Chimbuko: A Workflow-Level Scalable Performance Trace Analysis Tool

C. Kelly

Submitted to the ISAV'20 In Situ Infrastrectures for Enabling Extreme- ScaleAnalysis and Visualization USDOE O Conference
to be held at Virtual
November 09 - 19, 2020

Computational Science Initiative
Brookhaven National Laboratory

U.S. Department of Energy
USDOE Office of Science (SC), Advanced Scientific Computing Research (SC-21)

DISCLAIMER

Chimbuko: A Workflow-Level Scalable Performance Trace Analysis Tool

Christopher Kelly, Sungsoo Ha
ckelly@bnl.gov,sungsoha@amazon.com
Brookhaven National Laboratory, Amazon Web Service

Kevin Huck, Hubertus Van Dam, Line Pouchard
khuck@cs.uoregon.edu,{hvandam,pouchard}@bnl.gov
University of Oregon, Brookhaven National Laboratory

Gyorgy Matyasfalvi, Li Tang
matyasfalvi@princeton.edu,ltang@lanl.gov
Princeton University, Los Alamos National Laboratory

Nicholas D'Imperio, Wei Xu, Shinjae Yoo,
Kerstin Kleese Van Dam
{dimperio,xuw,sjyoo,kleese}@bnl.gov
Brookhaven National Laboratory

ABSTRACT

Due to the sheer volume of data it is typically impractical to analyze the detailed performance of an HPC application running at-scale. While conventional small-scale benchmarking and scaling studies are often sufficient for simple applications, many modern workflow-based applications couple multiple elements with competing resource demands and complex inter-communication patterns for which performance cannot easily be studied in isolation and at small scale. This work discusses Chimbuko, a performance analysis framework that provides real-time, *in situ* anomaly detection. By focusing specifically on performance anomalies and their origin (*aka* provenance), data volumes are dramatically reduced without losing necessary details. To the best of our knowledge, Chimbuko is the first online, distributed, and scalable workflow-level performance trace analysis framework. We demonstrate the tool's usefulness on Oak Ridge National Laboratory's Summit system.

KEYWORDS

Performance Trace, Benchmark, Profiling, Anomaly Detection, Visualization, Provenance

ACM Reference Format:
Christopher Kelly, Sungsoo Ha, Kevin Huck, Hubertus Van Dam, Line Pouchard, Gyorgy Matyasfalvi, Li Tang, and Nicholas D'Imperio, Wei Xu, Shinjae Yoo, Kerstin Kleese Van Dam. 2020. Chimbuko: A Workflow-Level Scalable Performance Trace Analysis Tool. In *Proceedings of SC Conference (Conference'20).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Many modern HPC applications comprise multiple tightly-coupled components executing concurrently that exchange information and compete for hardware resources [4]. These *workflows* are becoming more prevalent due to the growing disparity between computational power and I/O capability, which often necessitates some form of *in situ* data analysis and reduction. Due to the complex intercommunication patterns and resource contention, assessing performance and identifying possible bottlenecks in large-scale jobs requires the development of novel tools capable of capturing this interaction and coping with the tremendous data volumes generated by tracing multiple applications, while simultaneously preserving salient features and avoiding impacts on application performance.

The Chimbuko performance analysis tool seeks to address these problems by performing *in situ* analysis of trace data generated by the TAU tracing and profiling tool [7], focusing specifically on performance anomalies. By capturing only the anomalous events and sufficient information for a root cause analysis (*provenance*) the amount of data is reduced dramatically to a level that can be more easily stored and analyzed by the application developer.

Chimbuko differs from existing tools by its ability to integrate with and perform scalable, trace-level, realtime performance analysis on complex workflows, allowing for the identification of bottlenecks stemming, for example, from inefficient communication patterns or resource contention. The novel, distributed design allows this analysis to be performed in real-time with minimal overhead. Detailed anomaly provenance is stored in a database for offline analysis and an online visualization component enables monitoring and analyzing performance anomalies as the application is running.

We demonstrate the capabilities of our tool using a workflow based on the NWChem computational chemistry code [10] running at large scale on the Summit supercomputer. Chimbuko is an open source project and the code is freely available on GitHub [14].

2 CHIMBUKO ARCHITECTURE

Our goal is to provide a tool for performance trace analysis that can diagnose workflow-level performance anomalies and is scalable to thousands of nodes and tens of thousands of concurrent processes. The application comprises four components: the TAU tracing tool, the online anomaly detection module (AD), the provenance database, and the visualization module. These components are laid out according to the architecture described in Figure 1.

At the lowest level the TAU-instrumented application communicates trace information in real-time to the AD instances (one per rank and process) that perform the *in situ* trace analysis, filtering out anomalous events and gathering provenance information that

Conference'20, Nov 2020, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

is subsequently stored in the centralized provenance database, thus avoiding the need for aggregating trace data across the workflow.

In order to perform reliable judgements on the anomaly status of trace events, the AD instances synchronize the parameters of the anomaly detection algorithm with a centralized parameter server (PS) which forms the second component of the AD module. The PS also aggregates statistics on performance (i.e. profile data) and anomalies that are passed to the visualization module.

The visualization module produces real-time displays of statistical information gathered by the parameter server and also allows for online interaction with the data stored in the provenance database.

While designed primarily on online analysis, offline analysis is also supported allowing the analysis to be performed remotely. In the following sections, we detail the implementation of these components.

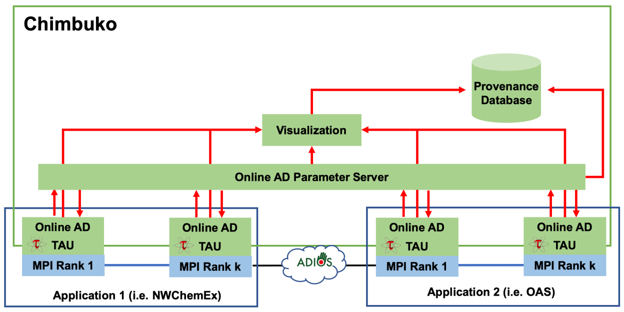

Figure 1: Chimbuko architecture diagram describing the layout of the major components: TAU, the online AD module, provenance database and visualization. The image illustrates two concurrently running applications.

2.1 TAU

The TAU Performance System [7] is a portable profiling and tracing toolkit capable of supporting parallel programs written in Fortran, C/C++, Python and other languages as well as GPU codes written in CUDA, OpenACC and other APIs.

For our purposes TAU is configured to capture trace data of running processes including MPI communications, I/O actions and GPU kernel activity, which are passed to Chimbuko by means of TAU's ADIOS2 plugin. For online analysis we utilize the ADIOS2 *Sustainable Staging Transport* (SST) engine which provides step-based communication whereby trace data is aggregated over a configurable time period (typically 1 second) and sent via DMA to the receiver AD process. Offline analysis can be performed by storing the trace using the alternative, BPFile (binary) engine.

TAU provides Chimbuko with three classes of data: *metadata* regarding the system characteristics; *counters*, which are instantaneous measurements of system and performance parameters such as memory usage, GPU temperature, etc; and *timers* which contain the trace data for function executions and MPI communications.

2.2 Online AD module

Instances of the AD module are spawned alongside each running process and perform *in situ* analysis of the *timer* trace data provided by TAU. Using these data Chimbuko forms a continuous snapshot of the execution stack for each thread and accelerator device, from which function execution objects are generated that contain the execution time, links to parent and child executions as well as all associated communication events and counters that describe the event provenance. The set of function execution objects is updated on each ADIOS2 I/O step, discarding those from the previous step to maintain a small memory footprint.

The anomaly detection algorithm is applied as a filter to the function execution list. As execution time imbalances are a major source of workflow performance variability, our current algorithm focuses on this metric. We model the function execution times as a unimodal distribution and maintain for each function the computed moments of the distribution (mean, variance, skewness, etc). Function executions are tagged as anomalous if they have extraordinary execution times, i.e. if the execution time deviates from the mean by more than α standard deviations, where α is a tunable parameter that is set to 6 for our present studies.

The moments of the function distributions are computed on-the-fly per Ref. [6] such that the algorithm becomes more reliable as the run progresses, although Chimbuko also supports importing of previously computed function statistics if available. Robust statistics are obtained rapidly by taking advantage of the native parallel nature of the applications: each AD instance synchronizes and merges its parameters with a globally aggregated set of statistics maintained on the PS. By performing this global update our experiments show that we achieve only a minor reduction in accuracy relative to a run with predetermined parameters.

The final role of the AD is to collect prescriptive provenance information on the detected anomalies. We collect all associated counters and metadata, including the full function stack and, for GPU kernel events, information about the device and the parent CPU function. We also collect a "window" snapshot of some number (5 in our present studies) of function executions occurring before and after the anomalous execution on the same thread/device, from which contextual information can be obtained. The provenance data is communicated directly from each AD instance to the provenance database component. For comparison a small number of normal (non-anomalous) executions are also send to the database. Statistics on function anomalies and execution times are also sent to the PS.

2.3 Online AD Parameter Server

The online AD Parameter Server (PS) fulfils two distinct roles: it maintains the globally aggregated parameters of the anomaly detection algorithm, and acts as a server for information provided to the visualization component. A single instance of the PS is typically placed on the head node of the job allocation.

Communication with the AD modules is performed using the ZeroMQ library with the PS acting as a simple RPC service provider. A threaded master/worker model is used to service multiple simultaneous requests and prevent bottlenecks.

On each I/O step the AD modules send function execution statistics measured on that step. The PS combines these data with the current global state and returns the updated statistics to the AD module. After selecting anomalous events the AD modules return statistics on the number of anomalies (total and by function) and on various counters. The PS forwards these statistics, aggregated over multiple steps, to the visualization module periodically using the TCP/IP protocol via libcurl providing to the user an overview of the performance trace analysis results in real-time.

2.4 Provenance Database

The provenance database maintains detailed provenance information on each anomaly provided by the AD instances. A single instance of this component is typically placed on the head node.

The provenance database component is implemented using the Mochi Sonata framework which connects a serverless document-store (JSON) UnQLite database to an RPC engine capable of handling database storage and query from many connected clients. Database queries of arbitrary complexity can be formulated in the jx9 query language. A subset of queries can be submitted dynamically via the visualization module, and a database query tool is provided to allow for more detailed offline analysis.

2.5 Visualization Module

The visualization module provides a real-time inspection of the identified anomalous behaviors. Dynamic performance statistics are displayed and deeper investigation can be performed by selecting a specific time interval or function.

The real-time performance statistics are provided periodically by the parameter server and detailed anomaly data are stored in the provenance database. Therefore, the visualization module has three major roles: 1) receiving and processing the streaming statistics from parameter server, 2) querying the provenance database and processing the queried data, and 3) visualizing the processed data to users and responding to the user interactions. The goal is to provide a scalable server that is able to digest requests asynchronously with minimal latency or memory overhead, and can handle concurrent requests or long running tasks of the connected clients.

To meet these requirements, the visualization server has been redesigned from previous works[12][13] to have two levels of scalability [5]. At the first level, uWSGI [9] is adopted to handle multiple concurrent connections. At the second level, the requests are distributed to celery workers [2] and handled asynchronously for both short and long running tasks. Finally, streaming (or broadcasting) data to the connected users is completed by using Websocket technology with socket IO library [8].

Below we discuss two frontend visualizations by which we present data in a "overview first, zoom and filter, then details on-demand" mechanism [1].

- **Dynamic Statistics Visualization** Streaming data from the PS is processed into a number of anomaly statistics including the average, standard deviation, maximum, minimum and the total number of anomalous function executions. Users can select a statistic along with the number of ranks for which it is visualized. A dynamic "ranking dashboard" of the most problematic MPI ranks with rank-level granularity is provided.

Selecting corresponding ranks activates the visualization server to broadcast the number of anomalies per time frame (e.g., per second) of these ranks to the connected users while performance traced applications are running. This streaming scatter plot serves as a time frame-level granularity by showing the dynamic changes of anomaly amount of a MPI rank within a time frame.

- **Detailed Functions Visualization** This visualization is designed to retrieve data from the provenance database and show the function execution details. It consists of two parts: a function view and a call stack view. In the function view, it visualizes the distribution of functions executed within a selected time interval. The distribution can be controlled by selecting the X- and Y-axis among different function properties.

In the call stack view, users can more closely investigate a selected function execution in details. The invocation relationships among functions and their communications over other ranks are presented for users to interpret the potential cause of the anomalous behavior. An example will be illustrated in Sec3.

Table 1: Chimbuko overhead over NWChem execution time

# MPI	80	160	320	640	1280	2560
without Chimbuko	1.85	2.60	5.13	6.92	8.54	18.27
with Chimbuko	1.31	2.13	5.53	6.85	16.67	24.56

3 EXPERIMENT AND RESULTS

In this work, we adopt the NWChemEx exascale computing project for the demonstratation. NWChemEx targets a range of computational chemistry methods, from molecular dynamics (MD) to high-order many-body methods. It performs MD simulations of about one million atoms on $O(1\ \mu s)$ timescales, taking roughly one billion time steps to complete [3]. The resulting time sequence comprises a large volume of data, and *in situ*, concurrent analysis is necessary to target the dynamics of interest. This workflow is an ideal target for Chimbuko due to the potential impacts on scalability of the complex interaction between the analysis and simulation components. Of particular interest are performance issues related to intra- and inter-node communications arising from resource contention or load imbalance.

As NWChemEx is under development we have instead modified NWChem [10] to include an *in situ* analysis component that communicates with the simulation using ADIOS2. We consider a system comparable in scale (1.2 million atoms) and complexity to those that are targeted for running under NWChemEx on next-generation systems. The simulations were performed on the Summit [11] supercomputer at the Oak Ridge Leadership Computing Facility (OLCF).

We utilize TAU's native filtering capability to remove from the trace functions that are unlikely to result in performance bottlenecks. Below we measure the impact of this filtering upon the trace data volume. Note that the recently introduced provenance database component was not included in these tests.

We focus initially on the overhead of the application, comparing the native runtime to that with TAU instrumentation but without Chimbuko, and with both TAU and Chimbuko. In Tab. 1 we show the overhead of these two cases as defined by the relative increase in runtime, as a function of the number of MPI ranks, computed as the average over 15 independent runs. For less than 1000 MPI processes

Chimbuko results in negligible additional overhead compared to running with TAU alone. Beyond this point the overhead becomes more significant, roughly 8%, and we are investigating the cause of this jump. Nevertheless this increase in cost is likely acceptable given the data reduction volumes we demonstrate below.

In Fig. 2 we plot the volume of trace data as a function of the number of MPI ranks for NWChem/TAU running with and without Chimbuko. We also compare those with and without the initial filtering of the trace data described above. We achieved averages of 14x and 95x reductions in data volume for filtered and unfiltered (full) data, respectively; for the largest job we achieved 21x and 148x data reduction, respectively, reducing 2, 300 GB of raw data to 15.5 GB in the unfiltered case, and 117.5 GB to 5.5 GB in the filtered, at only a 6% runtime increase.

Despite the reduction in data volume, Chimbuko was able to provide significant insights into the origin of the detected performance anomalies. We illustrate this in the form of a short case study that serves also to highlight the features of the online visualization module: The scientist was specifically interested in finding anomalies in one of the major simulation functions, "MD_NEWTON". By first visualizing the top 5 anomalous ranks, Rank 1164 was selected and the dynamic scatter plot for the step-wise anomaly status was tracked. A number of consecutive steps reported normal execution for this function such as step 70 shown in upper pane of Fig. 3. However, one execution was identified as anomalous in step 86, shown in the bottom pane of that figure, taking almost 3x longer to execute than is typical. By comparing the event window plots the scientist was able to associate the decreased performance to a delay in the launch of the child function "MD_FORCES".

4 CONCLUSION

In this work we presented Chimbuko, a performance analysis framework for real-time, distributed streaming anomaly detection and visualization designed to support complex workflows running at the exascale. Through *in situ* analysis of trace data, the volume of performance data can be reduced by two orders of magnitude with only a small additional overhead as we have demonstrated in a study of the NWChem application running on Summit. This data reduction is achieved while retaining important contextual provenance information that can be used to diagnose performance abnormalities. A dynamic visualization component allows the user to monitor and find details on anomalous executions in real time.

In future work we will continue to improve the overall performance to further minimize overheads, and also aim to improve the anomaly detection capability with a more advanced algorithm. Additional features will be added to the visualization module to allow greater insight into the application performance.

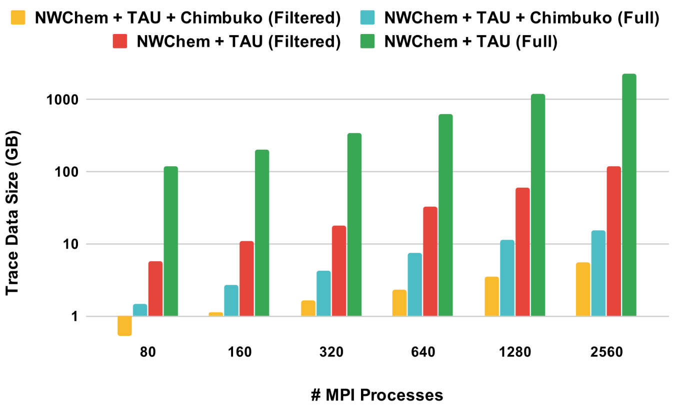

Figure 2: Trace data size over MPI processes.

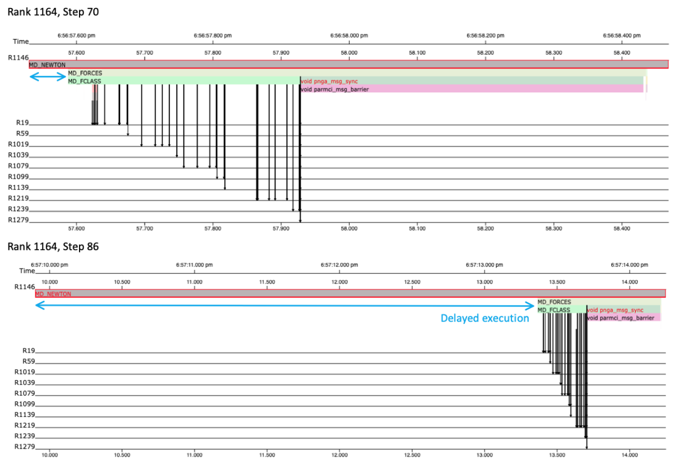

Figure 3: The call stack views of the NWChemEx application showing the function delay.

ACKNOWLEDGMENTS

This project is supported by ECP CODAR project 17-SC-20-SC.

REFERENCES

[1] Mackinlay Card. 1999. *Readings in information visualization: using vision to think.* Morgan Kaufmann.
[2] Celery. 2019. Celery: Distributed Task Queue. http://www.celeryproject.org/
[3] BNL CODAR. 2019. CODAR: Advanced data performance analysis for next-generation applications. ECP. Submitted: 2019-04-04.
[4] Ewa Deelman et al. 2018. The future of scientific workflows. *The International Journal of High Performance Computing Applications* 32, 1 (2018), 159–175.
[5] Miguel Grinberg. 2016. Flask At Scale. https://speakerdeck.com/pycon2016
[6] Philippe Pierre Pebay. 2008. *Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments.* Technical Report. Sandia National Laboratories.
[7] Sameer S. Shende and Allen D. Malony. 2006. The Tau Parallel Performance System. *Int. J. High Perform. Comput. Appl.* 20, 2 (May 2006), 287–311. https://doi.org/10.1177/1094342006064482
[8] socketio. 2020. socketio. https://socket.IO
[9] uWSGI. 2019. uWSGI. https://uwsgi-docs.readthedocs.io/en/latest/
[10] Marat Valiev et al. 2010. NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications* 181, 9 (2010), 1477–1489.
[11] Sudharshan S. Vazhkudai et al. 2018. The design, deployment, and evaluation of the CORAL pre-exascale systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis.* IEEE Press, 52.
[12] C. Xie, J. Wang, G. Matyasfalvi, H. Van Dam, K. Mueller, S. Yoo, and W. Xu. 2019. Exploratory Visual Analysis of Anomalous Runtime Behavior in Streaming High Performance Computing Applications. *ICCS* (June 2019).
[13] C. Xie, W. Xu, and K. Mueller. 2018. A Visual Analytics Framework for the Detection of Anomalous Call Stack Trees in High Performance Computing Applications. *IEEE Transactions on Visualization and Computer Graphics* 25 (2018), 215–224.
[14] Shinjae Yoo et al. 2019. Chimbuko Framework. https://github.com/CODARcode/Chimbuko