Performance Understanding and Analysis for Exascale Data Management Workflows

Final Report (September 1, 2014 – August 31, 2018)

DOE Award: SN10019 DE-SC0012381

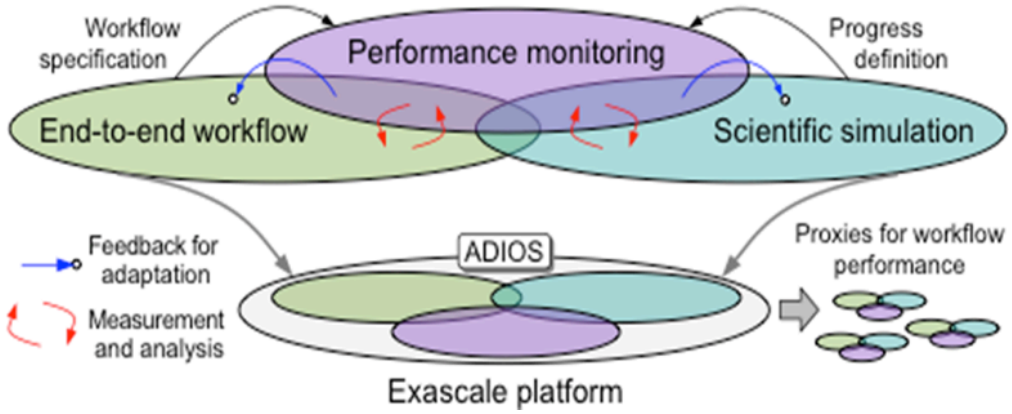
Allen D. Malony
Department Computer and Information Science, University Oregon, Eugene, Oregon, 97403

Project Goals

The goal of the Exascale Data Management Workflows project is to create tools to support techniques for in-situ data analysis and data management. Online data management co-running with simulations accelerates the scientific processes being carried out, provides rapid and timely scientific insights, and can help avoid unnecessary and scientifically invalid simulation computations, particularly when combined with or able to utilize data from experimental instrument and/or from past simulation runs. The University of Oregon team is contributing to the performance instrumentation and analysis aspect of this project. Our focus has been on efficient performance monitoring of data management at scale.

The general approach of the MONA project is depicted in Figure 1 below. The figure shows performance monitoring applied to an I/O workflow associated with a scientific simulation, with online measurements captured from this workflow informing methods for workflow reconfiguration and dynamic adjustment. The goal is to maintain suitable levels of Quality of Service for workflow execution, by understanding the underlying causes of workflow performance and behavior. One outcome is workflow performance models able to characterize realistic workflows. Another outcome is performance 'mini-apps' implementing such workflow behavior. Out of scope for this project are advanced methods for online workflow control.

Figure 1: MONA approach.



1 Summary of Accomplishments

Over the course of the project, our accomplishments included instrumentation of the ADIOS data management workflow infrastructure, with an initial focus on API instrumentation, followed by instrumenting key internal elements of a select ADIOS transport to capture data movement details. We also instrumented the LAMMPS [9] and GTS [12] materials modeling code with TAU to measure their performance. We created a new lightweight workflow monitoring infrastructure, WOWMON (Workflow Monitor) which enables user's access not only to the cross-application performance data such as end-to-end latency and execution time of individual components at runtime, but also to customized performance events. Through the study of real scientific workflows-(e.g., LAMMPS and GTS) with the help of WOWMON [17], we identified their key functions which contribute most to the end-to-end latency of the workflows.

In implementing the WOWMON infrastructure, we made the following contributions:

- We designed a simple interface for users to access in-memory performance data across multiple applications/components of workflows, and implemented a flexible and lightweight runtime system, supporting data sampling, multiple network topologies, metric selection for customized coverage of application-level and system-level metrics.
- We implemented WOWMON based on TAU [11] and EVpath [4] libraries. With the help of WOWMON our experiments using realistic scientific workflows-(e.g., *LAMMPS* and *GTS*) show that the end-to-end latency of workflows can be affected by memory buffer size for in-situ operations, policy of mapping processes of applications to nodes, and hardware configurations.
- We demonstrated how to evaluate the metrics using a feature selection algorithm from the machine learning software Weka and then use its output to further reduce the volume of performance data by 54% for *LAMMPS* workflow, while still providing in-depth insights for end-users.

Over time, it became clear that because of some naive design decisions relating to architecture and incompatibility with batch systems on key DOE supercomputers, the WOWMON infrastructure was not going to be sufficient to meet the needs of the project. For that reason, we abandoned the WOWMON prototype and began work on the Scalable Observation System (SOS). In addition, other accomplishments occurred during this time:

- The MONA project includes aspects of workflow modeling and evaluation for purposes of studying monalytics capabilities and services in different scenarios. A synthetic workflow framework was created to design workflow models and to generate synthetic workflow instances that will run on a cluster. The models capture workflow components and their inter-operation. The synthetic workflow instances derived from the models are parameterized to control rate and volume of inter-component interactions. A synthetic workflow program will be able to be configured with WOWMON services for purposes of testing, workflow characterization, and evaluation on different HPC platforms.
- A significant effort was placed on the design and development of a new observation system called (SOS) [15] that became the foundation for performance monitoring for the duration of the project.
- Utilizing the LAMMPS workflow as a target, we have created scalable 'workstation' entities that use monitoring and adapt their degree of parallelism based on global throughput feedback [3].
- We have also demonstrated low-level tight-feedback scheduling of analysis tasks on the GPU based on monitoring of 'gaps' in GPU utilization. This was work with the LAMMPS and PicOnGPU workflows and it enabled significantly more efficient utilization of the GPU by intermingling 'analysis' tasks on the GPU with the primary computations [5].

The above tasks were successfully completed during year 2. Year two of MONA was disrupted significantly by the illness and death of originating PI Karsten Schwan, as well as other departures and changes of key personnel. However, the implementation of SOS proved to be a success, and integrated many aspects of the project. As a result, we published several papers on the infrastructure and its use in the project.

- SOS was developed and was tested with synthetic workflows and real workflow case studies on large HPC clusters, scaling to thousands of MPI processes running on hundreds of nodes of Jaguar at ORNL. SOS was refactored as a TAU plugin [8], to enable greater flexibility with development and deployment.
- The EVPath transport (developed by our partners at Georgia Tech) was integrated with SOS to enable more dynamic on-the-fly processing of monitoring data. The EVPath transport became the default and best supported transport used in SOS.
- The ADIOS library was modified with permanent, always-on instrumentation. This instrumentation was added without including the TAU software as a build dependency, and allowed for link-time or run-time resolution of the instrumentation symbols. This instrumentation was designed as a callback interface, allowing measurement libraries such as TAU to implement the callbacks and measure the library. This instrumentation was a pre-requisite for the ability to extract ADIOS communication patterns within simulations and generate synthetic benchmarks for study [7].
- SOS was used in large studies with Fusion simulation codes. Three different scenarios were explored: large scale performance monitoring of coupled applications [6, 2], generation of synthetic ADIOS benchmarks in order to study I/O characteristics [7, 13], and exploring node/rank placement of coupled MPI applications in order to improve inter-application communication efficiency and reduce execution time. SOS was also used to demonstrate the ability to visualize performance data in the simulation domain during runtime [14].

We next present additional technical detail about the work conducted during the project.

1.1 ADIOS Instrumentation using TAU

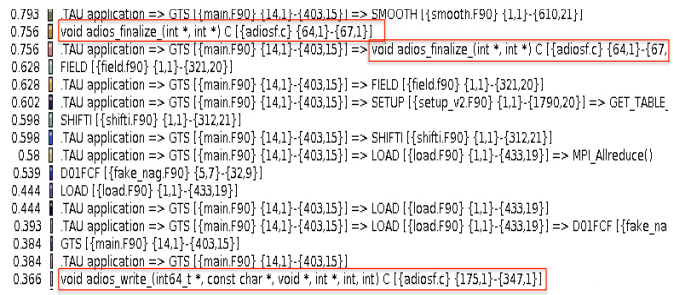


Figure 2: An example of TAU profile for GTS application

We instrumented ADIOS [1] library using TAU [11] for source instrumentation with a focus on data transfer between upstream and downstream applications. To reduce profiling overhead we selectively instrumented six files in ADIOS library, including adios.c, adios_read.c, adiosf.c, adiosf_read.c, adiosf_write_mod.f90, and adios_read_mod.f90. Figure 2 shows a profile generated when GTS [12] was run with the instrumented library.

In order to provide a more permanent, always-on instrumentation solution, we eventually added an instrumentation callback interface to the ADIOS 1.13 library. This callback API allows any performance tool to measure the ADIOS library by implementing a set of callback functions, and registering for events at program startup. The TAU performance library added this support, and we now have instrumentation callbacks for approximately 70 ADIOS external API and internal functions. This callback interface was instrumental in the ability to extract MPI and ADIOS behavior patterns for the Skel tool [7, 13].

1.2 Lightweight Instrumentation of Scientific Workflows

We implemented WOWMON, which uses lightweight instrumentation to monitor end-to-end latency of workflows and provided knobs to control the volume of collected performance data at runtime if needed. In its design we provided simple user interface (APIs) for end users to create and track timers, specify system and

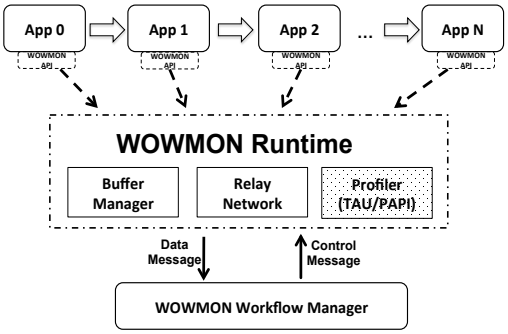


Figure 3: WOWMON architecture

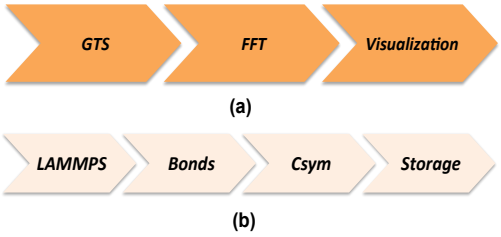


Figure 4: GTS and LAMMPS workflows with in-situ analytics

application events of interests, and customize communication patterns. Moreover, at runtime WOWMON supports efficient buffer management, portable library to access profile data in memory, and thin and flexible communication layer.

1.3 WOWMON Software Architecture

The WOWMON infrastructure includes three major components, *user interface (APIs)*, *runtime library*, and *workflow manager*. Programmers need to instrument source codes using the APIs. At runtime, WOWMON creates relay network connecting processes of applications and workflow manager using a network topology as specified in parameter (*WOWMON_TOPO_COMM*) by users (*star* topology is set by default). Then software and hardware events can be added to an initial metric set, whose members can be dynamically selected at runtime to control monitoring overhead. Specifically, software events count program performance such as function execution time, call depths, and memory heap size when entering and exiting a function. And hardware events are used to count hardware performance such as cache misses, floating point operations, and so on. WOWMON reads values of the events from in-memory data structure of profilers, such as TAU [11]. The data structure tracks all the current value of metrics in the set.

1.4 The Design of Metric Set for Representative Scientific Workflows

Metric Name	Function Description
bonds_read_input	Read data and give us a handle on throughput.
bonds_list_output	Output data and give us a handle on throughput out of bonds.
bonds_compute_send	Main computation function in *Bonds*. Most of time is spent in building the adjacent format output.
bonds_read_input_mem	Memory usage for executing bonds_read_input()
bonds_compute_send_mem	Memory usage for executing bonds_compute_send()
csym_read_input	Read data and give us a handle on throughput.
csym_compute_send	Main computation function in *Csym*. Most of time is spent in converting input data.
csym_output_results	Following function csym_output_results() is the end of workflow and where we end latency measurement.
csym_read_input_mem	Memory usage for executing csym_read_input_mem()
csym_compute_send_mem	Memory usage for executing csym_compute_send_mem()
lammps_start_timer	The timer is triggered when generated data is placed in buffer on LAMMPS end.
csym_stop_timer	The timer is triggered when the last analytic finishes.

Table 1: Metrics for LAMMPS Workflow

Metric Name	Function Description
gts_restart_write	Output data and give us a handle on throughput out of GTS.
gts_restart_write_mem	Memory usage of executing restart_write()
fft_phi	Main computation function in *FFT*. Most of time is spent on Fourier calculation.
fft_phi_mem	Memory usage for executing fft_phi()
gts_start_timer	The timer is triggered when checkpointing data is placed in networking buffer on GTS end.
fft_stop_timer	The timer is triggered when FFT calculation finishes.

Table 2: Metrics for GTS Workflow

To use WOWMON as a performance monitoring infrastructure, the initial metric set should be well designed in order to characterize data-driven behaviors of workflows. It requires a good understanding of both simulation codes and in-situ analytics. This involves non-trivial work and needs collaboration of domain scientists and programmers. In our workflows project, we used an approach combining both knowledge of domain scientists and hints generated from analyzing offline profiles. More specifically, for each application in the workflow, we instrumented it using TAU's compiler instrumentation function. Then we have a preliminary run for those instrumented functions without in-memory in-situ execution. After that the execution times and memory demands for functions in each application were ranked for inspection. We then made final selection after discussing with domain scientists on the metrics for on-line monitoring using WOWMON. However, these selected metrics may not be equally important with respect to correlation with workflow performance such as the end-to-end latency.

LAMMPS Metric Set: LAMMPS (Large Scale Atomic/Molecular Massively Parallel Simulator) [9] has been widely used for material science study. For a use case presented in Figure 4(b), the *Bonds* program performs all-nearest neighbor calculation to determine which atoms are bonds together. *Csym* then uses the output of *Bonds* to further determine whether there is a deformation in the material. If deformation is detected, *Csym* continues to calculate the conditions under which a crack occurred. For the LAMMPS workflow we list the key metrics related to different functions, which are either compute-intensive or memory-intensive or both. As shown in Table 1, functions such as *bonds_read_input()* and *csym_output_results()* are related to I/Os for moving data from upstream applications to downstream analytics or to disks. Functions such as *bonds_compute_send()* and *csym_compute_send()* are compute kernels.

GTS Metric Set: GTS (Gyrokinetic Tokamak Simulation) [12] is a global three dimensional Particle-In-Cell code which helps understand complex phenomena involving the coupling between the charged particles making up the plasma and the sea of waves generated by their collective interactions. GTS simulation outputs datasets for various purposes-(e.g., checkpointing, diagnostics, and visualization). For GTS workflow six key metrics are identified. On the GTS side, we are interested in when the checkpointing data is dumped to memory buffer. On the FFT side, memory usage and function execution time are monitored for this compute kernel. Metrics of the GTS workflow are listed in Table 2.

1.5 WOWMON

The WOWMON (WOrkfloW MONitor) prototype developed in Year 1 successfully demonstrated the ability to access and aggregate information about multiple applications/components of workflows and their execution. The research results presented in the ICCS paper (Zhang, 2016) highlight the tracking workflow performance behavior at runtime, both at the level of individual components and for aspects that require a workflow-wide perspective. WOWMON included an interface for capturing component-level and system-level performance data at runtime using the TAU Performance System, and a workflow manager that collected aggregated data and allowed selection of online metrics for analysis. WOWMON was applied to the study of end-to-end latency of LAMMPS and GTS scientific workflows, exposing workflow-dependent sensitivities to memory buffer size for in-situ execution, policy of mapping processes of applications to nodes, and hardware configurations.

The WOWMON prototype was intended more as a proof of concept than a robust workflow monitoring platform. WOWMON kept to a simple functional design and the implementation was specific to the requirements of the two case studies. The WOWMON API for component instrumentation was limited and allowed only TAU performance data capture. Also, the component aggregation was based on a many-to-1 monitoring

model used in TAU in the past. While it was possible to collect metrics from the LAMMPS and GTS workflow components, showing how insight into dynamic of workflow execution is possible, the observation approach was ad hoc. The consequence is that it becomes more difficult to create monalytics solutions that gather relevant data from multiple layers to aid in explaining workflow performance behavior. Although WOWMON could capture metadata in the individual component observations that allowed end-to-end latency analysis, it was difficult to determine causes of latency dynamics. This would require a more flexible means to observe events at different workflow layers and from different sources.

In the course of working with the WOWMON prototype, it became apparent that a general workflow monalytics architecture required a more flexible and configurable observation system. This motivated re-targeting of our efforts in Year 2 toward the Scalable Observation System (SOS).

1.6 SOS

The Scalable Observation System (SOS) was conceived as a fully realized implementation of the concepts in the MONA project proposal, but with a broader application than just for monitoring workflows. The reference implementation of SOS can be found in publications and online under the name "SOSflow".

The SOS design emphasizes a semantic data model with distributed information management and structured query and access. A dynamic database architecture is used in SOS to support aggregation of streaming observations from multiple sources. SOS provides interfaces for sources of information to encode data, metadata, and semantic context. Interfaces are also provided for in situ analytics to acquire information and send back results for application actuators. SOS launches with the application, runs along side it, and can acquire its own resources for scalable data collection and processing. The primary objectives of SOS are flexibility, scalability, and programmability.

SOS is implemented as a collection of daemons and services, designed to be either launched in concert with distributed applications and workflows or launched at the boot time of a computational node. Each computational node participating in SOS launches the sosd daemon that listens to a specified socket on the node, and one or more sosd back-end databases are launched on extra compute nodes in order to aggregate data across the participating nodes in the allocation. Because HPC systems frequently have highly constrained network infrastructure, the communication method between the daemons is was originally implemented using MPI. Over time this communication method was complemented with an implementation supporting EVPath, which has become the default communication technique.

Applications that wish to send data to the SOS daemons simply link with the SOS client library, and publish data using a simple API. Essentially, the SOS infrastructure executes as a parallel distributed service overlay within an HPC allocation. Applications connect to the SOS service over TCP/IP sockets on-node.

Figure 1 shows how the collection of SOS processes and application processes co-exist in a given allocation. The client API includes, but is not limited to the functions listed in Table 3.

Table 3: Key SOS client API functions. These functions are used by applications or libraries that wish to send data to the SOS daemons.

`SOS_init`	initialize the SOS client library, connect to the local daemon
`SOS_finalize`	disconnect from the local daemon, shut down the SOS client library
`SOS_pack`	pack a new name / value pair
`SOS_announce`	alert the SOS client library that there are new (not updated) packed values
`SOS_publish`	send the announced and packed values to to the SOS daemon

Applications can send data directly to SOS using the client API. In addition, libraries (such as ADIOS, MPI, etc.) can send data to SOS. In fact, we have modified the TAU measurement system to asynchronously and periodically send the full TAU profile over SOS.

Once data is sent to the SOS daemons, that data is stored in a lightweight SQL database, and by default is re-queued to be transmitted to the central SOS database(s). This behavior is highly configurable: Databases can be completely disabled, configured to run in memory-only to avoid filesystem contention, and can be set

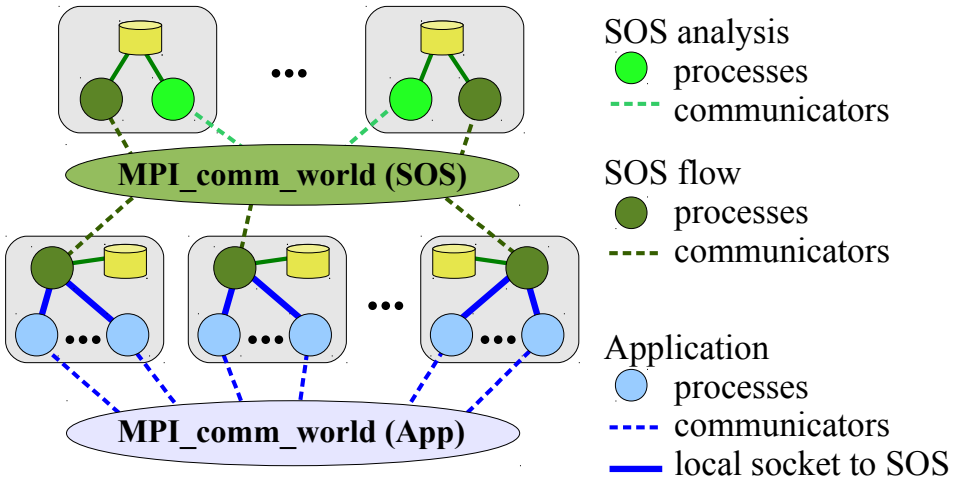


Figure 5: The SOS infrastructure exists as a parallel distributed service overlay. Applications transparently connect to the SOS service via its API, which internally communicates with the SOS daemon using TCP/IP sockets. SOS processes queries that are sent to it by analytics tools, allowing online data-driven workflow adaptability.

to export to a file only at the end of a run when the SOS daemons are being shut down. To handle cases of extremely high volumes of data, SOS's data processing options were extended to include a highly-efficient circular cache of values. The value vache is able to completely avoid filesystem impact while guaranteeing a specific and fixed memory footprint and query resolution speed, though requests made of values from the cache lack the expressive power of full SQL support. Both the cache and database can be enabled simultaneously, to allow for rapid data ingestion/interrogation, in addition to providing long-term SQL data storage and support for sophisticated queries.

SOS supports monalytics by also providing an analytics and query functionality over the same infrastructure. Clients request data from SOS daemons, and receive replies asynchronously in a message handling callback function. This allows clients to continue working while their data request is being processed. Clients are also able to register their "sensitivity" to a named channel, and send trigger messages with payloads to named channels. The SOS daemon ensures that all trigger events are delivered to clients that have expressed an interest in that channel, delivering it to the same callback routine within the client application. For the purposes of MONA, these trigger messages include workflow or application reconfiguration commands.

1.7 Synthetic Workflows

In the MONA proposal, we described the need for executable performance models and workflow mini-apps. In particular, we described the need to generate synthetic workflows of "applications" that would simulate real-world application behavior without the complexity of configuring, building and executing actual scientific workflows. In addition, as we developed the new SOS infrastructure it became clear that we needed a flexible test harness in order to explore different workflow configurations. To that end we have implemented a prototype synthetic workflow generator that can generate arbitrary graphs of pipelined application workflows. The generator itself is a Python script that takes a JSON configuration file as input. The script generates ADIOS XML files and PBS batch scripts for launching a generic "application" executable that can act as a producer, consumer, or producer/consumer of ADIOS (over Flexpath) data. The executable takes arguments that describe its name, the names of its children, the names of its parents and the number of simulated iterations to execute. Based on the input arguments to the executable, it is configured as a producer and/or consumer of data, and knows which other executable(s) it will communicate with over ADIOS. An example script is shown in Figure 6, along with a visual representation of the workflow graph described by the example.

Currently the synthetic workflow generator only creates static workflows with a fixed computation /

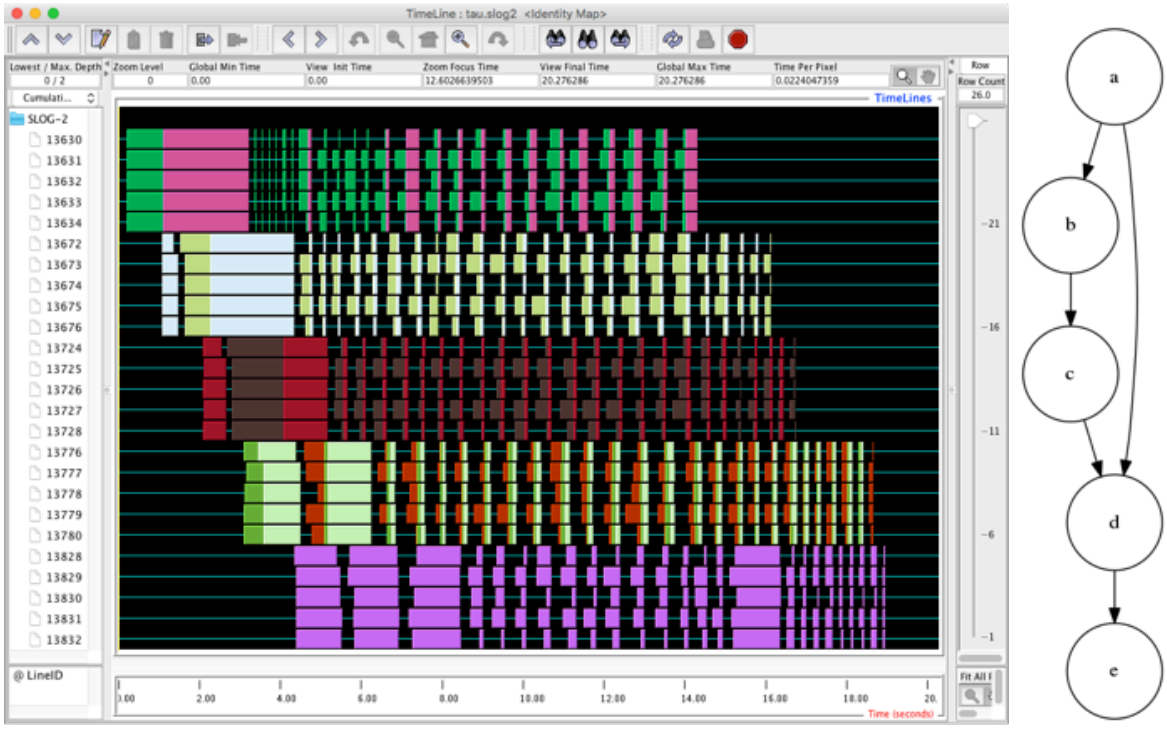
(a) Example workflow JSON specification.

(b) Resulting workflow graph.

Figure 6: Synthetic workflow example generated for evaluating SOS, and resulting workflow graph.

communication ratio and fixed communication between workflow components. In future work, we will add the ability to specify the duration and intensity of computation, the volume and frequency of communication over both MPI and ADIOS, as well as specify which ranks and communicators participate in each exchange between applications. Figure 3 shows a TAU performance trace of a similar workflow configuration in the Jumpshot trace viewer, using 5 MPI ranks for each of the applications in the workflow. In the trace viewer, the x-axis is the timeline and the y-axis represents each process for each of the 5 simulated applications in the workflow. For this example, the top five lines represent the MPI processes associated with application "a", the second five lines represent the MPI processes associated with application "b", and so on. Figure 4 shows the Jumpshot legend for this trace, indicating which events correspond to each color in the trace.

Eventually, it became clear that the best way to generate synthetic benchmarks with realistic behavior was to capture the behavior of actual applications, and use that event capture to generate synthetic benchmarks. Thanks to the ADIOS callback API described in Section 1.1 and the MPI wrappers in TAU, we were able to construct a TAU plugin such that MPI and ADIOS events would be aggregated over SOS during an application execution. The resulting SOS database is then post-processed with a set of python scripts that will extract out the ordered events and generate a YAML output that can then be used to generate an application simulation that has the same event ordering as the original application. The intention with this workflow was to identify periods of intense MPI activity in the application, and schedule ADIOS data transfers to occur during times when the MPI activity is low. In another study, the MPI and ADIOS events were used to understand potential constraints with respect to network communication and process placement on the Jaguar system at ORNL. Both of these studies were carried out by our colleagues at ORNL. These



(a) Five synthetic applications (A,B,C,D,E) integrated with ADIOS and captured with a TAU trace.

(b) Workflow graph.

Figure 7: TAU trace of a similar synthetic application example, using 5 MPI ranks per simulated application.

synthetic benchmarks and the MPI process placement research are further described in the publications [7] and [13].

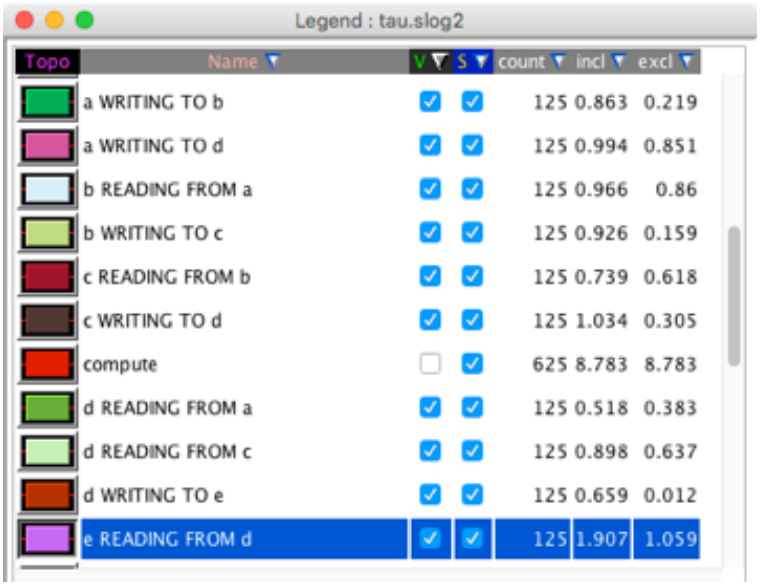


Figure 8: Jumpshot legend, showing which TAU trace events corresponds with each color in the trace viewer.

2 Publications and other Intellectual Outcomes

- Publication: Xuechen Zhang, Hasan Abbasi, Kevin Huck, and Allen D. Malony. "Wowmon: A machine learning-based profiler for self-adaptive instrumentation of scientific workflows." Procedia Computer Science 80 (2016): 1507-1518.
- Publication: Chad Wood, Sudhanshu Sane, Daniel Ellsworth, Alfredo Gimenez, Kevin Huck, Todd Gamblin, and Allen Malony. "A scalable observation system for introspection and in situ analytics." In 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT), pp. 42-49. IEEE, 2016.
- Publication: Jeremy Logan, Jong Youl Choi, Matthew Wolf, George Ostrouchov, Lipeng Wan, Norbert Podhorszki, William Godoy, Scott Klasky and Erich Lohrmann and Greg Eisenhauer and Chad Wood and Kevin Huck. "Extending Skel to support the development and optimization of next generation I/O systems." In 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp. 563-571. IEEE, 2017.
- Publication: Chad Wood, Matthew Larsen, Alfredo Gimenez, Kevin Huck, Cyrus Harrison, Todd Gamblin, and Allen Malony. "Projecting performance data over simulation geometry using SOSflow and ALPINE." In Programming and Performance Visualization Tools, pp. 201-218. Springer, Cham, 2017.
- Publication: Mark Kim, James Kress, Jong Choi, Norbert Podhorszki, Scott Klasky, Matthew Wolf, Kshitij Mehta, Kevin Huck, Berk Geveci, Sujin Phillip, Robert Maynard, Hanqi Guo, Tom Peterka, Kenneth Moreland, Choong-Seock Chang, Julien Dominski, Michael Churchill, David Pugmire. "In Situ Analysis and Visualization of Fusion Simulations: Lessons Learned." In International Conference on High Performance Computing, pp. 230-242. Springer, Cham, 2018.
- Publication: Jong Youl Choi, Choong-Seock Chang, Julien Dominski, Scott Klasky, Gabriele Merlo, Eric Suchyta, Mark Ainsworth, Bryce Allen, Franck Cappello, Michael Churchill, Philip Davis, Sheng Di, Greg Eisenhauer, Stephane Ethier, Ian Foster, Berk Geveci, Hanqi Guo, Kevin Huck, Frank Jenko, Mark Kim, James Kress, Seung-Hoe Ku, Qing Liu, Jeremy Logan, Allen Malony, Kshitij Mehta, Kenneth Moreland, Todd Munson, Manish Parashar, Tom Peterka, Norbert Podhorszki, Dave Pugmire, Ozan Tugluk, Ruonan Wang, Ben Whitney, Matthew Wolf, Chad Wood. "Coupling exascale multiphysics applications: Methods and lessons learned." In 2018 IEEE 14th International Conference on e-Science (e-Science), pp. 442-452. IEEE, 2018.
- Publication: Allen D Malony, Srinivasan Ramesh, Kevin Huck, Nicholas Chaimov, Sameer Shende. "A Plugin Architecture for the TAU Performance System." In Proceedings of the 48th International Conference on Parallel Processing, 2019 *(to appear)*.
- Publication: Matthew Wolf, Jong Choi, Greg Eisenhauer, Stéphane Ethier, Kevin Huck, Scott Klasky, Jeremy Logan, Allen Malony, Chad Wood, Julien Dominski, Gabriele Merlo. "Scalable Performance Awareness for In Situ Scientific Applications", eScience 15th International Conference, 2019 *(to appear)*.
- Poster: Chad Wood. "SOSflow: A Scalable Observation System for Introspection and In Situ Analytics", ICPP 2018. Summary: *Efficiently observing and interacting with complex scientific workflows at scale presents unique challenges. SOSflow helps meet them. This work presents the model and a design overview of the SOSflow In Situ Scalable Observation System, and shares some recent results oriented towards performance understanding of complex workflows at scale.* `http://oaciss.uoregon.edu/icpp18/publications/pos127s2-file2.pdf`
- Software: SOSflow software for performance monitoring of scientific applications and workflows. `https://github.com/cdwdirect/sos_flow`.
- Software: SOSflow synthetic workflow generator. `https://github.com/khuck/sos_flow_experiments`.
- Software: TAU Performance System. `https://tau.uoregon.edu`.

3 Impact

The project "Performance Understanding and Analysis for Exascale Data Management Workflows" has had a number of impacts in the area of performance measurement and monitoring of coupled application workflows. First, SOS is the first runtime monitoring system available in user space for aggregating and monitoring performance data from multiple coupled appliations in a holistic view. SOS has enabled the ability to monitor scientific applications at scale, and led to a number of publications as described in this report. In particular, SOS was used to monitor a coupled fusion application running on thousands of processes on the Jaguar supercomputer. This will be further developed to provide a "dashboard"-like application for monitoring simulations. Because compute time is a valuable commodity, users will have the ability to monitor large scale, long-running coupled simulations with the confidence that they are using the system efficiently and not wasting compute cycles on jobs that may be running in a less-than-efficient manner. This should have an impact on all scientific disciplines that would like the ability to monitor large simulations. Several of our graduate students have participated in developing technology for the project, and have authored or co-authored papers and posters related to the project. The technology developed during the project has subsequently inspired and enabled other DOE funded projects within SciDAC and the ECP project, such as using SOS for autotuning Raja applications (publication coming soon), an enhanced instrumentation interface for the redesigned ADIOS2 library, and led to new and exciting opportunities in the area of runtime performance analysis [16, 10]. The efficient use of HPC resources leads to the reduced consumption of valuable natural resources, and enables more comptuational simulation throughput on equally valuable and limited HPC systems.

References

[1] Adaptable IO System (ADIOS). `https://www.olcf.ornl.gov/center-projects/adios/`.

[2] Jong Youl Choi, Choong-Seock Chang, Julien Dominski, Scott Klasky, Gabriele Merlo, Eric Suchyta, Mark Ainsworth, Bryce Allen, Franck Cappello, Michael Churchill, Philip Davis, Sheng Di, Greg Eisenhauer, Stephane Ethier, Ian Foster, Berk Geveci, Hanqi Guo, Kevin Huck, Frank Jenko, Mark Kim, James Kress, Seung-Hoe Ku, Qing Liu, Jeremy Logan, Allen Malony, Kshitij Mehta, Kenneth Moreland, Todd Munson, Manish Parashar, Tom Peterka, Norbert Podhorszki, Dave Pugmire, Ozan Tugluk, Ruonan Wang, Ben Whitney, Matthew Wolf, and Chad Wood. Coupling exascale multiphysics applications: Methods and lessons learned. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 442–452. IEEE.

[3] Jai Dayal, Jay Lofstead, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Hasan Abbasi, and Scott Klasky. Soda: Science-driven orchestration of data analytics. In *2015 IEEE 11th International Conference on e-Science*, pages 475–484. IEEE, 2015.

[4] Greg Eisenhauer, Matthew Wolf, Hasan Abbasi, and Karsten Schwan. Event-based systems: Opportunities and challenges at exascale. DEBS '09, pages 2:1–2:10. ACM, 2009.

[5] Anshuman Goswami, Jeffrey Young, Karsten Schwan, Naila Farooqui, Ada Gavrilovska, Matthew Wolf, and Greg Eisenhauer. Gpushare: Fair-sharing middleware for gpu clouds. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1769–1776. IEEE, 2016.

[6] Mark Kim, James Kress, Jong Choi, Norbert Podhorszki, Scott Klasky, Matthew Wolf, Kshitij Mehta, Kevin Huck, Berk Geveci, Sujin Phillip, Robert Maynard, Hanqi Guo, Tom Peterka, Kenneth Moreland, Choong-Seock Chang, Julien Dominski, Michael Churchill, and David Pugmire. In situ analysis and visualization of fusion simulations: Lessons learned. In *International Conference on High Performance Computing*, pages 230–242. Springer, 2018.

[7] Jeremy Logan, Jong Youl Choi, Matthew Wolf, George Ostrouchov, Lipeng Wan, Norbert Podhorszki, William Godoy, Scott Klasky, Erich Lohrmann, Greg Eisenhauer, Kevin Huck, and Chad Wood. Extending skel to support the development and optimization of next generation i/o systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 563–571. IEEE, 2017.

[8] Allen D. Malony, Srinivasan Ramesh, Kevin Huck, Nicholas Chaimov, and Sameer Shende. A plugin architecture for the tau performance system. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP 2019, pages 90:1–90:11, New York, NY, USA, 2019. ACM.

[9] S. Plimpton, R. Pollock, and M. Stevens. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In *Proc of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 2007.

[10] L. Pouchard, K. Huck, G. Matyasfalvi, D. Tao, L. Tang, H. V. Dam, and S. Yoo. Prescriptive provenance for streaming analysis of workflows at scale. In *2018 New York Scientific Data Summit (NYSDS)*, pages 1–6, Aug 2018.

[11] S. Shende and A. D. Malony. TAU: The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, pages 287–311, 2006.

[12] W. X. Wang, Z. Lin, W. M. Tang, W. W. Lee, S. Ethier, J. L. V. Lewandowski, G. Rewoldt, T. S. Hahm, and J. Manickam. Gyro-Kinetic simulation of global turbulent transport properties in tokamak experiments. volume 13, page 092505, 2006.

[13] Matthew Wolf, Jong Choi, Greg Eisenhauer, Kevin Huck Stéphane Ethier, Scott Klasky, Jeremy Logan, Allen Malony, Chad Wood, Julien Dominski, and Gabriele Merlo. Scalable performance awareness for in situ scientific applications. In *2019 IEEE 15th International Conference on e-Science (e-Science)*. IEEE, 2019. to appear.

[14] Chad Wood, Matthew Larsen, Alfredo Gimenez, Kevin Huck, Cyrus Harrison, Todd Gamblin, and Allen Malony. Projecting performance data over simulation geometry using sosflow and alpine. In *Programming and Performance Visualization Tools*, pages 201–218. Springer, 2017.

[15] Chad Wood, Sudhanshu Sane, Daniel Ellsworth, Alfredo Gimenez, Kevin Huck, Todd Gamblin, and Allen Malony. A scalable observation system for introspection and in situ analytics. In *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, pages 42–49. IEEE, 2016.

[16] Cong Xie, Wei Xu, Sungsoo Ha, Kevin A Huck, Sameer Shende, Hubertus Van Dam, Kerstin Kleese Van Dam, and Klaus Mueller. Performance visualization for tau instrumented scientific workflows. In *VISIGRAPP (3: IVAPP)*, pages 333–340, 2018.

[17] Xuechen Zhang, Hasan Abbasi, Kevin Huck, and Allen D Malony. Wowmon: A machine learning-based profiler for self-adaptive instrumentation of scientific workflows. *Procedia Computer Science*, 80:1507–1518, 2016.