

Chimbuko V-2.0

Chimbuko: CODAR Framework for Performance Analysis and Visualization

Version of December 18, 2018

Kerstin Kleese Van Dam, Shinjae Yoo, Wei Xu, Line Pouchard, Li Tang, Gyorgy
Matyasfaivi, Cong Xie, Wonyong Jeong, and Hubertus Van Dam
Brookhaven National Laboratory
Kevin Huck
University of Oregon

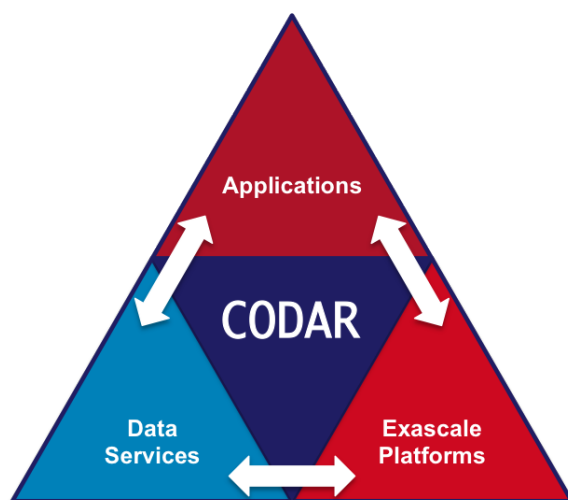


Table of contents

1	Introduction	1
2	Stakeholders	2
2.1	ECP Applications	2
2.2	Software Technologies	2
3	Use case	2
3.1	NWCHEM	2
3.2	NWCHEM with ADIOS	3
4	Related work	4
5	Design	4
5.1	Introspection (TAU + SOSFlow)	4
5.2	Prescriptive Provenance	5
6	Chimbuko Data Analysis Module	7
6.1	Software Requirement	7
6.2	Installation	7
6.3	Test	7
7	Chimbuko Visualization Module	8
7.1	Software Dependency	8
7.2	Installation	9
7.3	Using docker	9
7.4	Manual install	9
7.5	Execution	9
8	Success metrics	9

Chimbuko: CODAR Framework for Performance Analysis and Visualization

Kerstin Kleese Van Dam, Shinjae Yoo, Wei Xu, Line Pouchard, Li Tang, Gyorgy Matyasfalvi, Cong Xie, Wonyong Jeong, and Hubertus Van Dam
Brookhaven National Laboratory
Kevin Huck
University of Oregon

1 Introduction

The Chimbuko framework captures, analyzes, and visualizes performance metrics for complex scientific workflows that enables investigations into co-design tradeoffs for online data analysis and reduction on extreme-scale machines.

Chimbuko helps to compare different runs at high and low levels of metric granularity. Chimbuko provides this capability in both offline and online (in-situ) modes. Because capturing performance metrics can quickly escalate in volume and provenance can be highly verbose, Chimbuko includes a data analysis module for data reduction.

Chimbuko enables co-design studies by allowing scientific applications and workflows to profile their execution patterns on traditional and heterogeneous architectures. The detailed metrics and information (a.k.a. provenance of the execution) are extracted and then reduced and visualized by the Chimbuko data analysis and visualization modules, thus providing insights into the behavior of the codes and workflows at scale. Using provenance, Chimbuko provides these insights for both single applications and the complex orchestrated workflows that are becoming more prevalent to organize complex code execution.

Chimbuko version 2.0 provides performance visualization and data analysis in an online mode. The data analysis part provides a Python API to process TAU performance profile and traces. Its modules support the following functionalities:

- Parser: processes a given TAU trace (both streaming and batch through Adios).
- Event: keeps track of event information such as function call stack and function execution times.
- Outlier: detects outliers in performance of functions.
- Visualizer: provides an interface to Chimbuko's visualization software (both online and offline through requests API).

Chimbuko Visualization Framework provides online performance analysis. This framework mainly focuses on visualizing real-time anomalous behaviors in a High Performance Computing application so that any patterns of anomalies that users might not have recognized can be effectively detected through online visual analytics.

2 Stakeholders

2.1 ECP Applications

Stakeholders include scientific applications that are part of the ECP program. We have consulted many and received positive feedback regarding the usefulness to them of a system such as Chimbuko. ECP applications that exhibit unexplained variability in performance, strong data reduction needs, and different types of workflows are poised to benefit from the Chimbuko technology.

2.2 Software Technologies

ECP software technology projects such as ADIOS, Cinema, Swift, SZ, and ZFP are all potential contributors of software components that may be integrated into the Chimbuko framework.

3 Use case

Efficient and effective performance analysis techniques are critical for the development of future generation systems. They are the drivers behind the required co-design process that helps establish the principles needed for their design.

For the current release, we used the NWChem application as use case to study performance analysis techniques for co-design of data reduction using Chimbuko.

3.1 NWCHEM

The part NWChemEx project with whom we worked studies processes involving transmembrane proteins and zeolite catalysts. The processes of interest require the calculation of free energies and the dynamics of the molecular structures. In addition, to simulate realistic molecular environments, the molecular structures need least 100,000 atoms, different regions need to be evaluated at different levels of approximation, and the simulations need to work with time steps of about one femtosecond.

The simulation requires the evaluation and forces to update the molecular structures. While these calculations are executed, the statistics needed for the free energies are collected. In addition, selected structures along the trajectory are stored, so that additional properties for those structures can be calculated. These additional properties facilitate comparing the calculated trajectories against experimental observations. Hence, the workflow includes a large number of cores calculating the energies and forces and a small number of cores analyzing the results and storing selected time steps for more detailed simulations.

A large number of parameters define the total amount of work performed in particular parts of the simulation and varying the amount of work change the optimal work distribution. Performance characteristics need to be recorded in a way that enables comparison against prior simulations to establish the figures of merit of the development. This requires capturing some base characteristics that are always the same. For specific performance optimization, we need to capture the performance of specific parts of the code, depending, for example, on the functionality of interest, on the characteristics of the data distribution, or on the granularity of the tensor blocks and associated task sizes. The sheer volume of the data requires special online performance techniques and data reduction.

3.2 NWChem with ADIOS

The framework depends on the ADIOS framework [7] for managing the data stream between the components. The NWChem application was extended with the ADIOS APIs for its data management work. The detail about the implementation can be found on the following links:

- https://github.com/hjjvandam/nwchem-1/tree/pretauaudio/contrib/codar_integration
- <https://github.com/hjjvandam/nwchem-1/tree/pretauaudio>

4 Related work

Workflows are increasingly important in orchestrating complex scientific processes in extreme scale and highly heterogeneous environments. To date, however, we cannot reliably predict, understand, and optimize workflow performance. Sources of performance variability and in particular the interdependencies of workflow design, execution environment, and system architecture are not well understood. While there is a rich portfolio of tools for performance analysis modeling and prediction for single applications in homogeneous computing environments [3, 4, 5, 6], these are not sufficient for workflows, due to the number and heterogeneity of the involved workflow and system components and their strong interdependencies. In addition, no tool currently tracks the performance of workflow components as it relates to provenance. A literature review to support these claims has been published [1].

5 Design

The main design components are: (1) Introspection, (2) Provenance, (3) Data Analysis and (4) Visualization. Figure 1 shows the main components of Chimbuko and their interactions.

- Chimbuko monitors workflow execution, extracts provenance and visualize data
- ADIOS orchestrates workflows (blue line) and provides data streaming
- TAU [4] provides performance metrics for instrumented components 1 and 2
- TAU extracts provenance metadata and trace data (green lines)
- SOSFlow [2] stores and aggregates the data at each node
- Trace data is dynamically analyzed to detect anomalies (solid red line)
- Selected metadata and trace data is stored (e.g., time window for which trace event is interesting) (dashed red lines)

5.1 Introspection (TAU + SOSFlow)

A performance monitoring enables online access to performance data. This consists of two parts:

1. Monitoring Plugin Support: TAU provides a plugin framework where an event stream is exported by each MPI rank. An analysis plugin subscribes to events such as initialization/finalization, metadata values, interval event timers, and counters. The SOS client plugin monitors the data stream at runtime and provide aggregation/filtering/reduction utilities. It also provides a feedback path to TAU in order to increase/decrease resolution of measurement (sampling rate, callpath depth, hardware counters, etc.). See Figure 2.
2. Online Monitoring Access: Profile, event trace, and metadata events are provided to the analysis through SOS [2]. In a coupled application/workflow scenario, SOSFlow regularly/periodically aggregates TAU data over the SOSFlow infrastructure. This data is extracted from SOS through an SOS analysis extension that helps output the aggregated event stream as ADIOS streaming output. The ADIOS stream is used by the trace analysis tools and anomaly detection methods. See Figure 3.

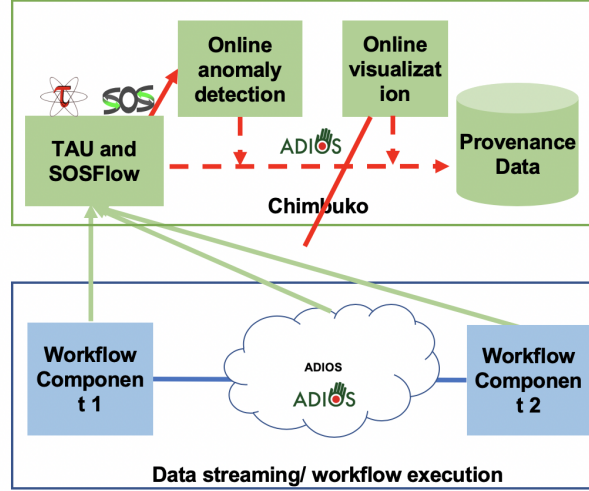


Figure 1: Chimbuko's main components.

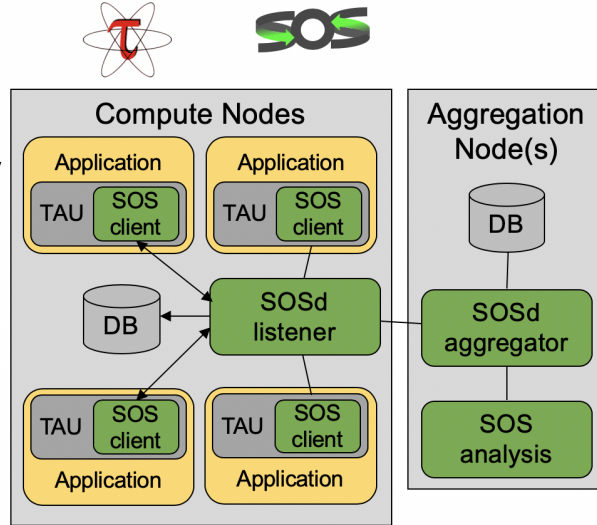


Figure 2: TAU and SOS Interaction.

5.2 Prescriptive Provenance

Provenance can be very verbose and persisting provenance for an entire run is impractical. In the future, we will persist detailed provenance *only* for the anomaly events or events of interest detected by online performance analysis. Ultimately, provenance is to be persisted in SOSFlow for a moving window prior to and posterior to an event with the additional capability to increase or decrease the verbosity of provenance (size and granularity of the moving window) selected for storage around the event. In addition, static information describing the system, runtime configuration, and workflow or application metadata similar to what is already extracted for offline analysis will be persisted at the start and end of the run (static information). This prescriptive provenance, provenance selection and use prescribed by the results of performance data analysis, is an end product of the analysis after training sets have been built. Specifications of the moving window of provenance prior to an outlier event will be studied for tradeoffs. For instance, we will study if this window is defined in terms of execution time or the number of time steps. Constraints on size

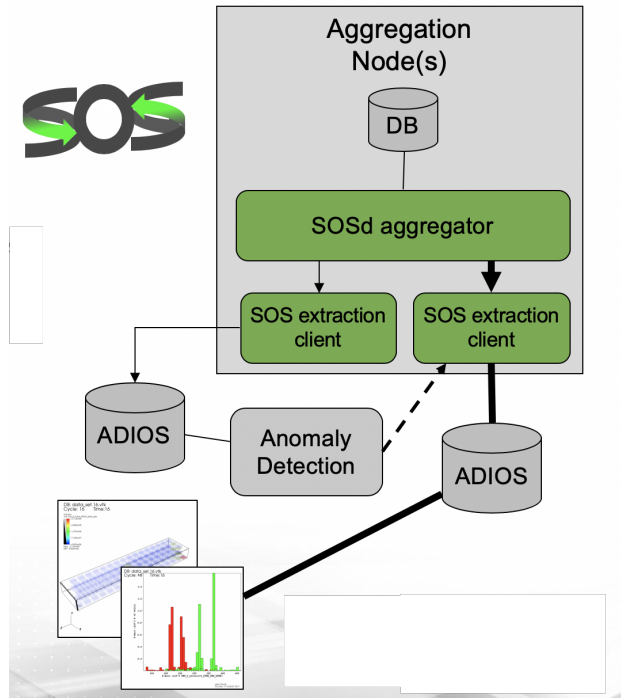


Figure 3: Chimbuko Online Monitoring.

and tradeoffs in the amount of details needed will be balanced with usefulness to a scientific code developer. This prescriptive provenance approach will help reduce the information passed to the data analysis and performance visualization components.

Figure 4 explains prescriptive provenance. Suppose we detect an abnormal behavior in performance for a communication intensive workflow component. The event starts at timestamp T2 and ends at timestamp T3. To analyze this event, we need all the communication information within a certain time window at the start of the event and at the end of the event. In Figure 4, X shows the time window at the start of the event till timestamp T1. The software, hardware, and system information within this time window will be correlated with the communication patterns and performance behavior of the workflow when the event occurred. Figure 5 shows the structure of provenance database.

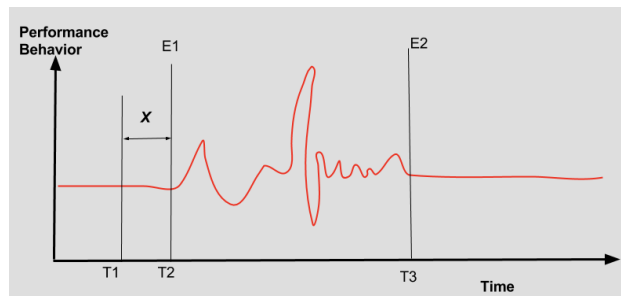


Figure 4: Prescriptive Provenance.

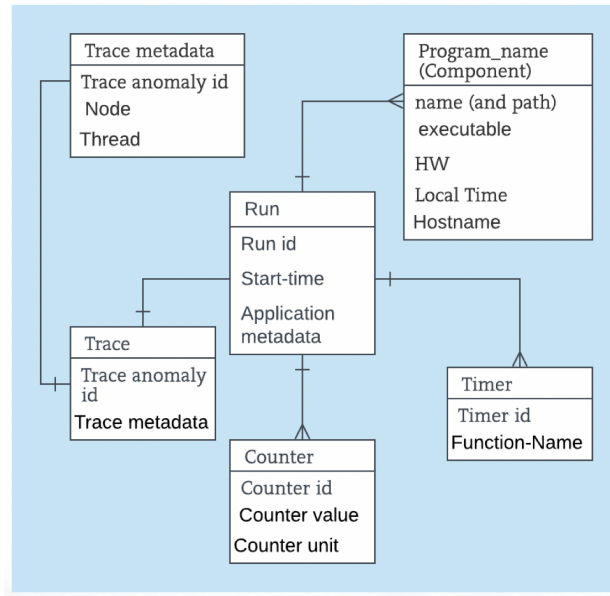


Figure 5: Prescriptive Database.

6 Chimbuko Data Analysis Module

This library provides a Python API to process TAU performance profile and traces. Its modules support the following functionalities:

- Parser: processes a given TAU trace (both streaming and batch through Adios).
- Event: keeps track of event information such as function call stack and function execution times.
- Outlier: detects outliers in performance of functions.
- Visualizer: provides an interface to Chimbuko's visualization software (both online and offline through requests API).

6.1 Software Requirement

Our codebase requires Python 3.5 or higher and python and pip to be linked to Python 3.5 or higher. All additional requirements can be found in the requirements.txt file.

6.2 Installation

Run the following script:

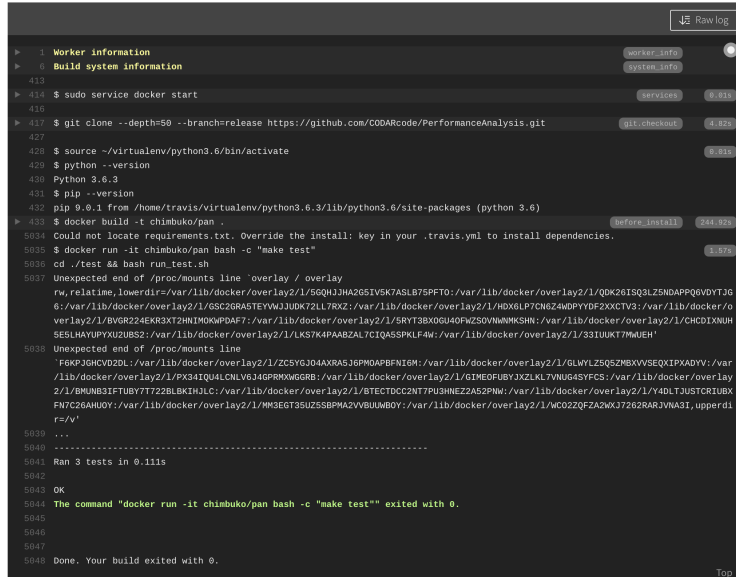
```
scripts/install-dependency.sh
```

```
bash scripts/install-dependency.sh
```

6.3 Test

To run tests:

```
make make test
```



```
413 $ sudo service docker start
414
415 $ git clone --depth=50 --branch=release https://github.com/CODARcode/PerformanceAnalysis.git
416
417 $ source ~/.virtualenv/python3.6/bin/activate
418
419 $ python --version
420 Python 3.6.3
421
422 $ pip --version
423 pip 9.0.1 from /home/travis/.virtualenv/python3.6.3/lib/python3.6/site-packages (python 3.6)
424
425 $ docker build -t chimbuko/pan .
426 Could not locate requirements.txt. Override the install: key in your .travis.yml to install dependencies.
427
428 $ docker run -it chimbuko/pan bash -c "make test"
429
430 cd ./test && bash run_test.sh
431
432 Unexpected end of /proc/mounts line overlay / overlay
433 rw,relatime,lowerdir=/var/lib/docker/overlay2/1/56QHJHA2G51V5K7ASLB75PFT0:/var/lib/docker/overlay2/1/QDK261S93LZ5NDAPPQ6VDVTJG
434 6:/var/lib/docker/overlay2/1/GSC26RASTEYVMJ3UDK72LL7RXZ:/var/lib/docker/overlay2/1/HDX6LP7CN6Z4WDPPYDF2XXCTV3:/var/lib/docker/o
435 verlay2/1/BVGR224EK83XT2HIMOKMPDAF:/var/lib/docker/overlay2/1/5RYT3BXDGU4DFWZSDVNMNMKSHN:/var/lib/docker/overlay2/1/CHCDIXNUH
436 SESLHAYUPYXU2UBS2:/var/lib/docker/overlay2/1/LK57K4PAABZAL7C1QASPKLF4W:/var/lib/docker/overlay2/1/S31UUKT7MUJH'
437
438 Unexpected end of /proc/mounts line
439
440 FBKP3GHCVD2L:/var/lib/docker/overlay2/1/ZCSYG304AXRASJ6PMDAPBFN18M:/var/lib/docker/overlay2/1/GLWYLZ5Q5ZMBXVVEQXIPXADYV:/var
441 /lib/docker/overlay2/1/PX34IQH4LCLNV6342PM0G6GRB:/var/lib/docker/overlay2/1/GIMEDFUBYIXZLKL7VNUG4SYFCS:/var/lib/docker/overl
442 ay2/1/BMUNB31FTUBYT7722BLBK1HJLC:/var/lib/docker/overlay2/1/6TECTDC2NT7PU3HNEZ2A52PM:/var/lib/docker/overlay2/1/Y4DLT3USTCRIUBX
443 FN7C26AHUOY:/var/lib/docker/overlay2/1/MM3EGT35UZ5SBPMA2VVBUMBDY:/var/lib/docker/overlay2/1/WC022QF2A2WJ37262R8RJVN43I,upperdi
444 r=/v'
445
446 ...
447
448 -----
449 Ran 3 tests in 0.111s
450
451 OK
452
453 The command "docker run -it chimbuko/pan bash -c "make test"" exited with 0.
454
455 Done. Your build exited with 0.
```

Figure 6: Build Test for Data Analysis Module

7 Chimbuko Visualization Module

This is a visualization framework for online performance analysis. This framework mainly focuses on visualizing real-time anomalous behaviors in a High Performance Computing application so that any patterns of anomalies that users might not have recognized can be effectively detected through online visual analytics.

This framework provides four major features:

- Streaming data reduction and aggregation
- Sliding time window of workflow overview for regular and anomaly function executions
- Selected function execution in a reduced call stack tree structure
- Selected function execution and message passing details in zoomable timelines
- Overall events distribution by rank

The following four visualization components are provided:

- Projection of Function Execution
- Selected Call Stack Tree
- Timeline and Message Visualization
- Execution Distribution by Rank

7.1 Software Dependency

This framework is a web application for which the back-end was built with Python3.x and Flask and the front-end was developed with Javascript and D3.js.

7.2 Installation

Please be advised that this framework is containerized as a single docker image so that there is no need to spend time on any issues with the software dependency.

7.3 Using docker

The docker repository is currently private. The access information will be added soon.

7.4 Manual install

In order to manually install this framework, make sure that Python3.x and pip3 are installed on your machine.

If they are already installed, Numpy and Flask can be installed by:

```
pip3 install Numpy
```

```
pip3 install Flask
```

Now, please clone the repository of this project to run the visualization framework by:

```
git clone https://github.com/CODARcode/Chimbuko.git
```

7.5 Execution

Before starting a visualization server, a port number can be modified if needed. Please open main.py:

```
port = 5000 replace 5000 with your preference Start visualization server:
```

```
python3 main.py
```

If the visualization server runs, by default, localhost:5000 will work.

8 Success metrics

The online version of Chimbuko provides a runtime view of different features important to complex scientific simulations and workflows. This information will allow us to manipulate runtime configuration parameters during the execution phase to improve performance of a given workflow a new machines. Our success metrics will be the number of co-design studies and applications that can assess and visualize the performance metrics through Chimbuko. Another success metrics is our ability to explore the performance of co-design studies by interacting with the visualization interface, which will show the outliers identified, the function call details in a call tree structure, and other information.

References

- [1] *Enabling Structured Exploration of Workflow Performance Variability in Extreme-Scale Environments*: Kleese van Dam, K., Stephan, E.G., Raju, B., Altintas, I., Elsethagen, T.O., and Krishnamoorthy, S., Proc. 8th Workshop in Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS) collocated with SC 2015.
- [2] C. Wood et al.,: *A Scalable Observation System for Introspection and In Situ Analytics-2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, Salt Lake City, UT, 2016, pp. 42-49. doi: 10.1109/ESPT.2016.010
- [3] Lammel S., Zahn F., Frning H. (2016) *SONAR: Automated Communication Characterization for HPC Applications*. In: Taufer M., Mohr B., Kunkel J. (eds) *High Performance Computing. ISC High Performance 2016*.
- [4] S. Shende, A. D. Malony, J. Cuny, K. Lindlan, P. Beckman and S. Karmesin, *Portable Profiling and Tracing for Parallel Scientific Applications using C++*, Appears in: *Proceedings of SPDT'98: ACM SIGMETRICS Symposium on Parallel and Distributed Tools*, pp. 134-145, Aug. 1998.
- [5] Knpfer A. et al. (2012) *Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir*. In: Brunst H., Mller M., Nagel W., Resch M. (eds) *Tools for High Performance Computing 2011*.
- [6] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. *HPCToolkit: Performance tools for scientific computing*. In *SC '08: Proc. of the 2008 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2008. ACM.
- [7] <https://www.olcf.ornl.gov/center-projects/adios/>