Scalable Performance Awareness for In Situ Scientific Applications

Matthew Wolf*, Jong Choi*, Greg Eisenhauer‡, Stéphane Ethier¶, Kevin Huck§, Scott Klasky*†‡, Jeremy Logan*, Allen Malony§, Chad Wood§, Julien Dominski¶, and Gabriele Merlo‖
*Oak Ridge National Laboratory, Oak Ridge, TN, USA
‡School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA
¶Princeton Plasma Physics Laboratory, Princeton, NJ, USA
†The University of Tennessee, Knoxville TN, USA
§University of Oregon, Eugene, OR, USA
‖University of Texas, Austin, TX, USA

Abstract—**Part of the promise of exascale computing and the next generation of scientific simulation codes is the ability to bring together time and spatial scales that have traditionally been treated separately. This enables creating complex coupled simulations and in situ analysis pipelines, encompassing such things as "whole device" fusion models or the simulation of cities from sewers to rooftops. Unfortunately, the HPC analysis tools that have been built up over the preceding decades are ill suited to the debugging and performance analysis of such computational ensembles. In this paper, we present a new vision for performance measurement and understanding of HPC codes, *Mon*itoring *A*nalytics (MONA). MONA is designed to be a flexible, high performance monitoring infrastructure that can perform monitoring analysis in place or in transit by embedding analytics and characterization directly into the data stream, without relying upon delivering all monitoring information to a central database for post-processing. It addresses the trade-offs between the prohibitively expensive capture of all performance characteristics and not capturing enough to detect the features of interest. We demonstrate several uses of MONA; capturing and indexing multi-executable performance profiles to enable later processing, extraction of performance primitives to enable the generation of customizable benchmarks and performance skeletons, and extracting communication and application behaviors to enable better control and placement for the current and future runs of the science ensemble. Relevant performance information based on a system for MONA built from ADIOS and SOSflow technologies is provided for DOE science applications and leadership machines.**

I. Introduction

As we look towards the future of high performance computing, it is becoming evident that the changes are going to be more than just novel hardware and programming languages. Part of the promise of exascale computing and the new generation of scientific simulation codes is the ability to bring together time and spatial scales that have traditionally been treated separately – the design of materials from the atomic level out to the helicopter rotor, the simulation of cities from the flow of air pollution to the flexing of buildings, or understanding a whole magnetic confinement fusion device experiment from the ions at the core of the plasma to the magnetic fields and casing materials at the edge. These ambitious projects put new constraints on the tools, middleware, runtimes, and even the software engineering of these codes.

In particular, we have seen a change from the construction of single, monolithic codes to computational experiments composed of multiple executables, run as ensembles, coupled codes, or as pipelines of in situ computations. This means that the tools that have been built up over the last decades for performance understanding and management of large MPI-based codes need to be extended to be able to deal with the new complexity. As a simple example, many performance analysis tools index based on the MPI rank ID, but in an ensemble where there are many concurrently running MPI domains, there is no enforced unique ID from the runtime. Solving that is relatively straightforward by itself, but you then add to it many additional constraints and concerns as a result of dealing with the measurement of collections of distinct runtimes and applications, and it becomes more than what a simple patchwork of fixes can address.

To address this change in the construction of high performance codes, we have sought to infuse some of the innovations from the cloud/enterprise computing world into the high performance computing (HPC) space. In data center scale monitoring of modern virtualized environments, one cannot afford to keep track of all of the details of everything running, even if it were legally or contractually possible [1]. The size of the monitoring data would be such that its collection would interfere with the performance of the system [2]. So it has become essential to build flexible monitoring systems that are capable of doing analytics in situ, or in place [3], [4], without requiring that all of the data be delivered back to a central database before being processed.

Building upon our previous experience with such distributed enterprise monitoring as well as the rich experience with performance measurement and understanding of HPC codes, we have developed an approach for HPC *Mon*itoring *A*nalytics (MONA for short) that allows one to address the detailed performance questions that come from the high performance community for these new sorts of runtime environments. Specifically, MONA has been targeting the composition of in situ analysis workflows and coupled codes through the Adaptable I/O System (ADIOS) [5]. ADIOS allows scientific end users to leverage an API that is familiar for regular disk I/O, looking much like C or Fortran binary I/O routines. However, the careful design of the interface also means that the same read and write operations will function as memory-to-memory messaging receives and sends. From a software engineering perspective, this allows separately developed executables to be debugged and tested in combination using stored files for testing, but then it is a matter of changing one line in a configuration file to switch to in-memory coupling of the codes [6].

There are several available memory-to-memory connection providers within ADIOS, such as DataSpaces [7], FlexPath [8], ICEE [9], and other research variants. They each are tuned for different types of connection scenarios. We focus in this work on peer-to-peer connections between producers and consumers, an implementation of which is offered in ADIOS through FlexPath. FlexPath uses the EVPath [10] event messaging constructor library as its control and data plane, and its control plane is extensible to deal with a variety of elastic provisioning roles for in situ workflow performance management [11]. Although the work presented here focuses on the 1.x series of ADIOS, on-going work applies these techniques and insights to the ADIOS 2.x framework [12] as well.

ADIOS has been incorporated into a number of science codes, ranging from target exascale computing applications like the Fusion Whole Device Modeling, to commercial-driven codes like ExaFOAM, to experimental data harnesses like CASA for radio astronomy. The Fusion example is particularly indicative of the complexity of modern codes, as that involves supporting the coupling of two distinct applications, one that simulates the core of the plasma and the other that simulates the edge. Around all of these science scenarios are a host of analysis codes, temporal and spatial reduction algorithms, and diagnostic indicators that ADIOS also must support. The complexity and importance of these use cases help drive our usage scenarios for online capture and analysis of monitoring data.

This online monitoring analytics approach has a range of uses in practice. It offers the opportunity to build robust and dynamically managed systems that can extract peak performance from the available hardware, but that is never where a realistic science scenario starts. Instead, getting consistent, higher-quality feedback on the performance of in situ components is critical for a host of pragmatic planning, debugging, allocation, and code tuning operations. So the first goals of MONA are firmly rooted in improving the software engineering practices and platform tools for the computational scientists and engineers. Generating synthetic, time-synchronized traces so that the multi-executable workflows can be analyzed with traditional tracing tools is a simple threshold which can nonetheless offer significant benefit to the community. As we shall see, more advanced scenarios support features like using *this* run's performance model to inform the scheduler how best to partition the available nodes *next* execution.

More generally, the MONA approach opens up an open environment for innovation and development of new computational science use models. For example, one might leverage near-real-time detection of anomalous behavior in the code execution as a proxy for finding anomalous behavior or state changes in the simulation data – new simulation state means following different code paths, which can have measurable perturbations on performance data. It is also a key component in enabling a more interactive view of a running system; "dashboard" interactive summaries of a running code have been requested features implemented in ad hoc or bespoke ways before [6], but adopting shared monitoring analytics frameworks enables a richer set of reusable performance understanding tools.

The rest of this paper is constructed to demonstrate the progressive power of adopting these reusable monitoring analytics tools for current and future science applications. In §II, some of the key driving concerns arising from the science use cases are examined, which then enables us to articulate the core architectural concerns for the MONA approach which are described in turn in §III. Section IV then showcases a set of examples that use the MONA approach to inform the software engineering process for the coupled multi-scale, multi-physics, and in situ analytics workflows of advanced science applications. From the science drivers through these practical examples, we demonstrate the need for advancing the state of the art in performance monitoring and how the MONA approach can be used to create more robust computational science environments.

II. Science Drivers

To help ground this general discussion, we present in more detail two examples of research workflows from fusion energy science. Both of these examples draw from the challenges of understanding how to use magnetic confinement of hot plasma to generate clean fusion energy in a type of device called a Tokamak. Successful development of a production Tokamak would have significant impact on energy production worldwide, but the cost and timelines associated with physical experiments mean that there is a substantial interest in extracting the most possible science out of simulated scenarios so as to focus the experimental endeavors.

The first use case demonstrates the replacement of a classic and expensive post-hoc analysis with an in situ workflow that carries out the analysis concurrent with the main simulation. The Gyrokinetic Tokamak Simulation code (GTS) is a 3D particle-in-cell code used for studying micro-turbulence in the core of magnetic confinement fusion devices [13]. The micro-turbulence features are both important for engineering and fundamental physics reasons since they impact energy confinement, so adequately characterizing and understanding them is important. In this particular case, the analysis consists of a spectral decomposition of the micro-turbulence field across the whole volume and across time. This requires the calculation of a Fourier transforms on a domain decomposed dataset of 300,000 to 100 million grid points at regular intervals during the simulation (every 10 time steps of a 20,000-step calculation). To store such a volume of data for post-hoc analysis would be prohibitive, and so scientists that are using post-hoc approaches tend to use coarser (either in spatial or temporal dimensions) data sets. Moving the workflow to online analysis allows access to a much finer-grain data resolution, but it then makes it much harder to adequately understand the requirements and provisioning needs of the total in situ workflow. Monitoring of the individual executables and the analytics over the fused dataset enable better planning, provisioning, and debugging of the software as a whole.

Moving on from this simulation+analysis template for in situ workflows, our second example is taken from the US DOE Exascale Computing Project (ECP). The fusion Whole Device Modeling application (WDMApp) [14] being developed as part of ECP aims at coupling the most successful first principles codes for the simulation and prediction of the multiscale physics in magnetic confinement fusion experiments [15]. The core applications driving this model, the XGC [16] and GENE [17] codes, are currently the most compute-intensive parts of a WDM simulation. XGC uses a so-called "*total-f*" particle-in-cell method [18] to resolve some of the complex dynamics and structures that arise at the edges of the plasma. This more data-intensive algorithm is needed in order to capture all of the fine details and non scale-separable physics in that region. GENE, on the other hand, implements the highly efficient "δf" algorithm [19], which relies on scale separation and is thus valid mostly in the core (center) of the plasma but breaks down near the edge. Each coupled code has historically been developed independently of each other for the study of plasma physics at these different length and time scales (for more information, see the overview in [6]).

The data exchange and representational conversion needs for the coupling between these applications, as well as a suite of in situ analytics and I/O reduction and optimization routines, form a complex workflow that requires constant monitoring and feedback to avoid unnecessary periods of processor idling and load imbalance. For this two-in-one application, the coupling needs to be done in a synchronous way to conform to the chosen mathematical integration technique, which brings about a challenge in terms of resources allocation and process placement. In the complex environment of heterogeneous node architectures (CPU+GPU, DRAM and non-volatile RAM, etc.) operating on a shared high performance interconnect backbone, understanding why or where performance features arise only adds more challenges to the performance monitoring and management task.

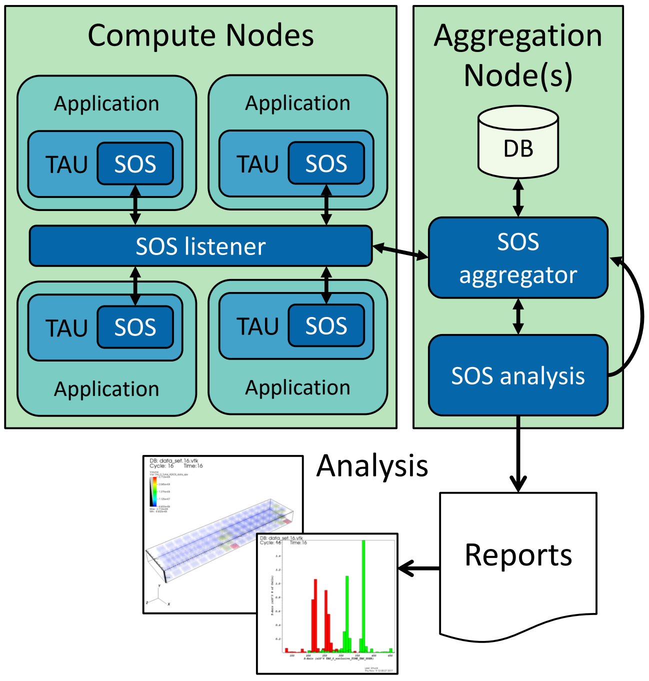

Fig. 1: The Scalable Observation System (SOS) monitoring system in action. Applications are linked with TAU, which contains an SOS plugin client that will aggregate TAU data to the on-node SOS listeners and on to the remote aggregator(s). Analysis can be run post-mortem or at runtime.

III. Architectural Design

The constraints of the examples described in §II combine with the goals of scalable, multi-application online monitoring frameworks from §I to give some general architectural principles for the construction of MONA.

- Static decision support means being able to synchronize the independent measurements from each of the components, generate coherent data set with all of them, and potentially reduce data to deal with extremes.
- Timeliness of data retrieval and handling are important to keep overheads low.
- Dynamic control should subsample monitoring streams to reduce data load decision analytics.
- Aggregation network architecture should be actively controlled.
- Parallel data generation with multiple tiers for filtering, reduction, and aggregation can lead to more robust monitoring analytics.
- Design informs choices of implementation to allow application specialization.
- Flexibility in monitoring infrastructure allows us to evaluate alternatives for several discrete use cases as detailed below.

In order to monitor parallel applications in a scientific workflow, it makes sense to re-purpose existing HPC measurement tools rather than engineer a new solution. There are several robust, scalable performance measurement tools (TAU [20], Scalasca [21], HPCToolkit [22], Vampir [23], and others), but broadly speaking, they are all designed for post-mortem analysis of single application executions. However, as was discussed in previous sections, we need the ability to monitor, perform analysis, and potentially take actions based on the result of those analyses and steer multiple applications in a workflow. An existing tool would need to be modified/enhanced because online analysis requires data aggregation with minimal interference with synchronous application communication. In addition, monitoring multiple applications requires additional data annotation beyond just node/thread data organization.

To overcome these issues, we have designed a data aggregation service for HPC called the *Scalable Observation System for Scientific Workflows (SOS)* [24]. SOS[1] is a framework for aggregating performance data from distributed workflows in support of in situ analysis and as a mechanism for feedback and control. SOS is designed as an aggregation network consisting of client data sources, intermediate data listeners (one per allocation node), and one or more aggregation daemons running on one or more additional allocation nodes (See Figure 1). Client applications send data to the listener daemons over a local socket, and the listeners are connected to the aggregators over EVPath. The aggregators are launched on additional hardware resources, within the allocation request but outside the application computation nodes. All communication to SOS happens on the same hardware resources as the applications, but without relying on or interfering with the application's use of MPI infrastructure (other than initialization/finalization).

We selected TAU as the performance measurement system primarily due to our familiarity with the tool, the ability to selectively insert and remove measurement probes, and the plugin infrastructure TAU offers to extend its functionality. SOS is an example of a plugin integrated with TAU with the ability to access TAU performance data from each application process and aggregate the data as performance profiles or partial/full event traces.

In a typical usage scenario, the SOS plugin is configured to initialize the SOS client connection during TAU plugin initialization. It is assumed that one or more SOS aggregation daemons are already running on additional allocation nodes, having been launched by the submission script. Because the plugin initialization happens during the `MPI_Init()` wrapper, we are guaranteed that all ranks are blocked, synchronously waiting for the call to return. As a result, as long as the SOS aggregator daemon(s) have been launched, the TAU plugin and SOS client code can handle construction of the aggregation network, including launching the listener daemon on each application node.

Within each application process, the TAU/SOS plugin spawns a new thread for the *SOS listener*, which can read TAU performance profiles periodically by iterating over all timers, threads, and metrics, including TAU counters. The SOS client library then sends the data to the SOS listener over TCP/IP sockets. Once the SOS listener has data, it aggregates the performance data to one or more SOS aggregators over EVPath[10]. The data is aggregated in SOS either in a file-based database, an in-memory database, or (for faster runtime analysis client access) a hash table of time-series data values. The data is periodically aggregated asynchronously over the SOS network in a tree-like fashion.

[1]The name *SOSflow* is used to refer to an implementation of SOS.

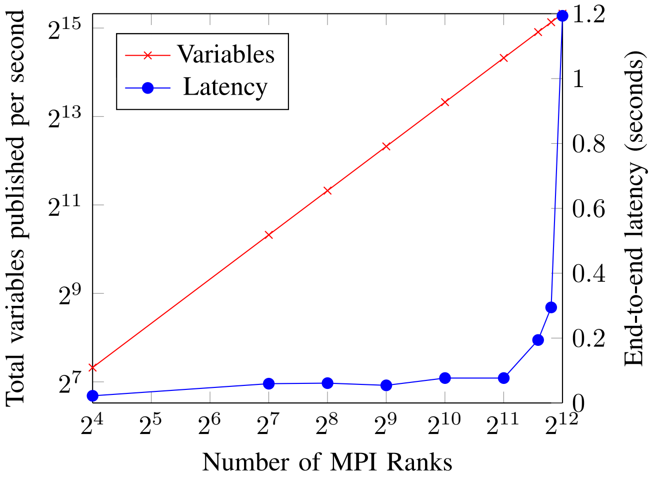

Fig. 2: Average end-to-end latencies when aggregating data over SOS. In this experiment, each MPI rank is publishing 10 variables per second, with 16 MPI ranks per Titan node. Each application node hosts an SOS listener daemon that communicates with the SOS aggregator daemon on the aggregation node.

IV. Case Studies in In Situ Monitoring Analytics

We present here a set of increasingly complex case studies of how online monitoring analytics can be applied to further the goals of increasing science performance and software efficiency in the complex HPC environments. We test these cases on machines from the Department of Energy's Computational Facilities, including Titan [25] at Oak Ridge National Laboratory, former #1 on the Top500 list, Cori [26] at NERSC, and Theta [27] at Argonne National Laboratory.

A. *In Situ Workflow Performance Monitoring*

One initial goal for online monitoring analytics as a user is to gain scalable access to monitoring data. Using monitoring analytics to aggregate and summarize an in situ workflow execution can make it much more understandable later. In order to determine the data ingestion capabilities of SOS on Titan, we performed a scaling study in which a constant rate of data was aggregated per application rank. As there are 16 cores per Titan node, the baseline case was a test MPI application with 16 ranks running on 1 node, and an SOS listener running on the same node. The test application will pack and publish a configurable amount of data during each iteration (organized by SOS as one *frame* of data), and then pause for a specified amount of time, up to N iterations.

TABLE I: Measurement overheads when executing GTS with 96 MPI ranks on Titan. *The tracing latency includes an average 1 second delay between pack and publish times for traced events. The send and process latency was only $0.74s$.

Config	Time(s)	Overhead	Timers	SOS Vars	SOS Values	Avg Latency
GTS	280.91	n/a	n/a	n/a	n/a	n/a
GTS+TAU	283.58	0.95%	n/a	n/a	n/a	n/a
GTS+TAU+SOS Profile	310.75	10.62%	58	$45,643$	$748,537$	$0.73s$
GTS+TAU+SOS Trace	316.59	12.70%	58	$70,850$	$4,346,883$	$1.73s$*

In our test case, we packed 10 name/value pairs of doubles per iteration from each application rank for 100 iterations. We doubled the number of allocated nodes, up to 256 nodes, or 4096 total MPI ranks. We define *latency* as the total time between when the SOS client library packed the variable until it was received by the aggregator. Figure 2 shows that up to 3584 application ranks ($35,840$ total variables per second), the aggregator can handle the stream. At 4096 ranks, the end-to-end latency is greater than the periodicity of the data generation rate, and the aggregator will eventually fail. Therefore, in deployments where more variables per second and/or more ranks are needed, additional aggregators can be deployed and can be queried concurrently for runtime analysis. It should be noted that the aggregators are also completely optional - for high data volume test cases, a "listener-only" configuration of SOS is possible, and the listeners can be queried by analysis resources concurrently. Listeners and aggregators are the same executable with the same capabilities, and only differ in how they are deployed.

We also performed some overhead measurements to determine how much of an effect there is in running an SOS listener daemon on the same resources where the computation is taking place. In this example, we ran the GTS application as configured for Section IV-B, executed using 96 MPI ranks placed on 6 nodes. Table I shows the comparison between running 1) GTS without TAU measurement, 2) GTS with TAU linked in to profile MPI and ADIOS calls, 3) GTS with TAU and SOS where TAU is aggregating profile data every 15 seconds, and 4) GTS with TAU and SOS "tracing", where TAU is packing every MPI collective and ADIOS timer as they complete, and publishing every 2 seconds. While $10-13\%$ is a relatively high amount, it should be noted that the compute nodes are oversubscribed in scenarios 3) and 4), because the SOS listeners running on each computation node are sharing resources with the application. No additional cores are reserved for the local processing of SOS data, which could be done in other configurations or with other applications by modifying `aprun` arguments. In addition, for scenario 3) the current implementation of the SOS plugin requires TAU to block the application while iterating the internal data structures. We are already working on an optimization that would eliminate that requirement, allowing true asynchronous behavior. Finally, scenario 4) represents a true stress-test for the infrastructure, providing a fully aggregated trace at runtime. For that reason we conclude that 12.7% overhead is reasonable for that added functionality.

B. *Using monitoring data to generate skeletal workflows*

To understand workflow performance, it is helpful to be able to capture aspects of the dynamic behavior of complex science workflows. Performance engineering and correctness testing of these codes is difficult, so it is useful to create surrogates through automated generation of benchmark codes that allow us to reproduce various aspects of this dynamic behavior. We use program trace information to populate a model of the communication and I/O behavior of each process. This is an extension of our previous work with the Skel tool [28], [29], and it provides a more complete view of program behavior by incorporating inter-process communication (MPI) and better representing total runtime rather than focusing solely on time to complete I/O calls.

The execution model is extracted as follows. We first identify events of interest, such as MPI_Comm events, MPI collectives, and I/O events, that we will represent in the model. Next, we observe that our program has three distinct phases: an initialization phase, a series of compute phases [30] interspersed with I/O, and a finalization phase. Consequently, we divide the trace events into these parts using SOS. For the initialization and finalization phases, we extract desired events from the trace. We then identify non-negligible gaps between such events, and insert "computation" events that track the time of those gaps. We follow the same procedure for the compute phases, however, instead of capturing every compute phase in its entirety, we select a representative phase

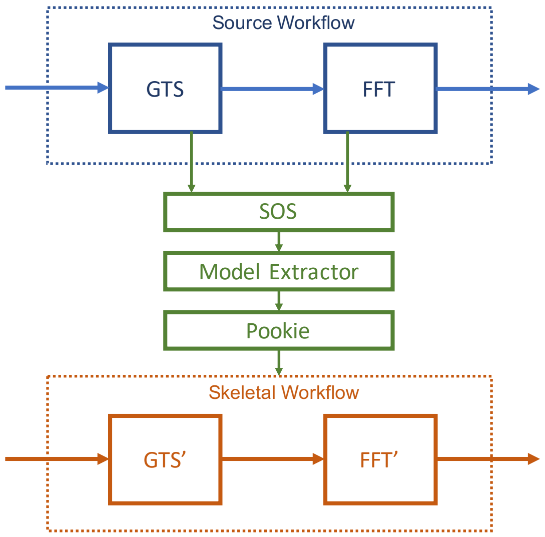

Fig. 3: Extracting a Pookie model using SOS

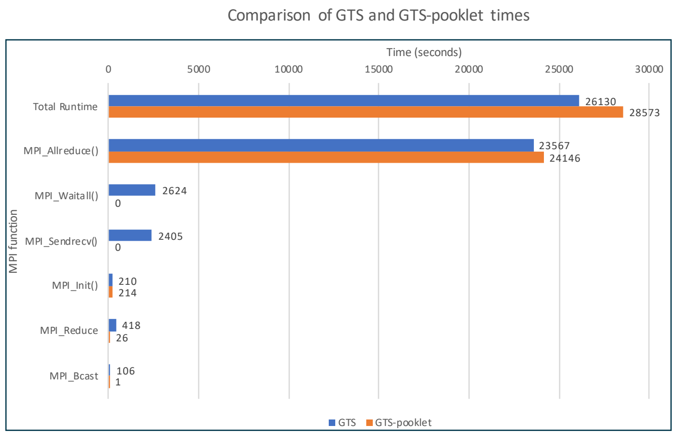

Fig. 4: Comparison of GTS and GTS-Pooklet traces for a 96-process run on Titan showing total time on all processes for selected MPI functions.

to capture, and keep track of the total number of phases.

To leverage such an execution model, we have constructed Pookie, a generative tool for producing a benchmark code that exhibits the behavior described by the model. Like Skel, Pookie leverages a code template that can be instantiated by an execution model to produce desired codes. The template can be modified by the user, allowing the user flexibility in leveraging model elements that represent desired aspects of a program. However, the results presented in this section focus only on the single model described above.

The use of SOS, the Model Extractor, and Pookie to skeletonize a complete workflow is illustrated in Figure 3. First, SOS is used to capture trace events for MPI communications and ADIOS I/O operations. Next, an extraction script queries SOS to acquire a unified view of communication and I/O across a collection of processes that make up an application, as well as cooperating analysis and visualization components, producing a set of application models that represent a complete workflow. Finally, Pookie uses the extracted application models to create a skeletal workflow that mimics the communication and I/O behavior of the original workflow.

To demonstrate the behavior of these tools, we first performed a 96-core run of the GTS application (discussed in Section II) on Titan. Next we used SOS to capture trace events, and used the Model Extractor to create a model of that run. Providing the model as input to Pookie, we produced a skeletal code, or "pooklet". Finally we ran the GTS-pooklet on Titan and compared the resulting traces.

Figure 4 shows a performance comparison between the original GTS run, and the GTS-pooklet run. There are several things to note about this comparison. First, the pooklet does a good job of representing the overall performance of the GTS run. It succeeds in showing that the `MPI_Allreduce` function accounts for the majority of the time of the run. Next, we see that `MPI_Waitall` and `MPI_Sendrecv` functions account for a not insignificant portion of the GTS runtime, but are not reflected in the pooklet trace due to our decision to omit point-to-point calls and focus on collective communications. Finally, looking at some of the smaller contributors to runtime, such as `MPI_Reduce` and `MPI_Bcast`, we see that the measurements differ by as much as two orders of magnitude. The reason is that our model captures only a single timestep, rather than a more comprehensive representation of all timesteps, so that the variability from step to step is not accurately modeled.

These last two points reflect the significance of the model schema in this process by highlighting that the choice of model dictates the degree to which the generated codes will reflect particular aspects of the behavior of the represented application. Creating application skeletons is not a one-size-fits-all endeavor, but rather the characteristics of a useful model will depend on the particular behavior that a user is attempting to capture. For instance, if it were important for our pooklet to be better at representing the step-by-step variability of the application, we might instead use the following approach. Rather than choosing to capture computation times from a single representative step, we could incorporate data from many steps into a statistical model. We could then use the model to randomly generate computation times that don't match the source trace, but instead attempt to mimic the overall distribution of times. This would leave the model less susceptible to inaccuracy caused by relying on a single time step, and better reflect some types of performance variability.

C. *Performance monitoring for variability study*

Performance variability or jitter caused by the concurrent use of shared resources with multiple users, such as network interconnect and I/O system, is an increasingly important issue as HPC systems grow larger and serve greater numbers of concurrent users. In particular, for long-running simulations that are part of complex workflows, detection of performance variability caused by external or internal resource contention becomes critical. Presence of such variability needs to be monitored and handled in a timely manner before delays propagate to other dependent tasks. However, it is currently quite challenging for users to monitor application performance, to check the current "weather" of the system, and to develop a code to respond to jitter events when they occur. To tackle such a challenge, we have explored using MONA to expedite the process of monitoring and consuming on-line monitoring data. We note that the term OS jitter [31] is commonly used to describe a particular class of variability, however, here we are describing a more general set of phenomena (which includes OS jitter) for which we do not attempt to assign a cause to the observed jitter.

In this section, we demonstrate the use of MONA to observe performance variability on various DOE facilities. First, we try to observe the existence of performance variability by using our motivating application, XGC, which was introduced in Section II. To this end, we ran multiple instances of the XGC application using identical input and compared the observed performance difference between runs. To ensure a fair comparison, we configured the application to be deterministic rather than stochastic to eliminate internal variability. In addition, to minimize a common source of variability, we set the application to utilize the parallel filesystem as little as possible, as shared filesystems are known to be a common source of variability, caused by the unpredictability of other users accessing concurrently.

In this experiment, we chose a coupling application developed for fusion energy science, called XGC-F_{total} coupling, which is used to run a fusion simulation (XGC), and its particle distribution analysis application (F_{total}) concurrently. Both are parallel codes written using MPI. We ran the same code with the same input at multiple times over the course of a few weeks on different DOE HPC machines: Theta at ANL and Titan at ORNL.

Fig 5 shows a few representative runs for each machine. We observed the performance variability in the XGC-F_{total} coupling over these runs. As the XGC-F_{total} run progressing with steps (represented in the x-axis), the trajectory of time per step (represented in the y-axis) varies run by run.

Identifying the sources of the variability is beyond the scope of this paper. However, we believe the main source is the use of the shared interconnect network, which is shared with multiple concurrent users. Different placement of processes between jobs and different neighboring applications also contribute to such variability. SOS enables us to monitor and collect such information necessary for further performance improvement.

Second, we demonstrate how we analyze captured monitoring data from SOS by using autocorrelation analysis [32]. Autocorrelation is an analysis method developed for time-series data to find repeating patterns or periodicity. For a given time-series data, autocorrelation is to find the degree of similarity, or correlation, within the same input after varying delays (or lags). With mathematical notation, we can define autocorrelation as follows. For a time-series data $\boldsymbol{x}$ represented by a vector $(x_1, x_2, ..., x_N)$ of N observations, autocorrelation of $\boldsymbol{x}$ with k delays, r_k, is defined by

$$r_k = \frac{\sum_{i=1}^{N-k}(x_i - \mu)(x_{i+k} - \mu)}{\sum_{i=1}^{N}(x_i - \mu)^2} \tag{1}$$

where μ is a mean value of x_i's.

In Fig 6, we show an example of a performance trace collected from a single XGC-F_{total} run (top) and its corresponding autocorrelation analysis result (bottom). In the autocorrelation analysis, the observed performance maintains a strong correlation (0.7-1.0) up to about the first 25 lags (timesteps) and drops to near zero at around 160 lags. Correlations below the 95% confidence interval (dotted line) are considered to be random behavior. We can also observe week periodic correlations, which appear as bumps in the plot, and which can be used to identify periodic patterns of performance. This analysis implies we can analyze the monitoring data from SOS to identify the performance variability patterns for developing on-line performance tuning.

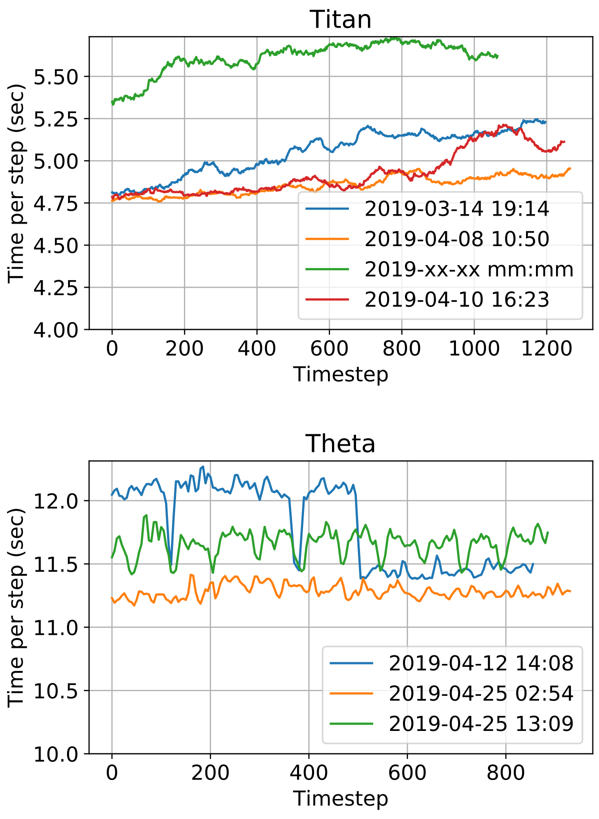

Fig. 5: Performance variability of XGC-F_{total} measured on Titan at ORNL and Theta at ANL at different times. The performance of fusion coupling workflows shows variability. We used the following ratios of MPI processes between XGC and F_{total}: 8,192 to 256 on 528 Titan nodes, as well as and 2,048 to 256 on 132 Theta nodes.

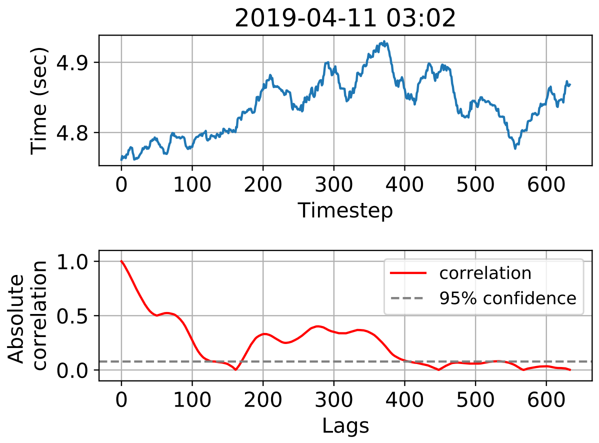

Fig. 6: Example of variability analysis of XGC-F_{total}. The performance monitoring data from MONA (top) and its result with autocorrelation (bottom) are demonstrated. We measured by using 8,192 XGC to 256 F_{total} MPI processes over 528 Titan nodes.

D. Communication pattern monitoring for task placement

Understanding the impact of communication patterns in a parallel application is essential for identifying bottlenecks, avoiding workload imbalance, deciding task placement, etc. However, the increasing scale and complexity of parallel applications running on HPC machines due to factors such as coupled executions, in situ analysis, and node heterogeneity makes it increasingly challenging for a user to comprehend the communication patterns displayed by non-trivial workflows. Using our MONA tools for preparatory model extraction is ideal for assisting with this challenge. In this section, we discuss how information from SOS can be used to improve the performance of a complex coupling workflow by offering a mechanism to optimize task placement in HPC environments.

The task placement problem, which seeks a mapping between parallel processes and distributed nodes to optimize the performance of the parallel application by minimizing communication congestion and interference, has been long studied in the area of HPC research and workflow systems. While initial studies have been constructed based on static information about an application's communication pattern, our previous work [33], named Task Graph Embedding or TGE, focuses on optimizing placement based on dynamic system information as well as inter- and intra-communication patterns within and between multiple concurrent applications. However, timely access to monitoring data has been an issue for users. A major goal of designing monitoring infrastructure is to allow users to assemble such information at runtime without excess impact on application performance.

Fig 7 shows how we leverage the MONA tools to obtain the communication pattern data that allows TGE to find an optimal process placement for our motivational application, called XGC and GENE coupling (discussed in Section II). Both XGC and GENE are independent, parallel fusion simulation codes focusing on different plasma physics happening on a specific spatial region (the edge area and the core area respectively), in a fusion reactor. When we are running them as a coupled workflow, they actively exchange a set of boundary condition data shared by the two codes, which creates a complex data flow pattern due to their inter- and intra-communications. Task placement for XGC-GENE coupling is more challenging due to these complex data flows.

In our experiment, we build both XGC and GENE code with SOS so that we can extract two types of information; i) communication patterns of XGC, GENE, and XGC-GENE interactions and ii) topology information of allocated resources, such as node layout, connectivity graph, and pairwise node distances. Based on that information as input, TGE then calculates optimal process placement by using a graph-theoretic embedding algorithm [33].

Figure 8 shows a comparison of XGC-GENE coupling with and without TGE and SOS on Titan. The boxplot in the figure draws the summary statistics of multiple runs using a fixed number of nodes (1024) but using different allocation diameters. The diameter, defined by the longest hop-distance

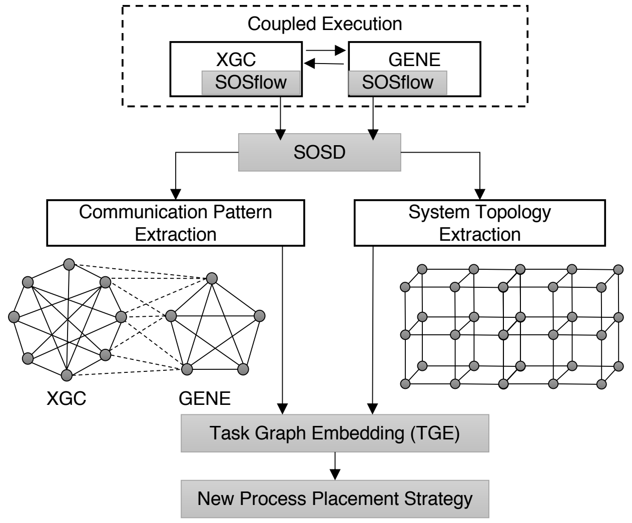
Coupled Execution
XGC
SOSflow
GENE
SOSflow
SOSD
Communication Pattern Extraction
System Topology Extraction
XGC
GENE
Task Graph Embedding (TGE)
New Process Placement Strategy

Fig. 7: Using TGE with SOS for XGC-GENE coupling.

between any two nodes, illustrates the compactness (or sparseness) of an allocation. On Titan, like most other HPC systems, each job is assigned to use a dynamically allocated set of nodes. The particular set of nodes that are assigned depends on availability, and largely determines the connectivity between nodes. Within the set of assigned nodes, we are able to allocate tasks in any order we would like. TGE attempts to tell us how to assign tasks to the nodes such as to obtain optimal performance. The results shown in Figure 8 demonstrate both performance improvement and reduced variability when using TGE over various ranges of allocation diameters, as compared with the default placement method (no SOSflow and canonical process ordering).

The astute reader will note that we generally observe better performance as the allocation diameter increases, which is a surprising result. Our rule of thumb has been that communication performance is *higher* with compact allocation since there is less chance for external interference and generally fewer network hops are required. Regardless of the details of this specific example, we think this demonstrates exactly why we need runtime monitoring, as performance often presents in non-intuitive ways. Identifying and reasoning about the cause of poor performance is key for optimizing in situ workflows. In these particular experiments, we believe this counter-intuitive trend was caused by internal interference in XGC-GENE coupling, combined with the fact that no consistent external noise was encountered during the experiments, but further testing would be necessary to determine root cause. Tools like MONA will become increasingly critical as more complex workflows are introduced.

V. Related Work

The literature on performance measurement, modeling, and tuning is long and complex, even within the restricted space of high performance computing. A comprehensive survey of the space would be welcome, but we restrict ourselves here

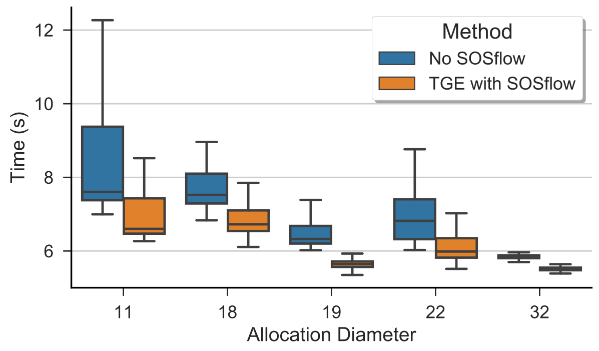

Fig. 8: XGC-GENE performance enhancement with SOS on Titan when placing processes onto 1024 node allocations.

to a selection of key connection points. There are a number of comprehensive distributed monitoring systems that enable capturing multifaceted monitoring data such as Ganglia [34], Nagios [35], Munin [36], Host sFlow [37], Heka [38] and Hindsight. Most of these, though, are more focused on resource utilization and hardware status for machine operator needs. Ganglia scales well and it targets monitoring of federations of HPC clusters. It uses XML to represent data, XDR as a data transport and RRDtool (Round Robin Database Tool) [39] to store and visualize data. Nagios offers a comprehensive monitoring solution for IT infrastructure allowing for monitoring applications, services, operating systems, network protocols, system metrics, etc [35]. It can be tailored to the user's needs with a rich selection of plugins. Munin [36] aims at providing aid in diagnosing performance issues. Munin emphasizes plug-and-play capabilities, easy of developing new plugins tailored to the user's specific needs. It focuses on monitoring resource performance, and, similar to Ganglia, it exploits RRDtool. The Host sFlow [37] aims at providing an open source monitoring solution to capture various server-related performance metrics. sFlow specifies the format of the exported data, and if a particular device is a sFlow capable device, the software can collect data from such a device. Among others, Ganglia, ntop, and Wireshark are capable of handling sFlow data. Mozilla Hindsight[40] is a higher-performance successor to the Heka [38] distributed monitoring analysis framework that consists of a C-based transport system with Lua plugins for data processing.

Another approach to online distributed monitoring and aggregation of information is provided by the DIMVHCM [41] model. Its principal service goals are around performance understanding through visualization tools rather than the holistic workflow applications and runtime environment. As a result, DIMVHCM provides only limited support for in situ query of information.

The Lightweight Distributed Metric Service (LDMS) [42] captures system data continuously to obtain insight into behavioral characteristics of individual applications with respect to their resource utilization. However, LDMS can not be configured with and used directly by an application. Additionally, it does not allow for richly-annotated information to be placed into the system from multiple concurrent data sources per node. As data aggregation infrastructure, SOS has many things in common with the Multicast Reduction Network (MRNet) [43]. MRNet utilizes a fully actualized Tree Based Overlay Network to aggregate data to a single top-level node. Because of its dependency on MPI to perform discovery and configuration of the network [44], it cannot easily be used in workflow scenarios with multiple distinct MPI application contexts. Similarly, some of the cloud/enterprise-scale monitoring examples mentioned in the introduction [2], [3] have interesting overlaps in motivation and technique.

The field of provenance capture has also yielded works that overlap with the real-time analytics space. Komadu [45] is a system to capture, analyze, and reduce streams of provenance data that includes both performance and procedural information, with a target towards Data Lake and similar map-reduce processing environments. Other similar work that mixes provenance with performance information is used in commercial offerings for tuning web application performance with tools like Amazon CloudWatch and Lambda [4].

At the far end of real-time application of MONA-type functionality are systems that monitor and adapt application behavior with changing circumstances, such as Goldrush[46] and Landrush [47]. Both systems use fine-grained monitoring and scheduling to ”steal” idle resources in ways that minimize interference between the simulation and in situ analytics, with Goldrush focusing on harvesting CPU cycles, while Landrush focuses on GPU cycles. Both, however, rely on bespoke monitoring systems that tie directly to their specific platform requirements, rather than general science-user programmability.

VI. Conclusions & Future Directions

As new application scenarios continue to use online, in situ, and code-coupled workflows in new ways, it is worthwhile to evaluate and, as appropriate, borrow concepts for monitoring analytics from different areas such as cloud computing. Here we have explored the usefulness of runtime performance monitoring and analytics (MONA) using several examples from science use cases, where we leverage infrastructure built with the Adaptable I/O System (ADIOS) and the Scalable Observation System (SOS). In Section IV-A, we described a scaling study investigating the performance impact of gathering runtime performance information. Section IV-B examined a benchmark generation tool that takes advantage of runtime tracing and performance monitoring of in situ workflows. The need to study and monitor performance variability was explored in Section IV-C. Finally, in Section IV-D we discussed the use of our monitoring infrastructure to inform task placement algorithms allowing the optimization of workflow communications. Each of these distinct usages of MONA capabilities could have been built as individual, bespoke systems. However, this progression demonstrates the need for new *common* performance frameworks for scalable computational science.

We foresee many opportunities to leverage an application runtime monitoring and analysis infrastructure such as MONA. There is an ongoing effort to increase performance predictability and portability for in situ applications, and tools like MONA will continue to be central to those efforts. Availability of performance information at runtime will also help to drive application code specialization and/or just-in-time customization, where different implementation options can be chosen based on measured performance on a particular platform. Furthermore, the performance information can also be used to inform future user-space management components capable of dynamically configuring workflows based on the observed performance and/or constraints of the software and hardware.

Engineering of research software in the extreme scale computing environment is difficult, and it requires new tools as we move away from the traditions of monolithic code construction. The MONA approach has demonstrated a path towards creating more robust and usable performance toolkits for science applications at the cutting edge.

Acknowledgment

We gratefully recognize the support from the Department of Energy's Office of Advanced Scientific Computing Research (ASCR Research) for enabling this work. Additionally, this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, the National Energy Research Scientific Computing Center (NERSC), and the Argonne Leadership Computing Facility which are supported by the Office of Science of the U.S. Department of Energy under Contract Nos. DE-AC05-00OR22725, DE-AC02-05CH11231, and DE-AC02-06CH11357, respectively.

References

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220. [Online]. Available: http://doi.acm.org/10.1145/1294261.1294281

[2] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf, "A flexible architecture integrating monitoring and analytics for managing large-scale data centers," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 141–150. [Online]. Available: http://doi.acm.org/10.1145/1998582.1998605

[3] C. Wang, I. A. Rayan, G. Eisenhauer, K. Schwan, V. Talwar, M. Wolf, and C. Huneycutt, "VScope: Middleware for troubleshooting time-sensitive data center applications," in *Middleware 2012 - ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings*, ser. Lecture Notes in Computer Science, P. Narasimhan and P. Triantafillou, Eds., vol. 7662. Springer, 2012, pp. 121–141. [Online]. Available: https://doi.org/10.1007/978-3-642-35170-9_7

[4] J. Varia, S. Mathew *et al.*, "Overview of Amazon Web Services," *Amazon Web Services*, pp. 1–22, 2014.

[5] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, may 2014. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3125

[6] J. Y. Choi, C.-S. Chang, J. Dominski, S. Klasky, G. Merlo, E. Suchyta, M. Ainsworth, B. Allen, F. Cappello, M. Churchill, P. E. Davis, S. Di, G. Eisenhauer, S. Ethier, I. Foster, B. Geveci, H. Guo, K. A. Huck, F. Jenko, M. Kim, J. Kress, S.-H. Ku, Q. Liu, J. Logan, A. Malony, K. Mehta, K. Moreland, T. Munson, M. Parashar, T. Peterka, N. Podhorszki, D. Pugmire, O. Tugluk, R. Wang, B. Whitney, M. Wolf, and C. Wood, "Coupling exascale multiphysics applications: Methods and lessons learned," *2018 IEEE 14th International Conference on e-Science (e-Science)*, pp. 442–452, 2018.

[7] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: An interaction and coordination framework for coupled simulation workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 25–36. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851481

[8] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki, "Flexpath: Type-based publish/subscribe system for large-scale science analytics," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014*. IEEE Computer Society, 2014, pp. 246–255. [Online]. Available: https://doi.org/10.1109/CCGrid.2014.104

[9] J. Y. Choi, K. Wu, J. C. Wu, A. Sim, Q. G. Liu, M. Wolf, C. Chang, and S. Klasky, "ICEE: Wide-area in transit data processing framework for near real-time scientific applications," in *Proceedings of BDAC 2013 : 3rd International Workshop on Big Data Analytics: Challenges and Opportunities*, 2013.

[10] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: Opportunities and challenges at exascale," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '09. New York, NY, USA: ACM, 2009, pp. 2:1–2:10. [Online]. Available: http://doi.acm.org/10.1145/1619258.1619261

[11] J. Dayal, J. Lofstead, G. Eisenhauer, K. Schwan, M. Wolf, H. Abbasi, and S. Klasky, "SODA: Science-driven orchestration of data analytics," in *2015 IEEE 11th International Conference on e-Science*, Aug 2015, pp. 475–484.

[12] "ADIOS2 Github," https://github.com/ornladios/ADIOS2, 2019.

[13] W. Wang, Z. Lin, W. Tang, W. Lee, S. Ethier, J. Lewandowski, G. Rewoldt, T. Hahm, and J. Manickam, "Gyro-kinetic simulation of global turbulent transport properties in tokamak experiments," *Physics of Plasmas*, vol. 13, no. 6, p. 092505, 2006.

[14] "ECP WDMApp," https://www.exascaleproject.org/project/wdmapp-high-fidelity-whole-device-modeling-magnetically-confined-fusion-plasmas/, 2017.

[15] J. Dominski, S. Ku, C.-S. Chang, J. Choi, E. Suchyta, S. Parker, S. Klasky, and A. Bhattacharjee, "A tight-coupling scheme sharing minimum information across a spatial interface between gyrokinetic turbulence codes," *Physics of Plasmas*, vol. 25, no. 7, p. 072308, 2018.

[16] C. S. Chang, S. Ku, P. H. Diamond, Z. Lin, S. Parker, T. S. Hahm, and N. Samatova, "Compressed ion temperature gradient turbulence in diverted tokamak edge," *Physics of Plasmas*, vol. 16, p. 05168, 2009.

[17] M. K. F. Jenko, W. Dorland and B. N. Rogers, "Electron temperature gradient driven turbulence," *Physics of Plasma*, vol. 7, p. 1904, 2000.

[18] S. Ku, R. Hager, C. S. Chang, J. M. Kwon, and S. E. Parker, "A new hybrid lagrangian numerical scheme for gyrokinetic simulation of tokamak edge plasma," *J. Comp. Phys.*, vol. 315, pp. 467–475, 2016.

[19] S. E. Parker and W. W. Lee, "A fully nonlinear characteristic method for gyrokinetic simulation," *Physics of Fluids B*, vol. 5, no. 1, pp. 77–86, 1993.

[20] S. S. Shende and A. D. Malony, "The TAU parallel performance system," *The International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[21] M. Geimer, F. Wolf, B. J. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 702–719, 2010.

[22] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCToolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[23] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The Vampir performance analysis toolset," in *Tools for High Performance Computing*. Springer, 2008, pp. 139–155.

[24] C. Wood, S. Sane, D. Ellsworth, A. Gimenez, K. Huck, T. Gamblin, and A. Malony, “A scalable observation system for introspection and in situ analytics,” in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, Nov 2016, pp. 42–49.
[25] “Titan,” https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/, accessed: 2019-08-02.
[26] “Cori,” https://www.nersc.gov/users/computational-systems/cori/, accessed: 2019-08-02.
[27] “Theta,” https://www.alcf.anl.gov/theta, accessed: 2019-08-02.
[28] J. Logan, S. Klasky, J. Lofstead, H. Abbasi, S. Ethier, R. Grout, S. Ku, Q. Liu, X. Ma, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, “Skel: Generative software for producing skeletal I/O applications,” in *2011 IEEE Seventh International Conference on e-Science Workshops*, Dec 2011, pp. 191–198.
[29] J. Logan, J. Y. Choi, M. Wolf, G. Ostrouchov, L. Wan, N. Podhorszki, W. Godoy, S. Klasky, E. Lohrmann, G. Eisenhauer, C. Wood, and K. Huck, “Extending Skel to support the development and optimization of next generation I/O systems,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 563–571.
[30] Y. Jin, X. Ma, M. Liu, Q. Liu, J. Logan, N. Podhorszki, J. Y. Choi, and S. Klasky, “Combining phase identification and statistic modeling for automated parallel benchmark generation,” in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS ’15. New York, NY, USA: ACM, 2015, pp. 309–320. [Online]. Available: http://doi.acm.org/10.1145/2745844.2745876
[31] K. B. Ferreira, P. Bridges, and R. Brightwell, “Characterizing application sensitivity to os interference using kernel-level noise injection,” in *SC ’08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Nov 2008, pp. 1–12.
[32] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
[33] J. Y. Choi, J. Logan, M. Wolf, G. Ostrouchov, T. Kurc, Q. Liu, N. Podhorszki, S. Klasky, M. Romanus, Q. Sun, M. Parashar, R. M. Churchill, and C. Chang, “TGE: Machine learning based task graph embedding for large-scale topology mapping,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 587–591.
[34] “Ganglia site,” http://ganglia.info/, 2019.
[35] “Nagios site,” http://www.nagios.org/, 2013.
[36] “Munin site,” http://munin-monitoring.org/, 2019.
[37] “Host sFlow,” http://sflow.net, 2013.
[38] “Heka–Data Acquisition and Collection Made Easy,” https://github.com/mozilla-services/heka, 2013.
[39] “RRDtool–logging and graphing,” http://oss.oetiker.ch/rrdtool/, 2013.
[40] “Hindsight,” https://github.com/Securing-DevOps/logging-pipeline, 2019.
[41] R. K. Tesser and P. O. A. Navaux, “DIMVHCM: An on-line distributed monitoring data collection model,” in *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2012, pp. 37–41.
[42] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, “The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications,” in *SC’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 154–165.
[43] P. C. Roth, D. C. Arnold, and B. P. Miller, “MRNet: A software-based multicast/reduction network for scalable tools,” in *SC’03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. IEEE, 2003, pp. 21–21.
[44] A. Nataraj, A. D. Malony, A. Morris, D. Arnold, and B. Miller, “A framework for scalable, parallel performance monitoring using Tau and MRNet,” in *International Workshop on Scalable Tools for High-End Computing (STHEC 2008)*, 2008.
[45] I. Suriarachchi, S. Withana, and B. Plale, “Big provenance stream processing for data intensive computations,” in *2018 IEEE 14th International Conference on e-Science (e-Science)*, Oct 2018, pp. 245–255.
[46] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky, “Goldrush: Resource efficient in situ scientific data analytics using fine-grained interference aware execution,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’13. New York, NY, USA: ACM, 2013, pp. 78:1–78:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503279
[47] A. Goswami, Y. Tian, K. Schwan, F. Zheng, J. Young, M. Wolf, G. Eisenhauer, and S. Klasky, “Landrush: Rethinking in-situ analysis for GPGPU workflows,” in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2016, pp. 32–41.