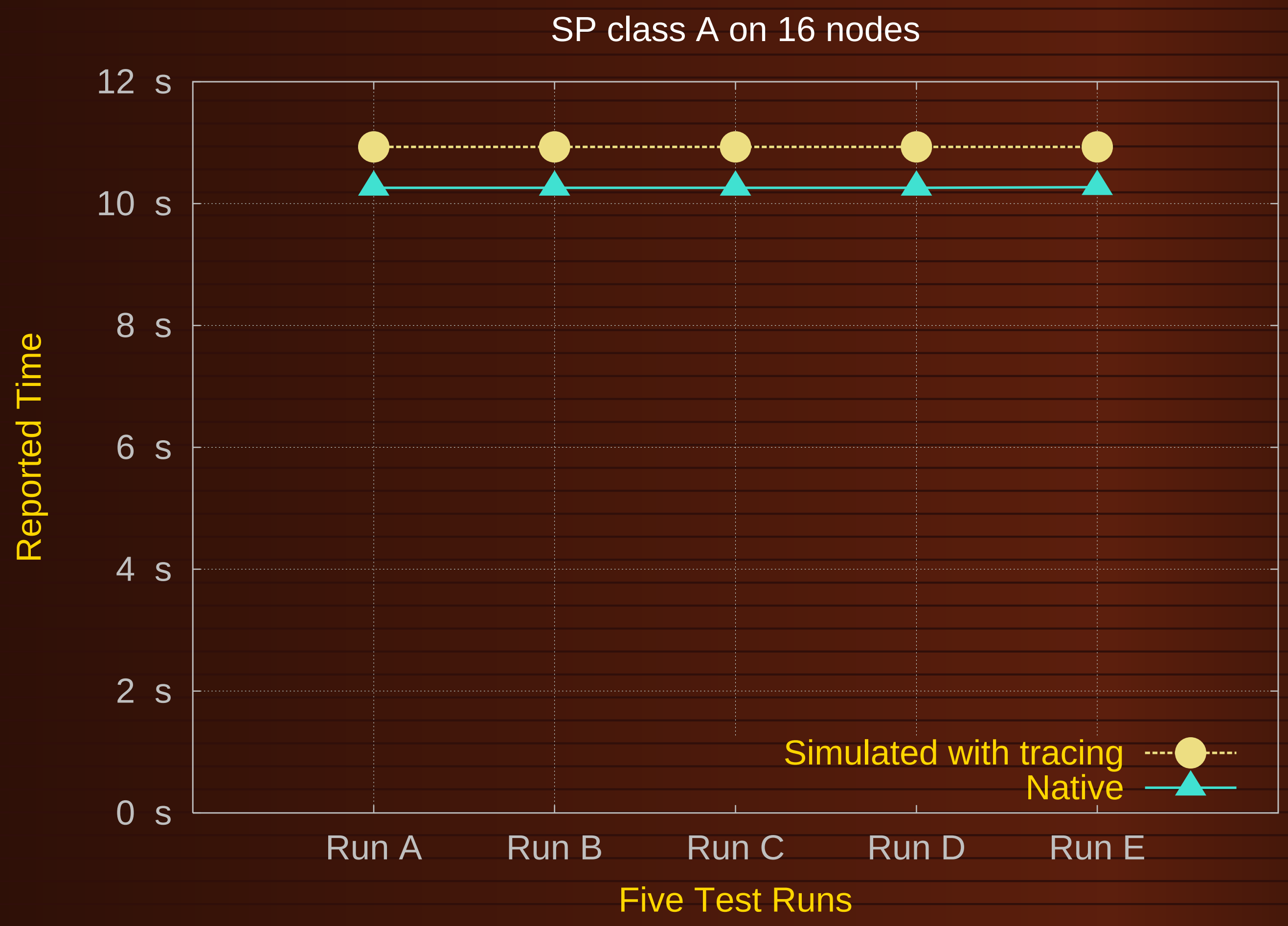
- Time of event at network simulator
- Source (or root) of message (collective)
- Destination
- Virtual send time
- Simulated time in network
- MPI tag
- Type of collective
- Length of message in bytes
- ASCII format, $\approx$ 90 bytes per event

Trace Statistics

Code	Nodes	Events	Wall Clock Time		Trace Size
			w/o	w/ trace	
All-to-all	128	4,826,000	1,300s	15,671s	397 MB
LU, A	4	126,635	30s	391s	11 MB
LU, A	16	759,699	10s	2,288s	63 MB
LU, A	64	3,545,003	4s	10,581s	285 MB
LU, A	256	$>$ 7,172,517	3s	$>$ 21,557s	$>$ 589 MB

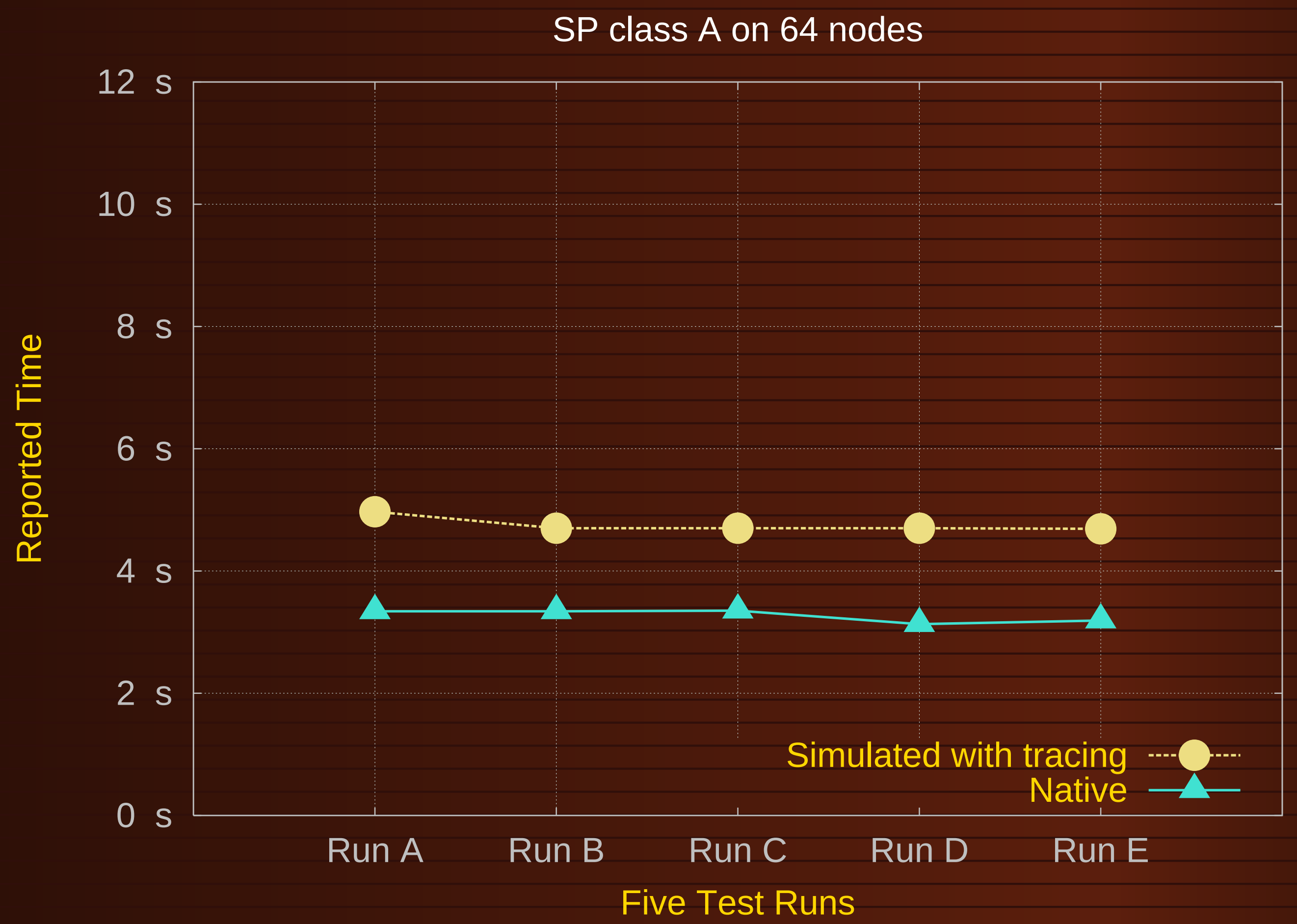
- 256-node LU job killed after 6 hours
- Trace file written to home directory (NFS, not parallel file system)

NAS SP Class A on 16 Nodes



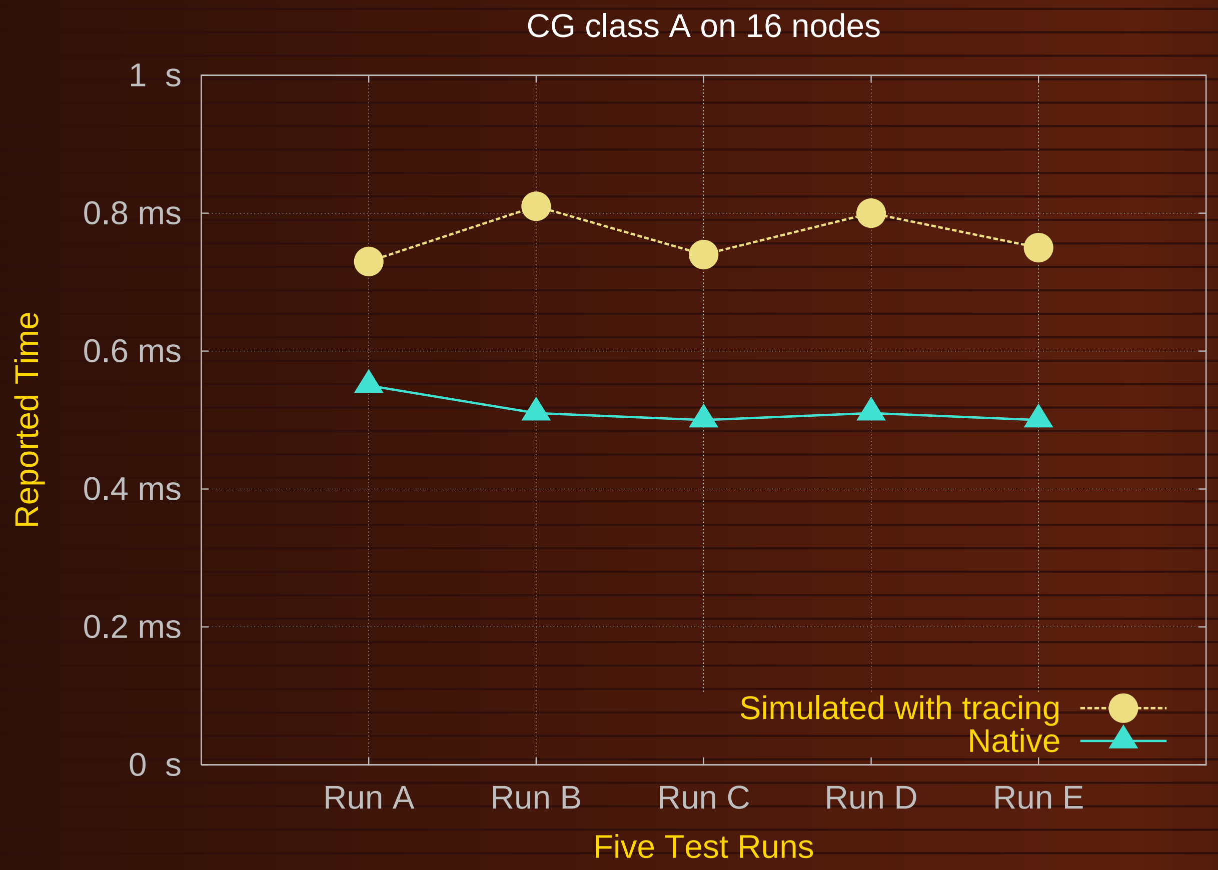
- Reported time w/ trace is 6.5% higher

NAS SP Class A on 64 Nodes



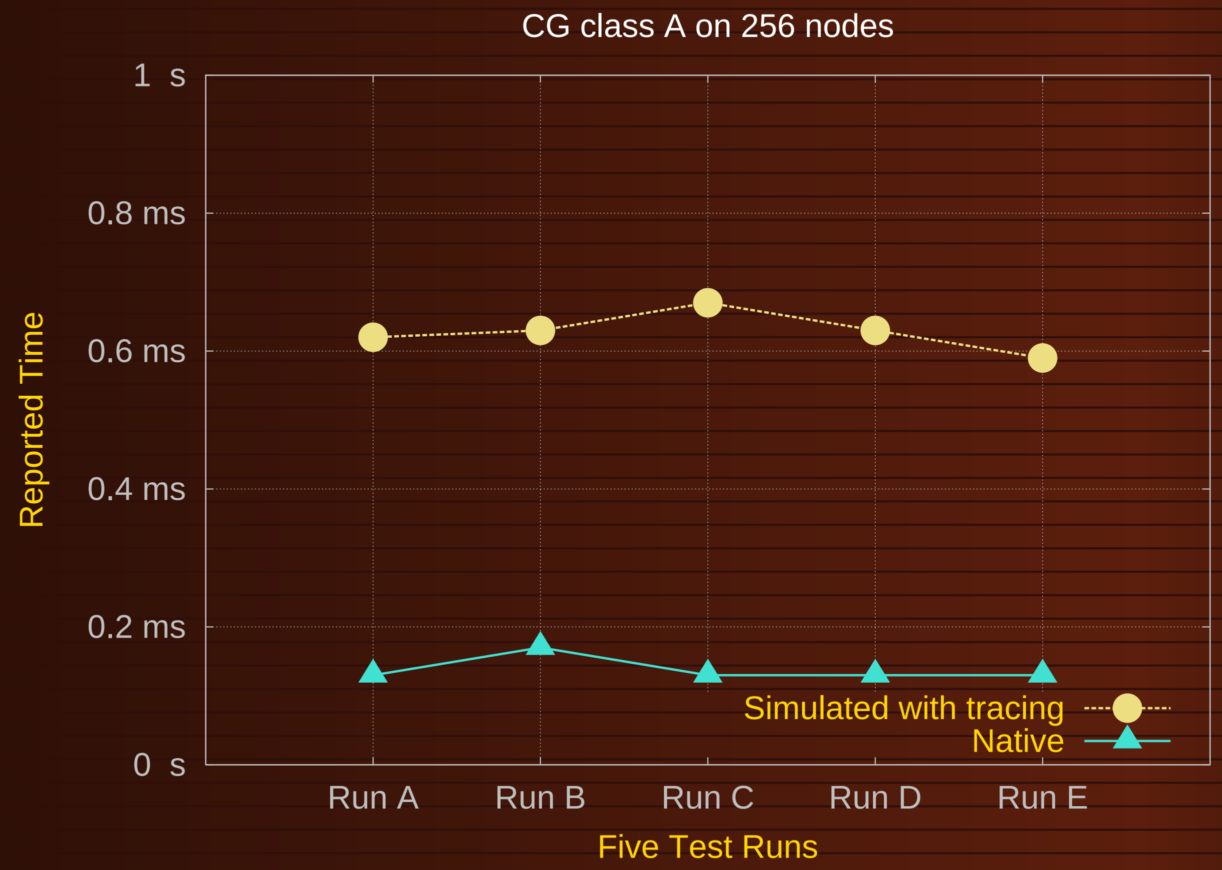
- Reported time w/ trace is 40% higher!

Experiments II



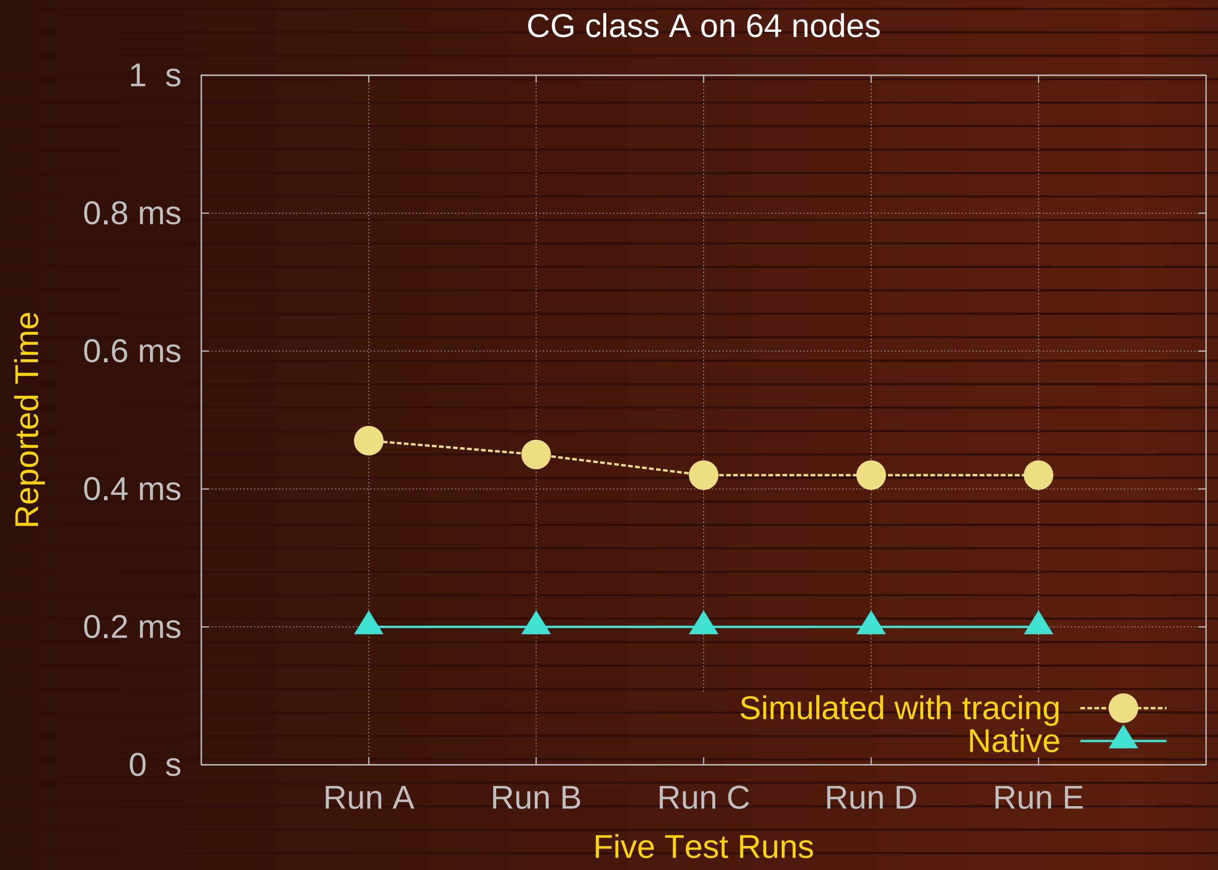
- Reported time w/ trace is 48% higher!

NAS CG Class A on 256 Nodes



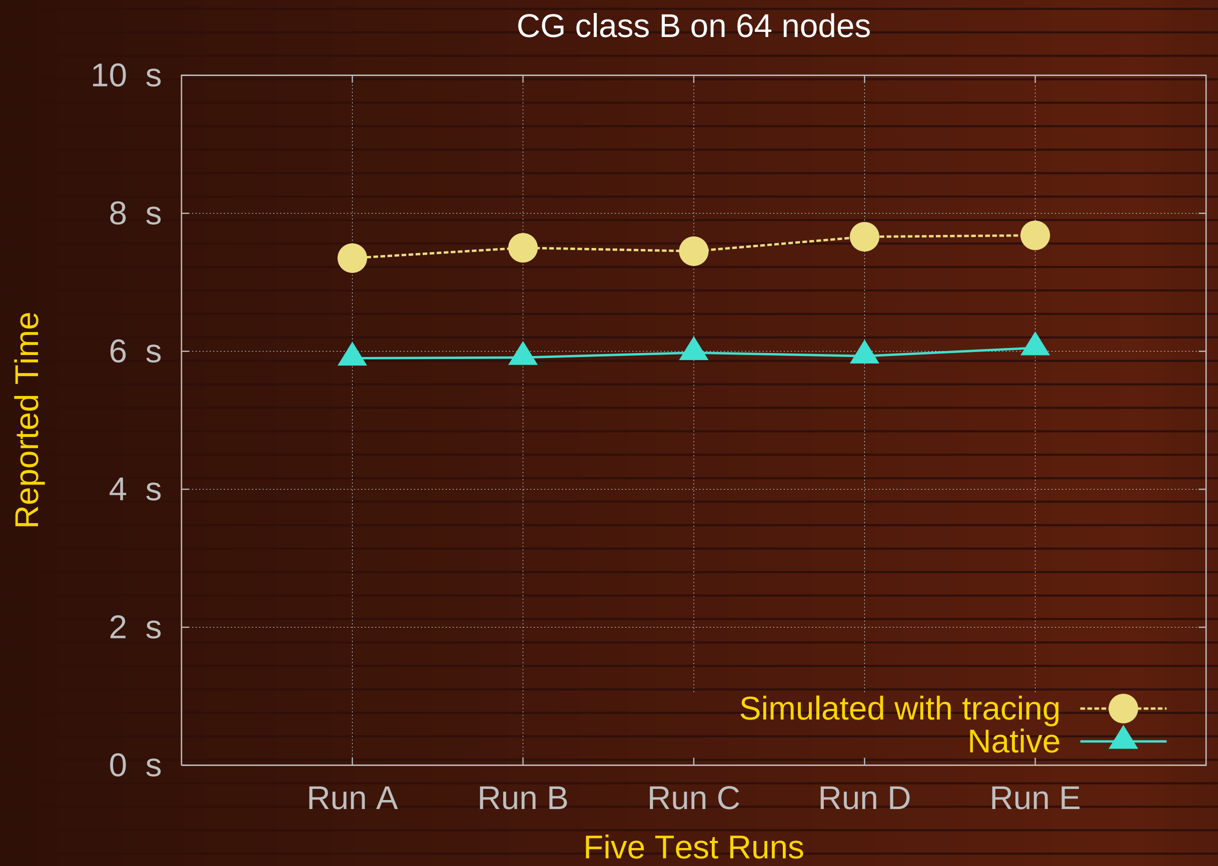
- Reported time w/ trace is 385% higher!
- Does benchmark class or trace size matter?

NAS CG Class A on 64 Nodes



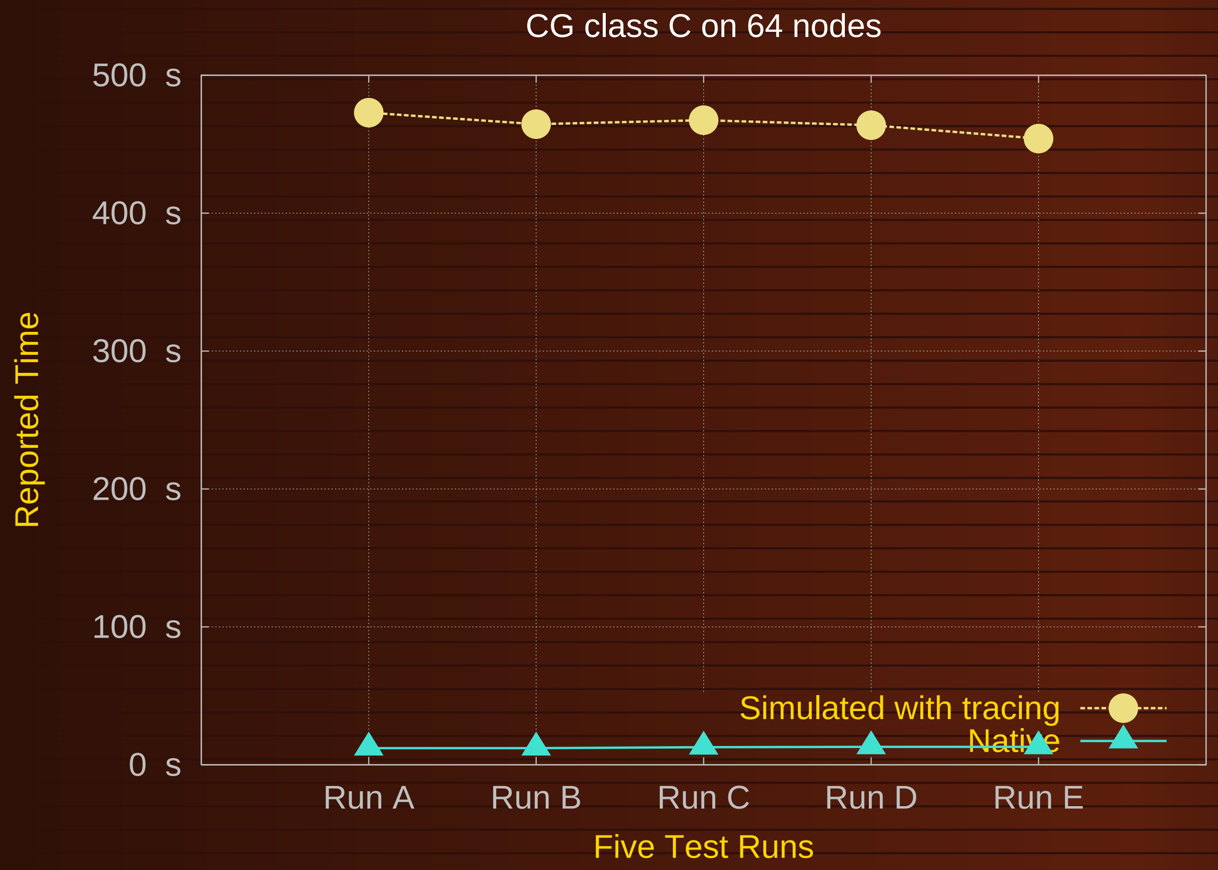
- Reported time w/ trace is 110% higher!
- Number of events: 269,501

NAS CG Class B on 64 Nodes



- Reported time w/ trace is 25% higher!
- # of events: 1,279,421 (5 times more than class A)

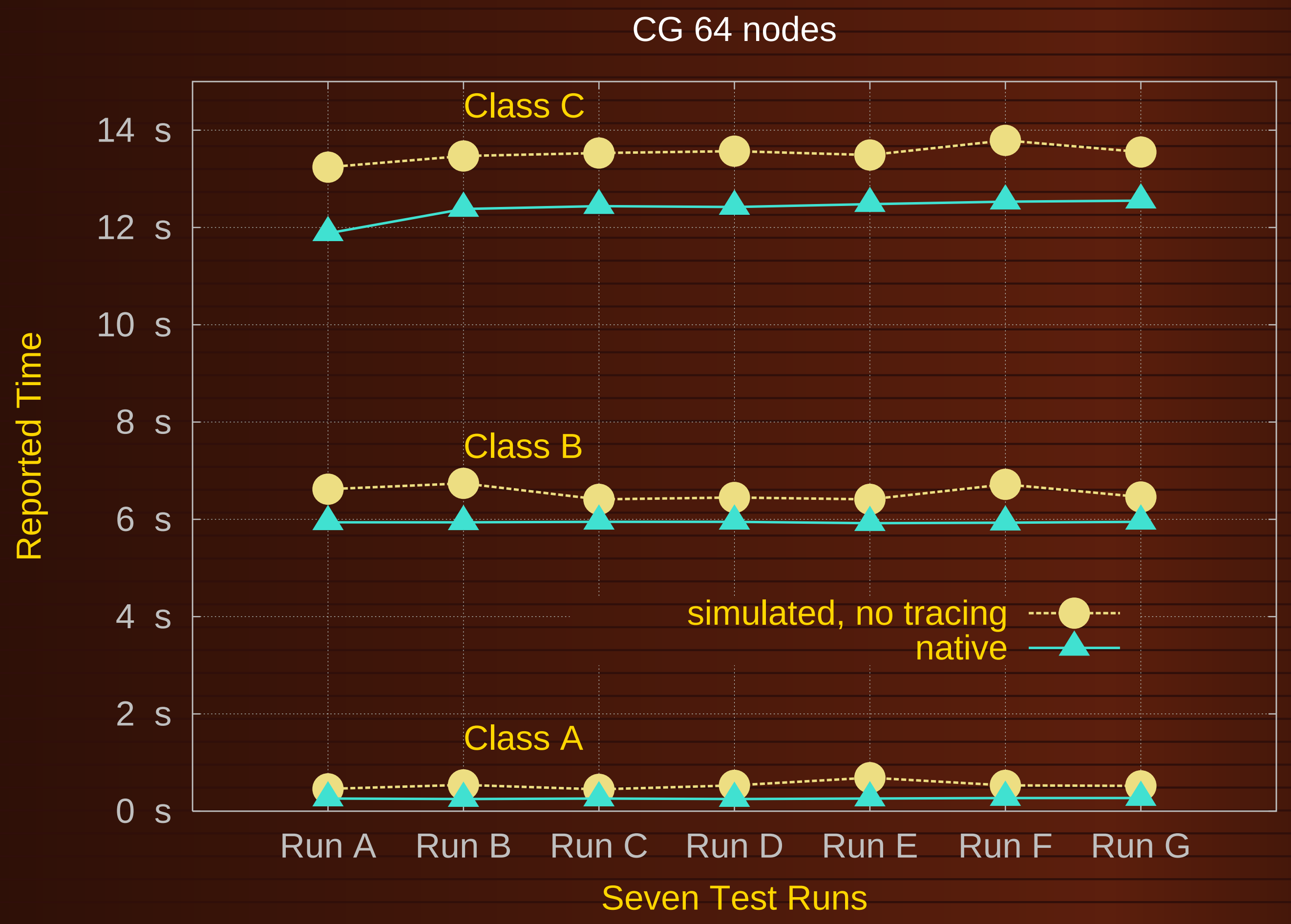
NAS CG Class C on 64 Nodes



- Reported time w/ trace is 3,557% higher!
- # of events: 1,279,421 (same as class B)
- Problem is not class or event size!

Experiments III

NAS CG on 64 Nodes



- Bug seems to be in virtual time adjustment
- Delay due to tracing exacerbates problem

Related Work

Related Work

- Two ways to assess message passing behavior:
  - Collect complete trace data, but alter application behavior
  - Collect only statistics
- Need to reduce size of trace and computation time
- E.g., IPDPS'07 paper (Michael Noeth et. al) compresses traces, but leaves timing information out

Summary and Future Work

Summary and Future Work

- Fix timing bug
- Proof of concept
- Clearly need to compress data
  - Buffer traces in sim node or on buffer-node to reduce wall-clock time.
- Customizable trace format and filter

Questions?