Performance Visualization for TAU Instrumented Scientific Workflows

Cong Xie[3], Wei Xu[1], Sungsoo Ha[1], Kevin Huck[2], Sameer Shende[2], Hubertus Van Dam[1],
Kerstin Kleese Van Dam[1] and Klaus Mueller[3]

[1]*Computational Science Initiative, Brookhaven National Laboratory, Upton, New York, U.S.A.*

[2]*Performance Research Lab, University of Oregon, Eugene, Oregon, U.S.A.*

[3]*Computer Science Department, Stony Brook University, Stony Brook, New York, U.S.A.*

{*xuw, sungsooha, hvandam, kleese*}@*bnl.gov*, {*khuck, sameer*}@*cs.uoregon.edu*, {*coxie, mueller*}@*cs.stonybrook.edu*

Keywords: Performance, Visualization, TAU, Scientific Workflow.

Abstract: In exascale scientific computing, it is essential to efficiently monitor, evaluate and improve performance. Visualization and especially visual analytics are useful and inevitable techniques in the exascale computing era to enable such a human-centered experience. In this ongoing work, we present a visual analytics framework for performance evaluation of scientific workflows. Ultimately, we aim to solve two current challenges: the capability to deal with workflows, and the scalability toward exascale scenario. On the way to achieve these goals, in this work, we first incorporate TAU (Tuning and Analysis Utilities) instrumentation tool and improve it to accommodate workflow measurements. Then we establish a web-based visualization framework, whose back end handles data storage, query and aggregation, while front end presents the visualization and takes user interaction. In order to support the scalability, a few level-of-detail mechanisms are developed. Finally, a chemistry workflow use case is adopted to verify our methods.

1 INTRODUCTION

Exascale systems allow applications to execute at unprecedented scales. With the increased data volume, the disparity between computation and I/O rates is more intractable, which leads to offline data analysis. Recently, a co-design center focused on online data analysis and reduction at the exascale (CODAR) (I. Foster, 2017) was founded by the Exascale Computing Project (ECP). A new infrastructure is expected for online data analysis and reduction to extract and output necessary information and accelerate scientific discovery. On the other hand, scientific workflows are commonly utilized in this scenario to schedule computational processes in parallel and coordinate multiple types of resources for different scientific applications. Thus, the capability to capture, monitor, and evaluate the performance of workflows in both offline and online modes is essential to confirm expected behaviors, discover unexpected patterns, find bottlenecks, and eventually improve the performance. Since parallel applications rely on the performance of a number of hardware, software and application-specific aspects, the captured performance evaluation data usually has multiple dimensions and disjoint attributes. This complication makes the exploration and understanding extremely challenging.

Visualization, as an indispensable technique for big data, has the capability to fuse the multidimensional and heterogeneous evaluation data, provides corresponding representations for exploration, and creates effective user interaction and steering. Specifically, performance visualization is the technique focusing on performance data of heavy computation applications. The performance data is acquired through instrumentation of a program, or monitoring system-wide performance information. It will then be depicted to users for the execution evaluation of the program. There are a number of existing instrumentations and/or measurement toolkits such as TAU (Tuning and Analysis Utilities) (Shende and Malony, 2006), Score-P (Knüpfer et al., 2012) and HPC-Toolkit (Adhianto et al., 2010). However, none of these tools can directly handle workflow performance acquisition for either offline or online modes. This lack of support from data acquisition end starves the further development of new performance visualization tools for workflows.

Therefore, we aim to provide a performance visualization framework dedicated to improving the execution performance of exascale scientific workflows. There are three major challenges: 1) the capability to instrument not only an application, but a scientific

workflow, 2) the captured performance data itself can be extremely large, heterogeneous, and collected in a streaming way, and 3) the cross-platform framework that handles data management to consume data scale, provides data aggregation, and supports real-time visualization, exploration and interaction.

In this paper, on the way to satisfy these requirements, we present a proof-of-concept framework with data acquisition and handling, as well as a few visual representation designs for workflows. Our framework is built with offline analysis use cases, but our design does not rely on the offline mechanism, thus is extensible to online. Our contributions are as following:

- An improvement on TAU instrumentation method to capture parallel workflows.

- A web-based framework connecting different types of performance data into one linked display with a variety of visual representations.

- A few level-of-detail visual methods enhancing the data exploration.

The remainder of this paper is structured as follows: Section 2 summarizes related works, Section 3 discusses our use case, Section 4 introduces the proposed framework, and Section 5 concludes the work.

2 RELATED WORKS

The general purpose of performance evaluation includes: the global comprehension, problem detection and diagnosis (Isaacs et al., 2014). Performance visualization is therefore designated to fulfill these goals. At a minimum, the design of the visualization must be able to show the big picture of the program execution. When an interesting area is targeted, users must narrow down the region and mine more detailed information. Moreover, comparative study looking for correlation or dependency must be supported. For problem detection, abnormal behaviors can be highlighted in ways that allow users to identify them easily.

Current visualization works can be grouped by their applications in four contexts: hardware, software, tasks and application (Isaacs et al., 2014). Specifically, it includes a few types of data: 1) an event table summarizing the start and end time of all function calls, 2) the message passing among cores, 3) profiling of certain metrics spent in each part of the code on each computing core, and 4) the call paths. Therefore, we only summarize the existing works that are commonly applied to our data types. Other works such as the visualization for network, system memory usage, or system logs for multicore clusters can be found in (Isaacs et al., 2014).
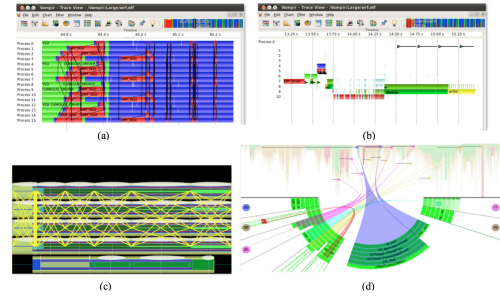

Figure 1: Trace timeline visualization examples: (a) Vampir timeline showing the execution on all processes (Knüpfer et al., 2008), (b) Vampir timeline for one process with detailed function entry and exit (Knüpfer et al., 2008), (c) the timeline of Jumpshot, and (d) the advanced visualization for focused thread comparison (Karran et al., 2013).

2.1 Trace Visualization

Tracing measurement libraries record a sequence of timestamped events such as the entry and exit of function calls or a region of code, the message passing among threads, and job initiation of an entire run. A common practice is to assign the horizontal axis to the time variable, and the vertical axis to the computation processes or threads. Different approaches are usually variations of Gantt charts.

Vampir (Knüpfer et al., 2008) and Jumpshot (Jumpshot, 2014) provide two examples of this kind of visualization. Generally, overview of the whole time period is first plotted. Then users can select interested area to reveal more detailed events happened during the selected period. Different functions or regions of code are colorized, and the black (yellow for Jumpshot) lines indicate message passing such as shown in Fig. 1. In addition, advanced visualization tools such as SyncTrace (Karran et al., 2013) provide a focus view showing multiple threads as sectors of a circle. The relationships between threads are shown with aggregated edges similar to chord diagram. Those tools can only handle small scale data.

2.2 Profile Visualization

Profiling libraries measure the percentage of the metric e.g. time spent in each part of the code. Profile does not typically include temporal information, but can quickly identify key bottlenecks in a program. Stacked bar charts, histogram, and advanced visualization in 3D are commonly used to give a comparative view of the percentage of time or other metric spent for different functions. ParaProf (ParaProf, 2014) is one example of this kind of visualization as shown in Fig. 2. It also supports the comparison of certain function calls in different execution runs.

The functions are color coded and plotted in different stacking modes. Other statistics can also be plotted for a selected function over all cores or for a selected metric correspondingly.
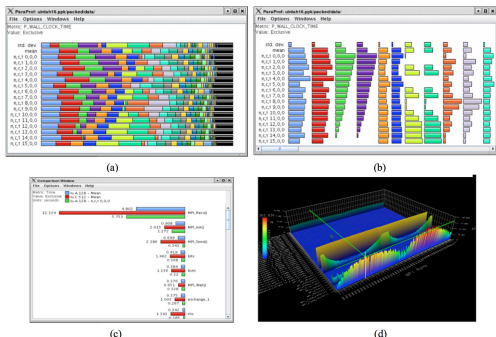

Figure 2: Profile visualization examples: (a) ParaProf showing the profile of all functions in stacked view (Para-Prof, 2014), (b) separated view (ParaProf, 2014), (c) the comparative view of different execution runs (ParaProf, 2014), and (d) the 3D visualization comparing different metrics (ParaProf, 2014).

2.3 Message Communication

As mentioned in the timeline visualization, message passing is also important. A straightforward approach is to draw a line between two functions for each message, as adopted by Vampir and Jumpshot. On the other hand, the message communication between threads or processes can also be summarized in terms of a matrix, with proper colorization indicating additional information.

2.4 Limitations

In general, although existing tools are well accepted by the scientific community and form as standard approaches, with tremendous growth in data scale and computation capability, there are yet some challenges to solve. Firstly, most approaches are not targeting workflow executions of many applications. Thus they lack the capability to illustrate the connection of workflow components and their communication. Another issue is the requirement for online evaluation. This requires online data acquisition, in memory data processing and visualization. Most existing works are however designed for offline analyses. Last but not least, the acquisition data can be heterogeneous and should be visualized in a fused and interconnected way. Our framework is established considering these aspects.

3 THE USE CASE

NWChemEx (NWChemEX, 2016), as the next generation of NWChem (Open Source High-Performance Computational Chemistry), is a scientific toolkit for simulating the dynamics of large scale molecular structures and materials systems on large atomistic complexes. In our paper, there are two modules: molecular dynamics (MD) module and the analysis module. Its workflow has a structure where the MD simulation runs in parallel, emitting snapshots of the protein structure along the trajectory, and concurrently the data analysis is triggered whenever the expected data is produced. In our use case, a classical MD run has 2200 timesteps on 4 nodes while the corresponding analysis has 1000 timesteps on 1 node. The MD execution took 309.3 seconds wall clock time in total. We selected the first 6.5s to illustrate how our visualization method works.

The major analysis task is for the scientists to monitor the overall performance of the workflow execution, and explore region of interests in details from multiple perspectives. In this way, they can apply the domain knowledge to assess the workflow healthiness and steer the experiment.

4 METHODOLOGY

4.1 TAU Data Acquisition

We adopted TAU instrumentation into the application code, as well as collected MPI inter-process communication using the standard PMPI interface (Shende and Malony, 2006). The post-processed TAU measurements include several types of data: 1) event table listing start and end time of all function calls, 2) the messages passed between processes, 3) profiling of certain metrics spent in each code region on each computing node/thread, and 4) the call path for each node/thread.

For a single core execution of a workflow component, the typical data collected include an event file (.edf), a trace file (.trc) and a profile file (profile.*). The files of independent workflow components are stored in separate directories. We aggregated the profiles and/or traces collected by each component with purpose-built post-processing scripts that generate structured JSON output and merged traces. Aggregating the profile data is somewhat straightforward, as the time dimension is collapsed within each component measurement. However, the trace data includes detailed communication information between processes within the component (i.e. MPI messages

Figure 3: The overview panel (top) and trace detail panel (bottom) of our framework. The overview summarizes the whole workflow execution in a timeline format, where the gray level indicates the density of events (call entry or exit) and the top axis shows message counts among nodes/threads. The trace detail expands the detailed function calls and message communication for the selected nodes. The function calls are color coded according to their functional groups.

from rank *m* to rank *n*). Additional post-processing must be done to remap component ranks within a global, workflow instance rank structure.

Finally, a trace file and profile file both in JSON format were generated and stored into back-end database. To deal with the scale issue to avoid managing large files, instead, separate trace/profile files can be generated according to the unique core ID. In our use case, the size of the single trace file is 16GB, and the profile file is 21MB, both in JSON format.

4.2 The Framework

Considering the analysis tasks, we designed our framework based on the "multilevel" visualization model discussed in (Naser Ezzati-Jivan, 2017) to store and display the hierarchy of events, with a proper navigation and exploration mechanism. In details, we devise a data model including the following information: 1) the overall structural description of the workflow, 2) the metadata about the workflow, 3) the connected trace events of the entire workflow, and 4) the connected profiling of the entire workflow. In order to explore all the above information, we de-

vised and developed a web-based level-of-detail and multiple-panel visualization framework with a front-end plotting the data and a back-end performing necessary aggregation and management.

Our front-end visualization includes four major components: overview, detailed view, node detailed view, and profile view that together establish an interactive analysis platform to visually explore and analyze the performance of a workflow.

4.3 Data Aggregation

Considering the scale of the workflows, different visualization panels must have separate data forms on the back end to reduce the amount of data sent to front end. Therefore, for the overview panel, we design two aggregations: 1) the event aggregation by projecting the events along the timeline into individual time bins, and 2) the message aggregation by projecting the messages to histogram along the timeline.

For a function call in the workflow, it is represented by two seperate trace events of function entry and function exit with the event timestamps. For example, the function of "TAU_init"
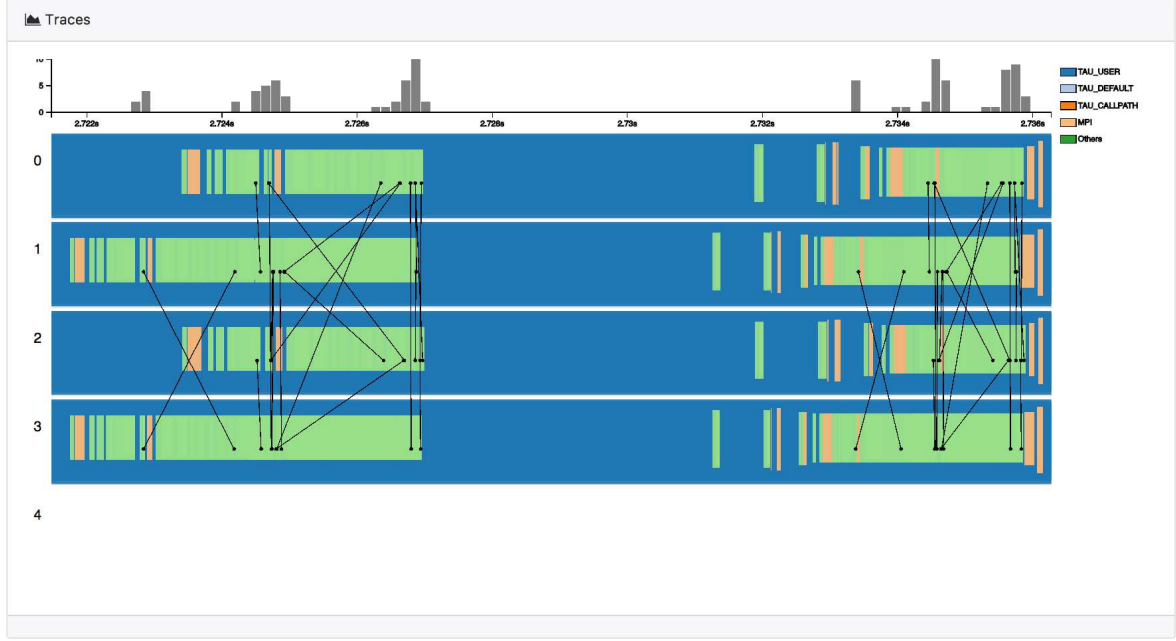
Figure 4: The trace detailed view panel: a coarse level of selected timeline is illustrated. Different transparency can be seen to enhance function separation when there are many small function calls.

has two events: {"name":"TAU_init", "event-type":"entry", "time":"12305", "node-id" 3, "thread-id" 0} and {"name":"TAU_init", "event-type":"exit", "time":"19870594", "node-id" 3, "thread-id" 0}.

Algorithm 1: Matching entry and exit events of functions calls.

Require: $E = \{e\}$ is a list of trace events, $t(e)$ is the timestamp of $e \in E$. F is the result matched functions. D is the list of the callpath depths of F.

1: **function** MATCHING(E)
2:     sort E by the timestamps
3:     s is initialized as an empty stack
4:     $d \leftarrow 0$
5:     **for** $i \leftarrow 1$ to $|E|$ **do**
6:         **if** e_i is an entry event **then**
7:             $d \leftarrow d + 1$
8:             pop e_i into s
9:         **else**
10:             $d \leftarrow d - 1$
11:             pop e' from s
12:             push (e', e_i) to F, push d to D
13:     **return** F,D

Since the same function can be called and exit multiple times in one core, pairing the entry and exit events E into complete functions F is necessary. Furthermore, in these un-matched traces, it is difficult to find the call relationships of different functions. We proposed Algorithm 1 to pair the events and calculate the callpath depths D of the functions F. Our algorithm is similar to a balanced parentheses matching algorithm.

As a result, the trace events E executed in one core are paired into a set of complete function calls F, (e.g., a function call of "TAU_init" is {"name":"TAU_init", "start":12305, "end" 19870594, "node-id" 3, "thread-id" 0}). The paired events are stored and indexed all in a back-end database, where the detailed view panels can query directly.

4.4 Overview

Overview shows the summary of the whole workflow execution as in Fig. 3(top). There are two parts: the trace events, and message counts. The trace events indicating the start and end time of each function call are shown as timelines. We use intensity to indicate the depth of the call path. A darker color represents a more nested function call. For each node/thread, the trace events are plotted separately. Above the timelines, we also visualized the message counts (sent or received) in a separate histogram view along the timeline. For the interaction, it allows the user to select a time range of interest and see more details in the trace detailed view panel.
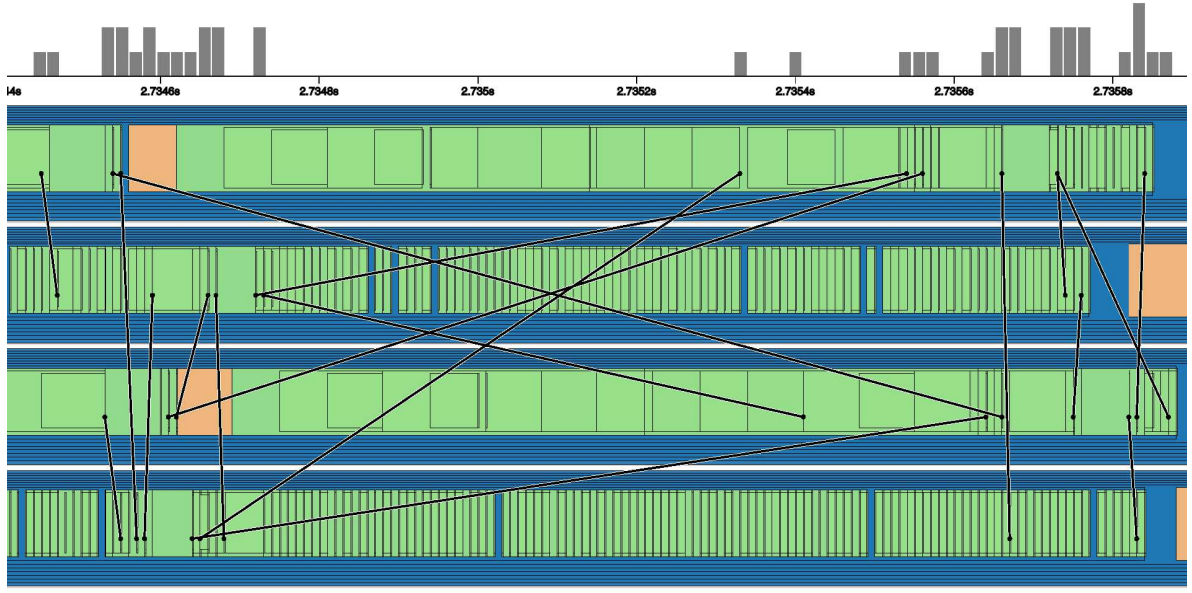
Figure 5: The detailed view panel: a fine level of selected timeline is shown when further zooming in. We enhance the function separation by adding the borderline to each rectangle. The height of rectangle is adjusted according to its call depth.

4.5 Trace Detailed View

The trace detailed view shows the function calls and messages in the selected time range as in Fig. 3(bottom). Each function call is visualized with a rectangle and its color representing the corresponding call group. The functions are visualized with nested rectangles that indicate their depths in the call path. This fashion is similar to what Jumpshot (Jumpshot, 2014) utilized, but we chose a more compact layout. In this view, users can still zoom in to explore more details by selecting a smaller time range. This is shown in Fig. 4.

When there are many short function calls in a nested call structure, it can be difficult to observe small events and differentiate each call. Therefore, we designed three features for that issue:

- First, we use different transparency to enhance the visibility of overlapped functions as can be found in Fig. 4.

- Second, when zooming in, we add the border line of the rectangle to enhance the separation of different functions, as shown in Fig. 5. They are color coded according to different call groups.

- Furthermore, in order to enhance the call path structure, we reduce the height of the rectangle along a call path. Therefore, a callee function must have smaller height than a caller function. As in Fig. 5, where by observing the number of horizontal lines around a function, users can easily detect the depth d of the call.

For the message visualization, additionally, we visualized the message passing (send and receive) between functions as straight black lines (see Fig. 5). As being organized in a timeline, the line direction is ignored since the message is always passing from left to right. Finally, when hovering over each rectangle, the detailed function name can be seen in text.

4.6 Node Detail View

In the detailed view, all functions are plotted as nested rectangles. This design is compact and reflects caller-callee relationship well. However, when there are too many short function calls one after another, it can be difficult to track how often and how long each function is executed. Therefore, we design a node level detail exploration tool as shown in Fig. 6. This view is aligned with detailed view panel when user selects one specific node to explore. Then the nested rectangles are replaced with stacked bar graphs, where overlapping is avoided. In this new view, the vertical order of each bar graph reflects its call path depth. In Fig. 7, a fine level design with highlighted strokes around bar graphs further enhances the separation of functions.

4.7 Profiles View

In the Profiles view, with the selected metric (time or counter), we visualize the percentage spent on each function for each node/thread in stacked bar graphs.

Figure 6: The node detail panel: a coarse level visualization of the selected timeline of node 3 is shown. The trace panel and the node detail panel are aligned along the time axis, which makes it easy to compare.
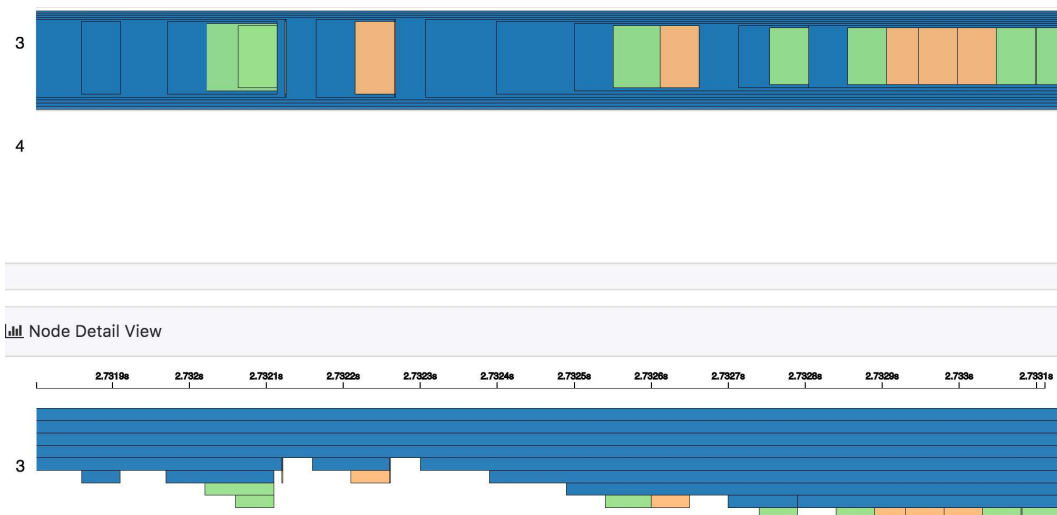

Figure 7: The node detail panel: a fine level visualization of the selected timeline is shown when further zooming in; the node 3 is selected, and the borderlines are plotted.

For example, in Fig. 8, we plotted the profile for "exclusive time" metric.
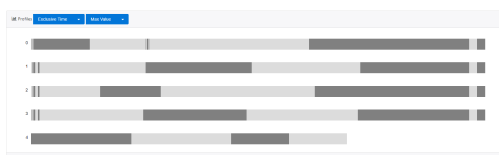

Figure 8: The Profiles view: stacked bar graphs to show profiles of nodes with maximum exclusive time metric.

5 CONCLUSION

In this paper, current progress on capturing and visualizing performance of scientific workflows is presented. We propose improved workflow acquisition, devise a web-based visualization framework integrating both trace and profile into one display. In specific, we present a few data aggregation methods, propose the visualization with different levels of details, provide message communication with both line connections and message count histograms.

As future work, we have made substantial plans: 1) Back end: more data aggregation mechanism to enhance scalability; efficient data query and storage method; 2) Front end: additional levels of detail of visualization to accommodate aggregation designs in overview level and trace detail level; workflow component data flow. We will also work on an extreme-scale or exascale use case, and conduct more case studies.

ACKNOWLEDGEMENTS

REFERENCES

Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., and Tallent, N. R. (2010). Hpctoolkit: Tools for performance analysis of optimized parallel programs. volume 22, pages 685–701. Wiley Online Library.

I. Foster, M. Ainsworth, e. a. (2017). Computing just what you need: Online data analysis and reduction at extreme scales. *Europar*.

Isaacs, K. E., Giménez, A., Jusufi, I., Gamblin, T., Bhatele, A., Schulz, M., Hamann, B., and Bremer, P.-T. (2014).

State of the art of performance visualization. In *Eurographics/IEEE Conference on Visualization State-of-the-Art Reports*, EuroVis.

Jumpshot (2014). Examining trace files with jumpshot.

Karran, B., Trumper, J., and Dollner, J. (2013). Synctrace: Visual thread-interplay analysis. *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, 00:1–10.

Knüpfer, A., Brunst, H., and et al (2008). The vampir performance analysis tool-set. *Tools for High Performance Computing*, pages 139–155.

Knüpfer, A., Rössel, C., Mey, D. a., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., and Wolf, F. (2012). *Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir*, pages 79–91. Springer Berlin Heidelberg, Berlin, Heidelberg.

Naser Ezzati-Jivan, M. R. D. (2017). Multi-scale navigation of large trace data: A survey. *Concurrency and Computation: Practice and Experience*, 29.

NWChemEX (2016). A new generation of nwchem, a high-performance computational chemistry software package.

ParaProf (2014). Profile analysis with paraprof.

Shende, S. S. and Malony, A. D. (2006). The tau parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2):287–311.