Debugging at Extreme Scale using proc++ and TBON-FS

Michael Brim
Paradyn Project

Paradyn / Dyninst Week
College Park, Maryland
March 26-28, 2012

What is Extreme Scale?

100,000+ hosts

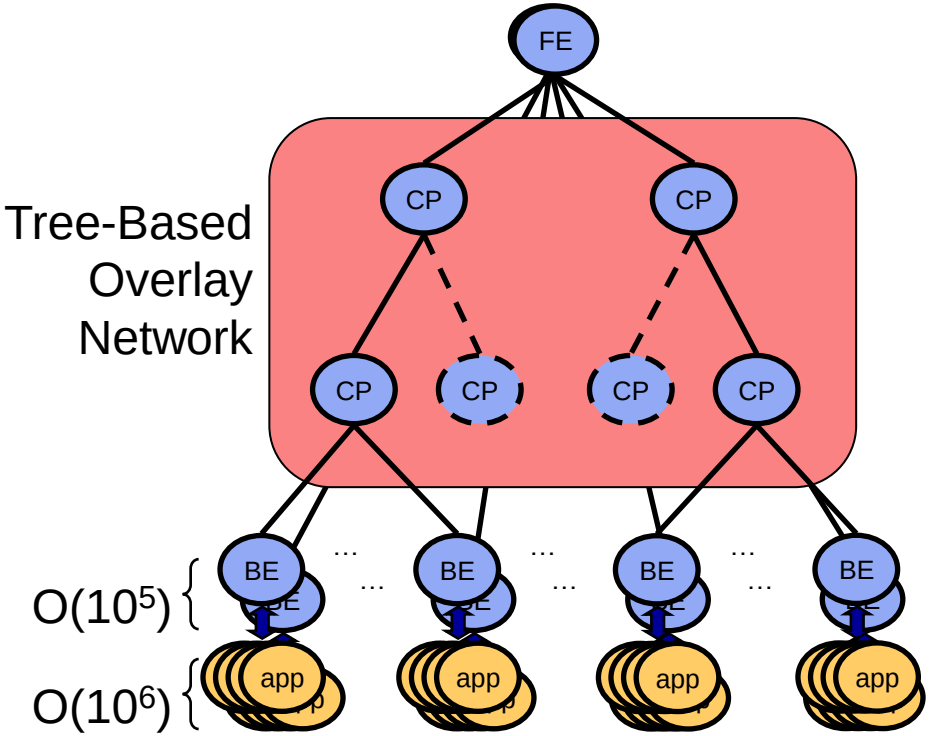
1,000,000+ processes and threads

- Deployed Systems
  - K Computer: ~88,000 8-core hosts
  - Tianhe-1A: ~7,000 12-core hosts + ~7000 accelerators
- Imminent Systems
  - Sequoia: ~98,000 16-core hosts
  - Titan: ~18,000 16-core hosts + ~1000 accelerators

Problems with Debugging at Scale

Control: needs to be fast and efficient

Inspection: how to deal with data overload?



Problems with Existing Parallel Debuggers

Designed for interactive use

- GUI
- fine-grained user control
- not easy to use in batch environments

Heavyweight

- client or servers
- kitchen sink phenomenon

No customization of aggregate views

Extreme Scale Debugger Use Cases

Uses that actually make sense at scale

- My app fails due to an error in some processes
- My algorithm is producing weird/bad results
- My app hangs

Better suited to other tools or analysis

- Finding bad MPI communication patterns
- Detecting data races, memory corruption

Debugging Primitives to Support Uses

Where are my processes/threads in their execution?

- right now
- when some process fails

What is the value of a variable/parameter at specific points in the program?

- equality, range validity, distributions
- evaluate simple expressions on many variables

Debugging Primitives: Execution

Launch, Attach

- identify processes (a potential bottleneck)
- bring processes under control
- classify processes (e.g., by exe, libs)
- identify threads

Where are my processes/threads?

- program counter equivalence
- stack trace equivalence (i.e., STAT)
  - optional: function parameters

Debugging Primitives: Data Analysis

Program variable inspection

- equivalence, binning, simple aggregates
- user-defined aggregations
- multi-variable expressions (e.g., (X + Y) ÷ 2)

Common breakpoints and watchpoints

Analysis points

- combines a breakpoint, variable inspection, and continue after data access

Scalable Debugger Building Blocks

Group file operations

- idiom that avoids iteration during group operations on distributed files

TBON-FS

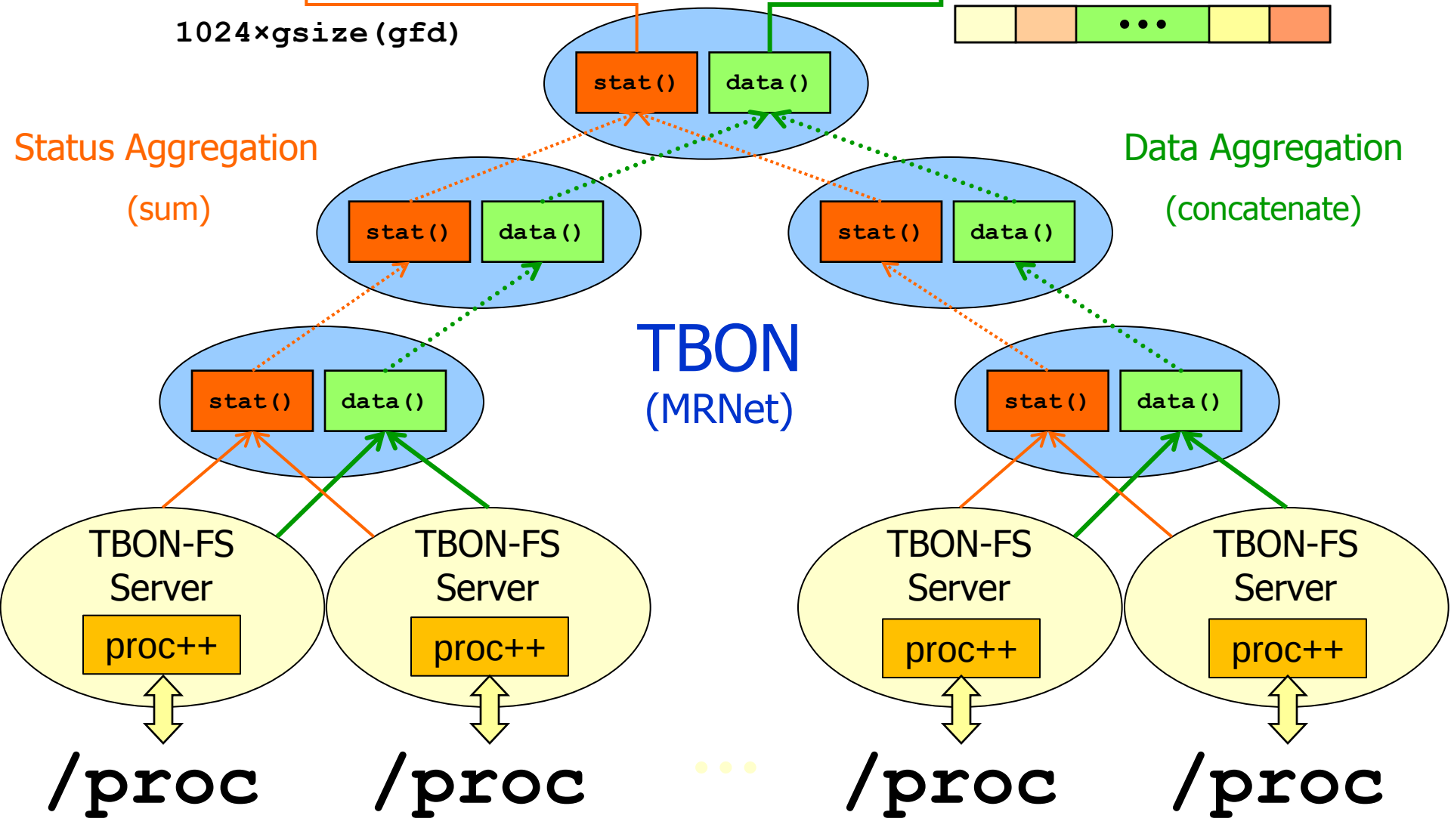
- implementation of scalable operations on distributed file groups using MRNet

proc++

- synthetic file service providing control and inspection of process and thread groups
- built on ProcControlAPI

TBON-FS: Scalable Group File Operations

```
int rc = read(gfd, databuf, 1024)
```



tbon-dbg: A lightweight parallel debugger

Initially, a test harness for TBON-FS & proc++

Group-centric

- default groups: all processes, all threads
- user-defined custom groups

Focus group operations

- **Control**: stop/continue/step, write memory/registers, insert/remove breakpoint
- **Inspection**: read memory/registers, gather events, gather stack traces

tbon-dbg: Interactive & Batch Use

CLI similar to a Unix shell

- navigate TBON-FS global name space using `cd`, `ls`, `pwd`
- scriptable for batch environments

```
# create a session, make it focus
session create 1
session 1
# attach to all processes running myexe
attachcmd myexe
# where are all the threads
group allthreads
group readreg pcequiv
# set a breakpoint in all processes, run to it
group allprocs
group break myexe code-offset
group ctl cont
group eventsequiv wait
# dump some interesting data
group readimg myexe variable-offset > /home/mjbrim/datafile
# all done, exit will automatically detach all session procs
exit
```

tbon-dbg Performance Evaluation

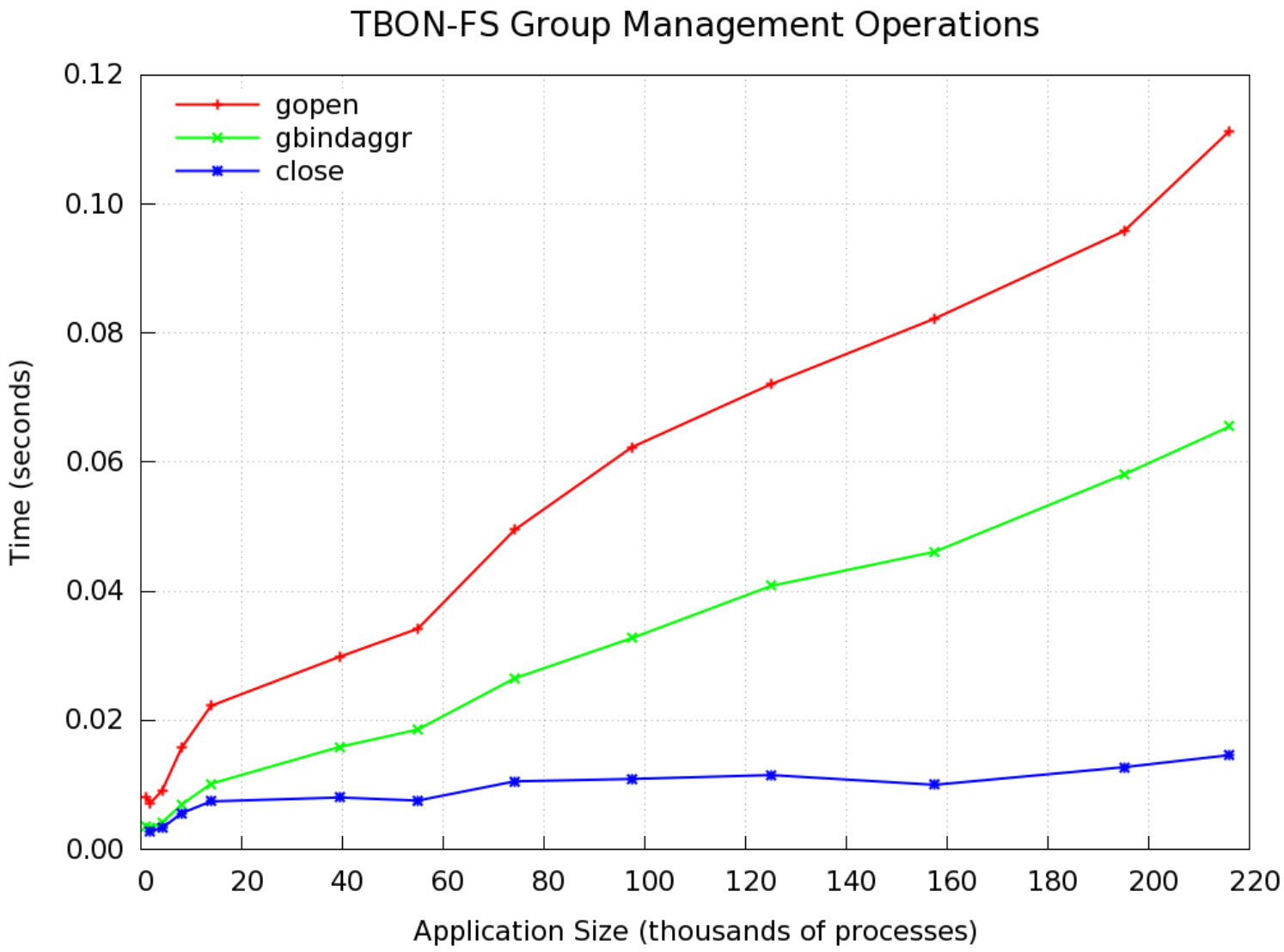
Experiments on ORNL JaguarPF (Cray XT5)

- Application: Sequoia IRS Benchmark
- Up to 216,000 processes

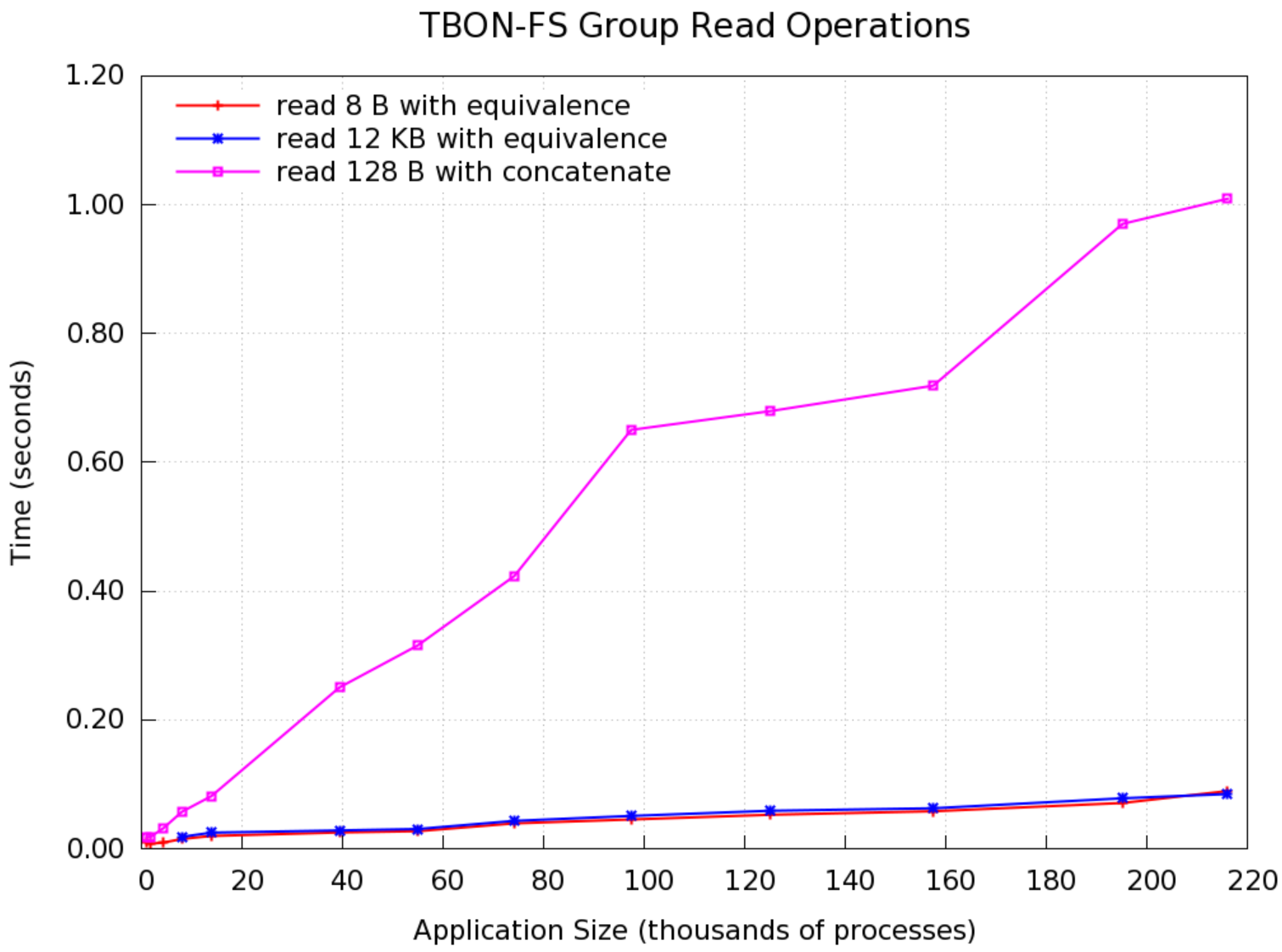
Two Experiments

1. Raw TBON-FS and proc++ performance
2. tbon-dbg group debugger operations

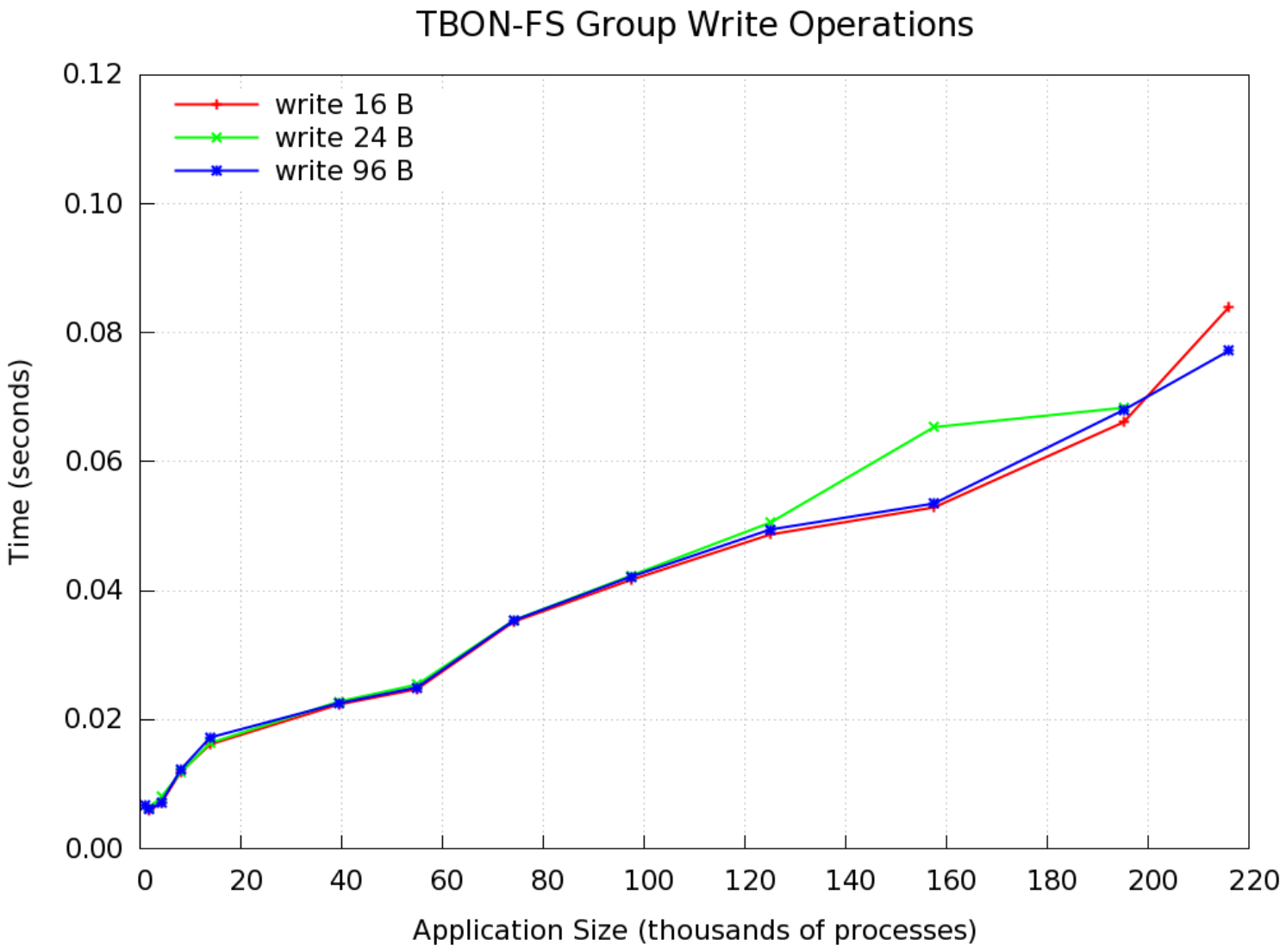
TBON-FS & proc++ Evaluation



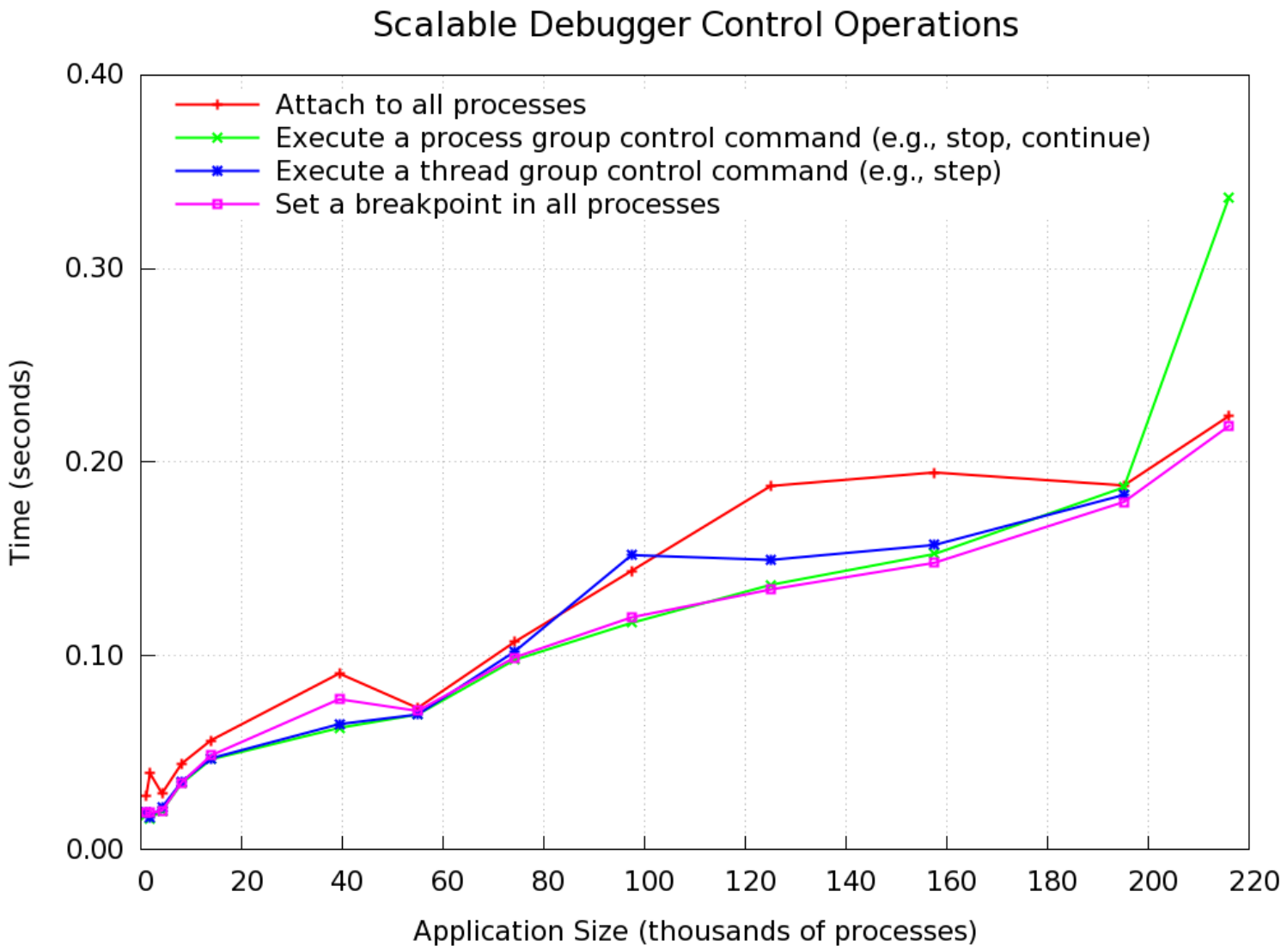
TBON-FS & proc++ Evaluation



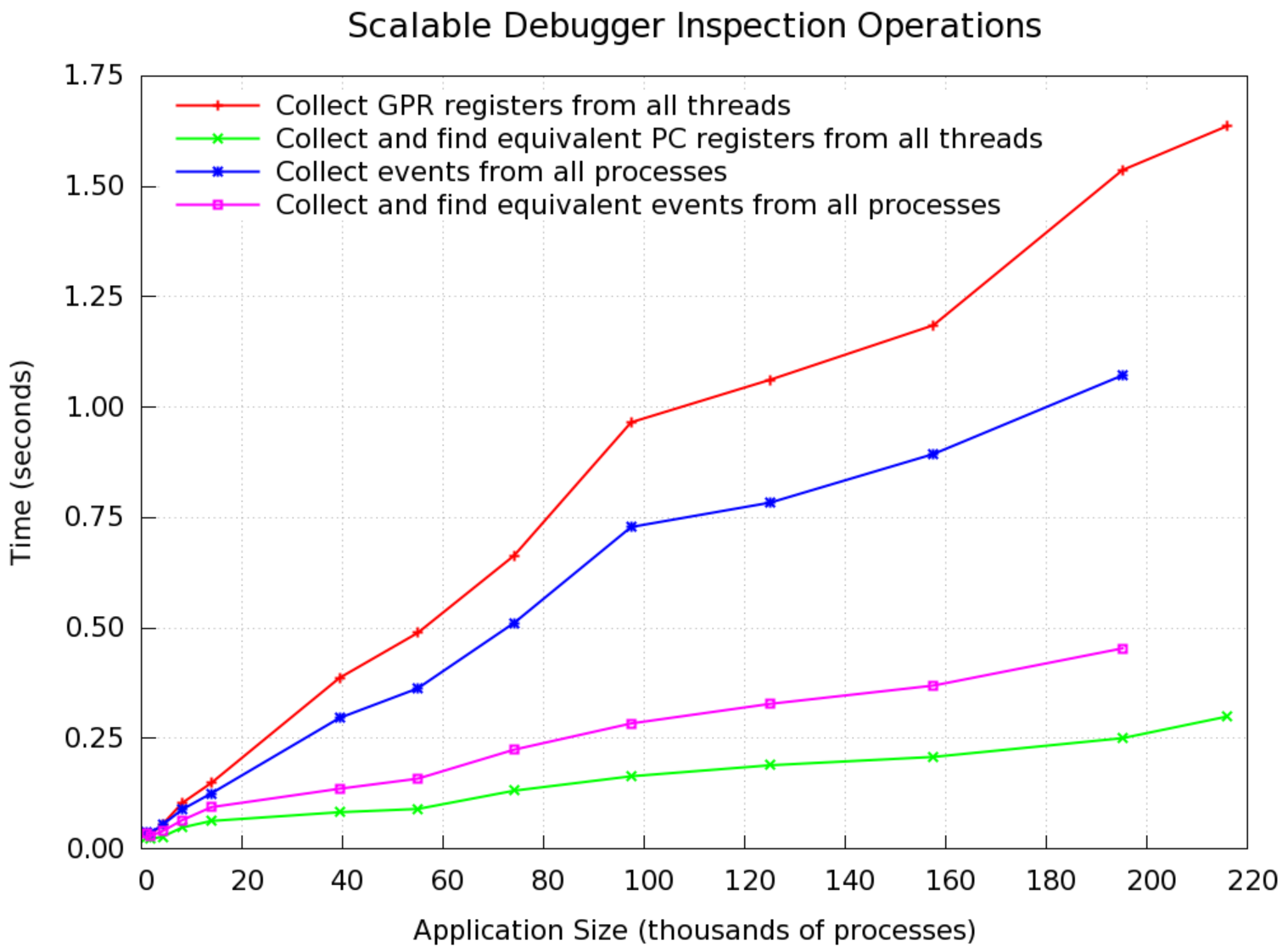
TBON-FS & proc++ Evaluation



tbon-dbg Evaluation – Control



tbon-dbg Evaluation – Inspection



tbon-dbg Status

Supports use case defined by LLNL for batch debugging of IRS

- milestone for TotalView scalability project
- TBON-FS & proc++ also integrated within TotalView, enabled runs up to ~150,000 procs

All components are free and open-source

tbon-dbg Limitations – Future Work

Not yet user friendly

- No knowledge of program symbols
  - manual code/data offset calculations
  - need to integrate SymtabAPI
- No custom data aggregations

Stack traces not yet supported

- need to integrate StackwalkerAPI
- implement default STAT-like aggregation

No watchpoints or analysis points