

# STAT: the Stack Trace Analysis Tool Quick Start Guide



Gregory L. Lee

Dorian C. Arnold

Dong H. Ahn

Bronis R. de Supinski

Barton P. Miller

Martin Schulz

**STAT: the Stack Trace Analysis Tool Quick Start Guide**

by Gregory L. Lee

by Dorian C. Arnold

by Dong H. Ahn

by Bronis R. de Supinski

by Barton P. Miller

by Martin Schulz

## Table of Contents

|                             |   |
|-----------------------------|---|
| Disclaimer .....            | v |
| Auspice .....               | v |
| License .....               | v |
| 1. When To Use .....        | 1 |
| 2. Using the STAT GUI ..... | 3 |



# Disclaimer

## Auspice

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

## License

Copyright (c) 2007-2008, Lawrence Livermore National Security, LLC.

Produced at the Lawrence Livermore National Laboratory

Written by Gregory Lee [lee218@llnl.gov], Dorian Arnold, Dong Ahn, Bronis de Supinski, Barton Miller, and Martin Schulz.

LLNL-CODE-400455.

All rights reserved.

This file is part of STAT. For details, see <http://www.paradyn.org/STAT>. Please also read STAT/LICENSE.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer (as noted below) in the documentation and/or other materials provided with the distribution.

Neither the name of the LLNS/LLNL nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL LAWRENCE LIVERMORE NATIONAL SECURITY, LLC, THE U.S. DEPARTMENT OF ENERGY OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### Additional BSD Notice

1. This notice is required to be provided under our contract with the U.S. Department of Energy (DOE). This work was produced at Lawrence Livermore National Laboratory under Contract No. DE-AC52-07NA27344 with the DOE.
2. Neither the United States Government nor Lawrence Livermore National Security, LLC nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights.
3. Also, reference herein to any specific commercial products, process, or services by trade name, trademark, manufacturer or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those

*Disclaimer*

of the United States Government or Lawrence Livermore National Security, LLC,  
and shall not be used for advertising or product endorsement purposes.

## Chapter 1. When To Use

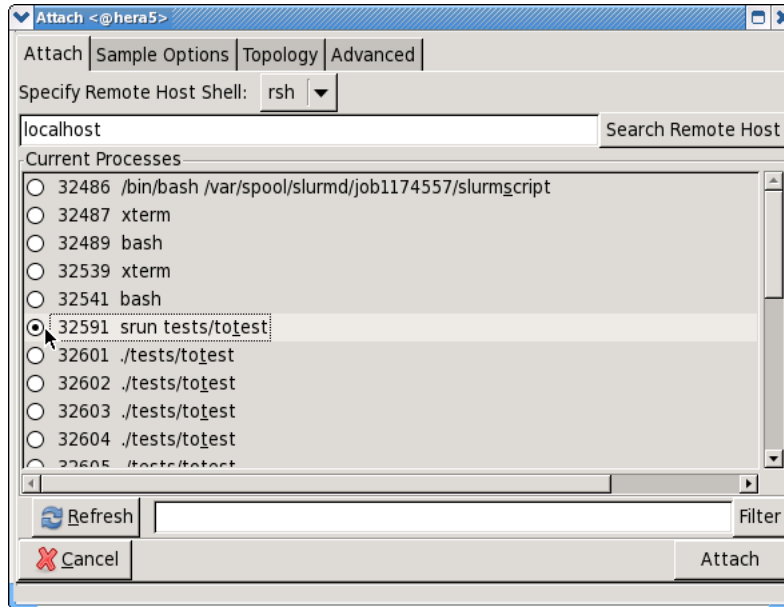
STAT is most effective for diagnosing parallel applications that are hung (i.e., deadlock or livelock), however, its analysis can also be useful for debugging other errors. The outputted, merged stack traces indicate where in the code all of the processes are at a given point in time, giving the user insight into where the bug may be. The merged traces also form process equivalence classes, which can be used to select a subset of tasks to feed into a full-featured parallel debugger such as DDT or TotalView for root cause analysis.





## Chapter 2. Using the STAT GUI

After launching the GUI via the **STATGUI** command you will first need to attach to the application via the **Attach** button. This will bring up the attach dialog (Figure 2-1). You will need to select the job launcher (i.e., **mpirun**, **srun**, or equivalent process). STAT will list processes owned by you on the localhost. If your job launcher process is on a remote host, you will need to enter that hostname in the remote host text entry box. Contact your local system administrator if you are not sure where to find your job launcher process. Once the appropriate process has been selected, click on the **Attach** in the lower right hand corner. STAT will then launch its daemons and gather an initial stack trace.



**Figure 2-1.** Use the attach dialog to select the job launcher process to attach to.

Once STAT has displayed the initial merged stack trace, in the form of a prefix tree, first you may want to look for common buggy patterns. This can be accomplished by using the analysis buttons in the toolbar across the top of the window. This includes operations to look for outliers such as processes with the shortest or longest stack trace or for the stack trace that was exhibited by the least or most processes. Note that these buttons are "traversal" buttons and they all initially operate on the full prefix tree. For example, the first click of the **[Shortest] Path** button will display the shortest path and subsequent clicks will display the next shortest path. Oftentimes bugs in parallel applications are triggered by a single or small subset of outliers in which case the **[Least] Tasks** button can quickly identify the outliers. Another common behavior is for a small subset of processes to be hung and the rest of the processes to be blocked in an MPI barrier or collective. In this case, the hung subset of tasks may have a shorter call path than the tasks in blocking in MPI, since the MPI implementation will usually be several frames deep. In this case, the **[Shortest] Path** can be useful.

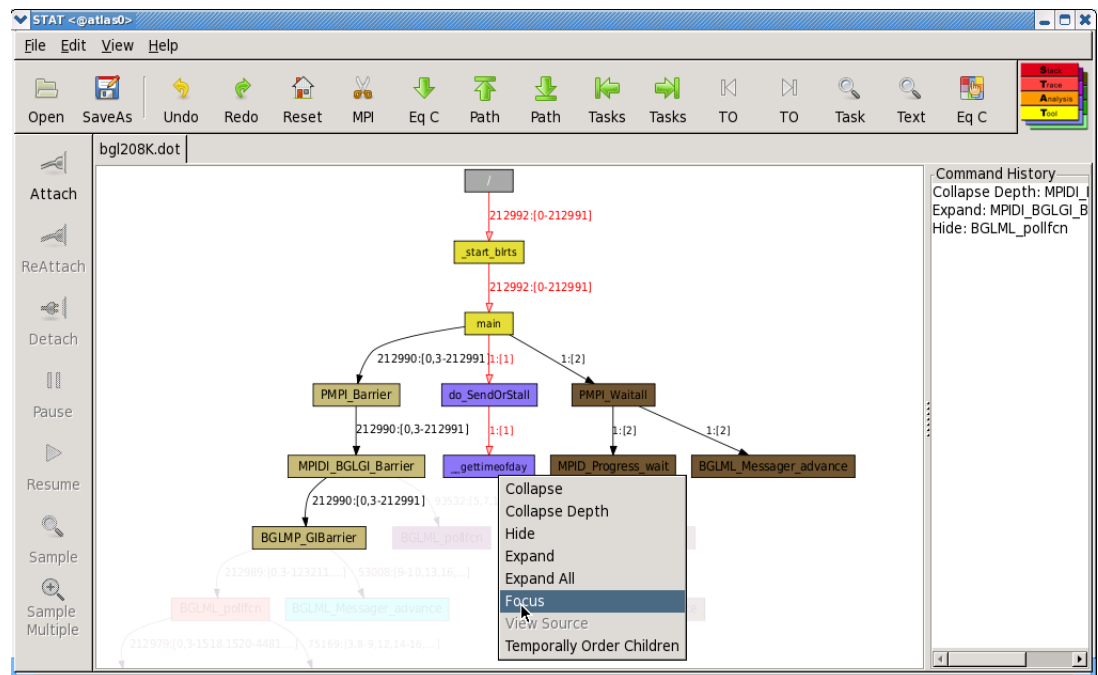


Figure 2-2. A screenshot of the STAT GUI.

Alternatively, you may wish to manually search through the stack traces. There are several buttons to aid in this process too. The **[Traverse] Eq C** will traverse the prefix tree, with each click traversing the down to the next point where there is a branch in equivalence classes. Another helpful button is the **[Cut] MPI** button, which will cut any frames below an MPI function call, thus allowing you to focus on application code as opposed to the MPI implementation stack frames. There are also buttons to search for specific MPI ranks or for stack frames with specified text. Finally, left or right clicking on a node in the prefix tree gives you the option to expand or collapse the prefix tree. Note also that you can zoom in and out of the prefix tree using the options in the **View** menu or by using the scroll button on your mouse. You can also hold the left mouse button to "grab" the whitespace in the displayed prefix tree and move the focus around.

By default, the initial sample will gather stack traces at the granularity of function names. You can gather an additional sample with more detail, by clicking on the **Sample** button and selecting the **function and line** radio button. Note this typically requires that the code be compiled with the `-g` flag to get the appropriate debug information. After clicking **OK** a new prefix tree will be generated. By gathering stack traces with line number information, you may now associate stack traces back to the source code using the **View Source** button after clicking on a node in the prefix tree (Figure 2-3).

```

035|
036|     do_Receive(prev, tag, &buf[0], &reqs[0]);
==>037|     do_SendOrStall(next, tag, rank, &buf[1], &reqs[1], numtasks);
038|     MPI_Waitall(2, reqs, stats);
039|
040|     MPI_Barrier(MPI_COMM_WORLD);
041|     MPI_Finalize();
042|     return 0;
043| }
044|
045| void do_SendOrStall(int to, int tag, int rank, int* buf, MPI_Request* req, int n)
046| {
047|     if (rank == 1)
048|     {
049|         fprintf(stderr, "MPI task 1 of %d stalling\n", n);
==>050|         while(1) ;
051|     }
052|
053|     MPI_Isend(buf, 1, MPI_INT, to, tag, MPI_COMM_WORLD, req);
054| }

```

Figure 2-3. A screenshot of the source view window.

STAT was not intended to be a full-featured debugger, so you may ultimately need to employ another debugger such as DDT or TotalView for root cause analysis. STAT includes an interface to launch either of these debuggers (where available) on a subset of the MPI tasks based on the equivalence classes that STAT identifies. This interface can be accessed through the **[Identify] Eq C** button in the upper right hand corner of the window. In order to allow the other debuggers to attach, STAT will first detach itself from the application. Pinpointing a bug may require several iterations of running STAT on the entire application and running a full-featured debugger on a subset. After detaching a full-featured debugger, you can quickly attach to your application again with the **ReAttach** button.

