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?

Tree-Based Overlay Network

$O(10^5)$

$O(10^6)$
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 at Extreme Scale using proc++ and TBON-FS
Debugging Primitives: Data Analysis

Program variable inspection

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

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

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

1024\times\text{gsize}(gfd)

Status Aggregation (sum)

Data Aggregation (concatenate)

TBON (MRNet)

TBON-FS Server

proc++

/proc

TBON-FS Server

proc++

/proc

TBON-FS Server

proc++

/proc

TBON-FS Server

proc++

/proc

...
tbon-db: 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-db: Interactive & Batch Use

CLI similar to a Unix shell

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

```bash
# 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

# set a breakpoint in all processes, run to it
group allprocs

# dump some interesting data
group readimg myexe variable-offset > /home/mjbrim/datafile

# all done, exit will automatically detach all session procs
exit
```
tbon-dbpg 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-dbpg group debugger operations
TBON-FS & proc++ Evaluation

TBON-FS Group Management Operations

- gopen
- gbindaggr
- close

Time (seconds) vs. Application Size (thousands of processes)

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

TBON-FS Group Read Operations

- read 8 B with equivalence
- read 12 KB with equivalence
- read 128 B with concatenate

Application Size (thousands of processes) vs. Time (seconds)
TBON-FS & proc++ Evaluation

TBON-FS Group Write Operations

- write 16 B
- write 24 B
- write 96 B

Diagram showing the time (seconds) vs. application size (thousands of processes) for different write operations.
tbon-dbq Evaluation – Control

Scalable Debugger Control Operations

- Attach to all processes
- Execute a process group control command (e.g., stop, continue)
- Execute a thread group control command (e.g., step)
- Set a breakpoint in all processes

Time (seconds)

Application Size (thousands of processes)
tbon-dbq Evaluation – Inspection

Scalable Debugger Inspection Operations

- Collect GPR registers from all threads
- Collect and find equivalent PC registers from all threads
- Collect events from all processes
- Collect and find equivalent events from all processes

Application Size (thousands of processes)

Time (seconds)
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-dbgs 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