Chaining Multiple Tools and Libraries Using Gotcha

Yiheng Xu[†], Kathryn Mohror[*], Hariharan Devarajan[*], Cameron Stanavige[*], Abhinav Bhatele[†]

[†]Department of Computer Science, University of Maryland, College Park, MD, USA
[*]Lawrence Livermore National Laboratory, Livermore, CA, USA

***Abstract*—In HPC, it is common to use tools to understand the performance of applications. Users often want to run applications linked with multiple performance tools to complement each other, as tools may record different information. Many tools operate by intercepting application function calls with "wrapper functions" that perform performance measurement tasks. However, this approach only works for using one tool at a time. Using multiple tools that intercept a common set of functions in the same run can lead to erroneous results.**

In this work, we enable multiple tools to measure a single application execution together utilizing a library called Gotcha. With our approach, we can use multiple tools in the same run to reduce overheads of running the application repeatedly and to produce more temporally aligned profiles from each of the tools. We present our work in making two I/O performance tools, Recorder and Darshan, Gotcha enabled.[1]

I. Motivation

Users often want to run applications linked with multiple performance tools to complement each other, as tools may record different information. For example, Figure 1 shows two performance tools (Recorder [1] and Darshan [2]) that record I/O data at different granularities. Most performance tools define wrappers that override functions. How applications find these wrappers is through the Global Offset Table, or GOT [3]. It is a section of a computer program's (executables and shared libraries) memory used to enable computer program code compiled as an ELF file to run correctly. Before the application runs, the runtime environment will go through all linked or loaded libraries and bind symbols with addresses. When a symbol is needed, the function pointer on the top will be used.

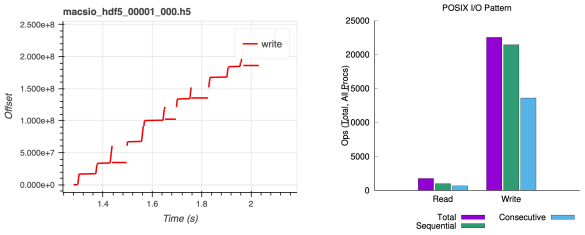

Fig. 1. Recorder and Darshan are two performance tools that record I/O data at different granularities.

However, this approach only works for using one tool at a time. Using multiple tools that intercept a common set of functions in the same run can lead to erroneous results. How wrappers are chained and the order of them in GOT is not deterministic and varies from system to system. For example, if we are to measure the performance of a user-level file system with a performance tool, we want the tool to first intercept I/O functions, then the user-level file system. But since their wrappers for a particular function look exactly the same, the runtime may put either one on top. Therefore, if the order is reversed, the work of the user-level file system won't be traced by the tool.

II. Methodology

To solve this problem, we decided to utilize Gotcha[1]. Instead of simply overriding the function and let the system deal with everything, Gotcha lets users explicitly define wrapper functions and binds them to functions to be intercepted. Gotcha provides data structures for users to create bindings, and APIs for users to wrap a function and get the real function that is wrapped. For a given tool, our work is to edit its source code to define Gotcha wrappers and wrappee handles for all functions it intercepts. We also set priorities for a tool so later we can control the order of tools.

However, using Gotcha introduces another concern. Most tools intercept `MPI_Init` and have their initialization in it. To use Gotcha, each tool has to initialize Gotcha as well. But we cannot put Gotcha initialization in `MPI_Init`, as it won't be wrapped by Gotcha. This brings us back to the original problem, that multiple tools are overriding the same function, which is `MPI_Init`.

Therefore, we decided to utilize constructors of shared libraries. The constructor runs when the shared library is loaded, typically during program startup. We defined a constructor for each tool and put Gotcha initialization in it. In this way, Gotcha is able to also wrap `MPI_Init`, and users don't need to care about initializing it in their code. Note that we leave the tool initialization in `MPI_Init`, so that the tool works as it originally was, i.e., it won't have extra tracings.

III. Results

We successfully made two I/O performance tools, Recorder and Darshan, Gotcha enabled. We run the same application with the original tools and the Gotcha version ones, and the results were the same. Users now can safely chain these tools together. They can also be used with multiple other Gotcha-enabled tools or a single regular tool. We present three benefits our work brings to HPC tools:

[1]This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-825920).

A. Ability to chain tools and libraries together.

We tested this with a software we have been developing called UnifyFS[4], which is a user-level file system. Since it is essentially a shared library working by interceptions, we cannot use other performance tools to measure its performance. But with our modifications, tools are now able to work with it.

Figure 2 shows the Recorder and Darshan profiles now contain UnifyFS calls. In the top figure, `/unifyfs` is a directory mounted by UnifyFS which doesn't exist and cannot be captured by Recorder originally. Moreover, the second figure is the writing to the Darshan logs in this run reported by Recorder. It shows the work done by Darshan is also captured by Recorder. It shows the time Darshan spent in generating its profiles. This means our work also makes it easy to measure the overheads of a tool: we don't need to modify the code of that tool. We simply need to use another tool to profile it.

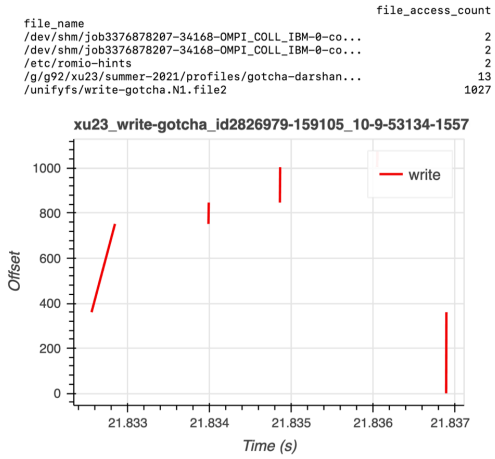

Fig. 2. Performance data captured by Recorder for UnifyFS[4] and Darshan when the tools are chained together.

B. Make profiles for a same run more comparable by eliminating system noises across runs.

In performance-related research, people usually use many different tools to profiles the same application run to get different levels of data. However, each run can only use one tool at one time, which means people have to run repetitively to use these tools separately. Due to the system noises, the profiles for the same experiment can be quite different. But now with our work, users can load all tools at the same time for an application run. We tested this with two I/O benchmarks Macsio and IOR running with 64 processes on Lassen. We first ran them with Recorder and Darshan separately. Then we did the same run with both Recorder and Darshan loaded at the same time. We repeated these two experiments several times. We checked the differences between some common metrics recorded by both tools. We can see the difference when running them together is less than running them separately.

The metric we pick for the plot on the left in Figure 3 is the max write time in rank 0 profiled by Recorder and Darshan. There is a big difference between running the application separately and together. Consider the overhead of running two together, this difference is more significant.

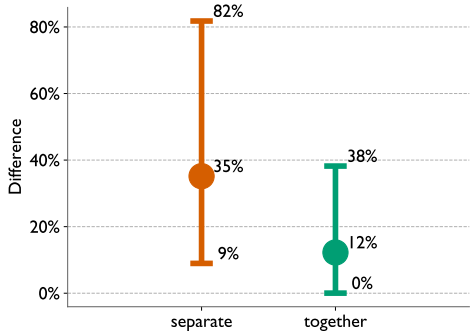

Fig. 3. Difference between common metrics ("time of longest write on rank 0") reported for the same application when the tools are used separately versus together in the same run.

C. Saves time and effort for running experiments with multiple tools.

Running the same job repetitively also wastes a significant amount of time and computing resources. But now with our work, users can load all tools at the same time for an application run. We used the same set of experiments mentioned above and recorded the time of finishing each run. The result (Figure 4) shows running with two tools only takes 50.2% of the time of the sum of running them separately.

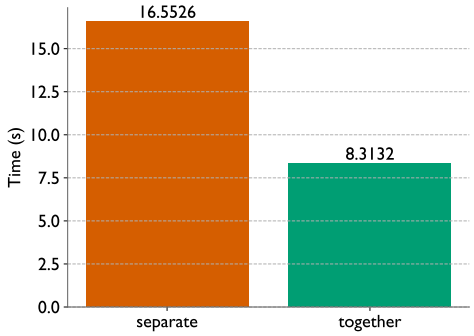

Fig. 4. Time spent in getting both Recorder & Darshan profiles when the tools are used separately versus used together in the same run.

IV. Conclusion

To solve the issue that multiple HPC tools cannot work together in the same run, we utilized Gotcha to chain tools together. We have successfully made Recorder and Darshan Gotcha enabled, and experiments showed they work the same as their original version. We also demonstrated our work benefit research using tools in three ways: being able to measure tools' performance, reducing the effect of system noises for the same run with different tools, and saving time. Next, we may formulate a standard way to make tools Gotcha enabled. We are also going to see how performance overheads change with more tools chained together.

REFERENCES

[1] C. Wang, J. Sun, M. Snir, K. Mohror, and E. Gonsiorowski, "Recorder 2.0: Efficient parallel i/o tracing and analysis," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 1–8.

[2] "Darshan." [Online]. Available: https://www.mcs.anl.gov/research/projects/darshan

[3] L. Foundation, "Dynamic linking." [Online]. Available: https://refspecs.linuxfoundation.org/ELF/zSeries/lzsabi0_zSeries/x2251.html

[4] A. Moody, D. Sikich, N. Bass, M. J. Brim, C. Stanavige, H. Sim, J. Moore, T. Hutter, S. Boehm, K. Mohror, D. Ivanov, T. Wang, C. P. Steffen, and U. N. N. S. Administration, "Unifyfs: A distributed burst buffer file system - 0.1.0," 10 2017. [Online]. Available: https://www.osti.gov//servlets/purl/1408515