DySectAPI: Scalable Prescriptive Debugging

Jensen, Nicklas Bo; Karlsson, Sven ; Quarfot Nielsen, Niklas ; Lee, Gregory L.; Ahn, Dong H.; Legendre, Matthew ; Schulz, Martin

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

DySectAPI: Scalable Prescriptive Debugging

Enabling users to construct probe trees for automatic, event-driven debugging at scale

Nicklas Bo Jensen, Sven Karlsson @ DTU Compute, Technical University of Denmark Niklas Quarfot Nielsen @ Mesosphere Inc
Gregory L. Lee, Dong H. Ahn, Matthew Legendre, Martin Schulz @ Computation, Lawrence Livermore National Laboratory

Motivation

- Current parallel debugging models are largely inadequate for extreme-scale systems
  - Traditional model quickly overwhelms developers even at moderate scale
  - Lightweight model, where information is sacrificed for scalability, scales well but can significantly limit debug information
- We need a new debugging model that can scale while still providing affordable information levels

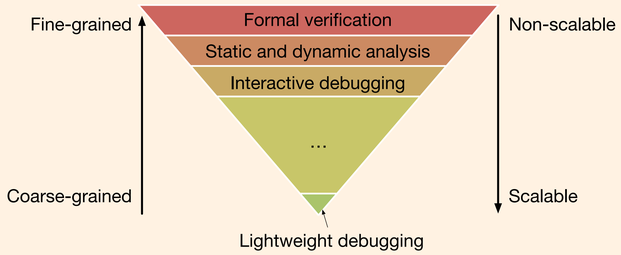


Figure: Debugging scalability based on strategies and amounts of details.

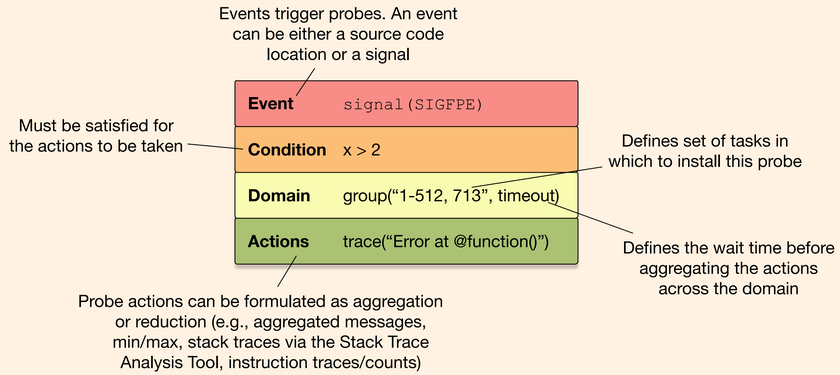
Contributions

- A novel debugging model that can strike a balance between scalability and capability
  - A prescriptive model
- Allows programmers to codify their debug intuition
- DySectAPI, an implementation of this model
- Performance and scalability evaluation
- Case study to demonstrate DySectAPI's effectiveness

Source Code

Open source code available at:
https://github.com/scalability-llnl/DysectAPI

Anatomy of a Debugging Probe



Probe Tree for a Real Use Case

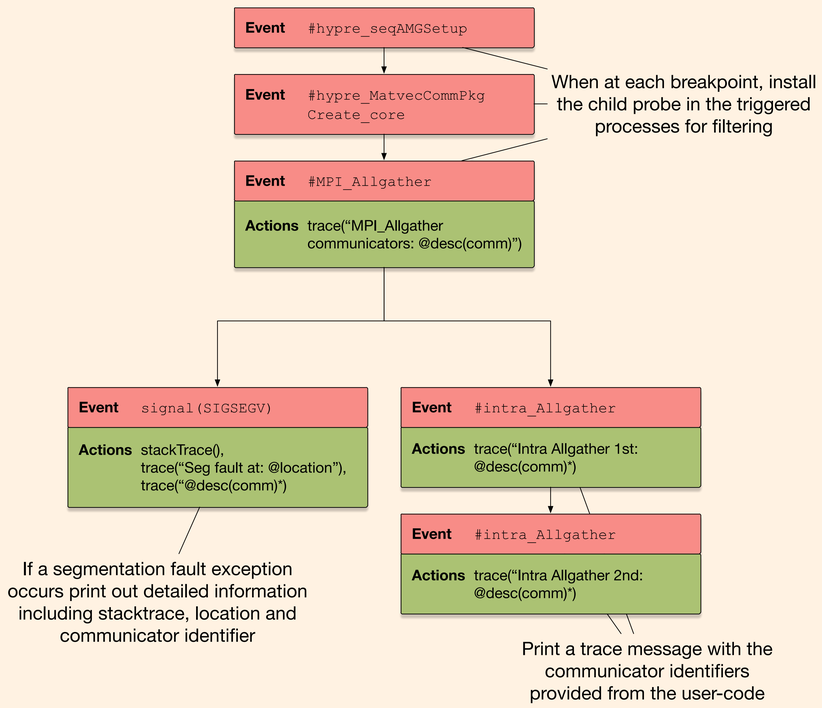


Figure: Probe-tree debugging session for an MPI bug only manifesting itself at 3,456 processes.

Demonstrating Scalability

- Goal: show the scalability of our prescriptive debugging model
- Uses the STAT infrastructure, including MRNet, for scalable tool communication and data processing
- Use an analytic runtime-overhead model to predict scalability at large scale
- Logarithmic scaling demonstrated experimentally and modeled
- Our overhead model predicts a four probe chain-shaped tree would incur 118 ms in debugging the entire Cab system (20,736 cores)
- Pruning debug information via a probe tree significantly reduces information presented to programmers
  - Four chained probes and pruning of 50% of the processes in each probe leave just 12.5% of the original processes

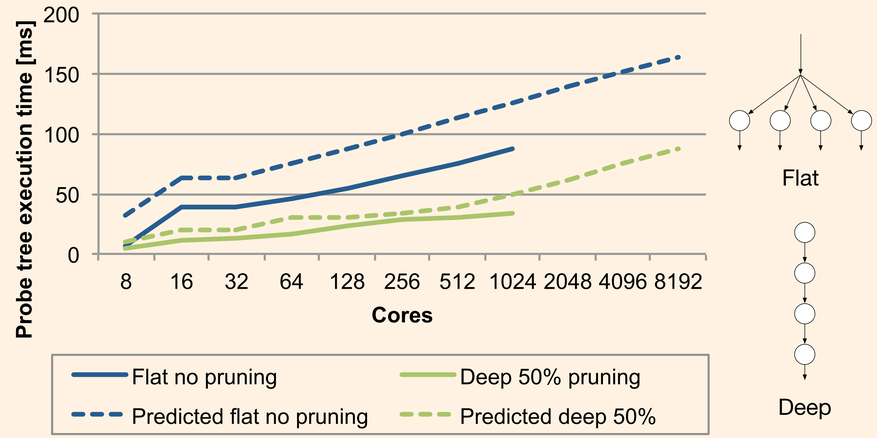


Figure: Actual and modeled execution time on Cab with 16 cores per node for a flat and deep probe tree.

Impacts

- A novel debugging model that can scale without sacrificing key debug information presented to programmers
- Already proven effective in isolating an MPI bug that manifested itself only at or above 3,456 processes
- Debugging of common debugging scenarios using probe-tree templates
- Future work includes advanced probes using dynamic binary instrumentation

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

LLNL-POST-779317