SOSflow and a Semantic Workflow Performance Model for Understanding Variability in Scientific Workflows at Scale

Chad Wood
Kevin Huck
Allen Malony
Department of Computer and Information Science
University of Oregon
Eugene, OR United States
Email: cdw@cs.uoregon.edu

***Abstract*—SOSflow provides a run-time system and performance model designed to enable the characterization and analysis of complex scientific workflow performance at scale.**

***Keywords*-hpc; exascale; in situ; performance; monitoring; introspection; scientific workflow;**

I. Introduction

Here we set up the problem space we are addressing.

It will be a good time!

[NOTE: Come back and flesh out the INTRO and ABSTRACT at the end, once we're sure the content we are after is a part of the paper.]

II. Motivations for Tracking Variability

[NOTE: More than simply demonstrating that there are different and possibly incompatible motivations for variability research, this section is setting up the necessity for a new performance model.]
Due to the novelty of this research area, there will be opportunities to further refine the way in which research is taxonomized, and to establish a set of definitions that provide adequate coverage of the conceptually distinct types of variability studies. Though such preliminaries are not our primary purpose here, it is a useful exercise to consider some of the different purposes behind current studies of performance variability. The design choices made for the SOSflow system and its inherent workflow performance model emerged from reflecting on the general nature of variability studies, the divergent purposes that are create tensions between the efforts being undertaken in this new field, and the anticipated realities of writing infrastructure codes for exascale HPC clusters.

A. Why Variability is Tracked

Many factors have contributed to the emergence of variability studies as an important research topic within the HPC community. At a low-level, HPC node engineering has grown in complexity and sophistication, many on-core processor behaviors that used to be isolated, synchronous, and predictable, are now interrelated, data-driven, asynchronous, and impossible to predict a priori. As core density increases on the nodes of a cluster (TODO: Cite paper showing this is the necessary direction of exascale) the unpredictability and inconsistency of the hardware itself becomes an increasingly significant contributor to observed variability. Hardware is an important source to consider when it comes to variability because there is almost nothing that can be done to control for it, the noise has to be admitted in the results, and therefore is an important part of the output of any performance-related experiment. At higher-levels of description, variability in workflow performance can be introduced by many different sources:

- Versions of software libraries across can differ across clusters, or even the same cluster across time.
- Tuning factors to extract maximum performance from a code can vary across clusters even if the code and the data do not change. Shared filesystem performance, data transport methods, available per-process memory, co-processor presence and architecture, ...and more, all can be responsible for influencing sensitivity to novel tuning factors.
- Concurrent activity elsewhere on the same cluster, activity that will necessarily vary between every iteration of the workflow, may be having a significant impact on observed performance.
- At extreme scales, parts of a simulation are almost guaranteed to fail due to the marginal failure rates of hardware components approaching absolute certainty as the number of involved components increases. These failures cannot be accurately predicted a priori, and the design constraints that account for and respond to them introduce performance perturbation and further complexity.
- ...
- ...

1) Awareness of Fine-Grained Performance Jitter: *[NOTE: This section is purposed with showing that even episodic close analysis of individual components at small scale will have issues with significant variability. There are tractible concerns (demonstrating sensitivity to var.) and inherently intractible concerns (controlling var. across runs).]*

Even in cases where nothing can be done to influence the variations in performance, it is important to recognize that performance variability is present and to attempt to both quantify it and render reasonable attributions of its source[s]. Significant variability between runs is seen (SCHULTZ paper/graph) when tracking a single simple experiment, even if the input data, job queue parameters, and hardware allocation is held constant. At extreme scales, on line detection and attribution of variability of a scientific workflow will require well-annotated metadata to facilitate "apples to apples" comparisons driven by unsupervised machine learning rather than a priori developer knowledge or offline centralized analysis. It is important to characterize a code's sensitivity to variability, as well as a cluster's propensity for creating performance variability.

2) Accuracy in Performance Research: [NOTE: This section is intended to motivate a new model for performance when considering scientific workflows on exascale systems. Nail the coffin shut on existing performance research validation tools and techniques. They are intractible, per the above notes, and irrelevant, per this discussion]

Performance research is principally concerned with decreasing the resource consumption and compute time required by low-level components and libraries that are used when constructing higher-level scientific workflows. For example: In order to validate a 5 percent increase in some code's performance, it will be necessary to show that performance increase was observed across a vast array of runs and hardware allocations, especially if it is the case that a particular HPC cluster (when in an overall state similar to the one it was in during those workflow runs) has a history of performance variability with any statistical significance relative to the observed performance gain. When it comes to the behavior of codes at extreme scales, accuracy validation using traditional models of component-based performance analysis will become cost prohibitive in both allocation consumption and developer time.

3) Reproduction of Experimental Results: Scientific workflows attempt to yield results that have truth-coorespondence with the physical world with some overt degree of significance.

B. How Variability is Tracked

1) Concepts and Methods:

2) Existing Models and Tools:

C. The Utility a Comprehensive Workflow Performance Model

1) Attribution:

2) Resource Requirement Prediction:

3) Automated Component Performance Tuning: Don't want conflicting optimizer purposes. Need to know where the hotspots *really* are and not

4) Intelligent Compiler Hints: Don't have to burn 1,000,000 hours of allocation to learn that you just re-created last year's mistaken

5) Intelligent Job Scheduling:

III. A Performance Model for Scientific Workflows

Traditionally, HPC performance monitoring is focused on low-level efficiency of an application binary on some particular iron. Higher-level systems (TACC Stats) allow the tracking and exploration of execution wall-time for various library versions, integration of multiple modalities of information (LDMS) like program invocation or work allocation across a cluster as informed by network congestion statistics, and other hybridized or meta-execution data points. Low-level metrics are more naturally suited for off-line episodic performance analysis of individual workflow components, but cannot yield insight into the run-time performance of a complex workflow. Characterizing and understanding the emergent properties of a workflow comprised of many components that are interacting asynchronously across a distributed HPC cluster requires taking a new approach to the problem, especially when considering the extreme scales of parallelism to which scientific workflows are being driven.

A. Workflow Variability is Revealed by Invariant Meaning

B. Levels of Description and "The View from Anywhere"

Not knowing *a priori* what component or layer of the workflow will be responsible for the introduction of variability, the workflow performance model needs to be populated by a diversity of information sources that provide metrics with tailored metadata and both logical and concrete events, arriving in real-time from many different layers of activity in a workflow:

- Simulation
- Algorithm
- Application
- Libraries
- Environment
- Developer Tools
- Operating System
- Node Hardware
- Network
- Enclave
- Cluster
- Workflow

- Epoch

Each of these layers constitutes a *level of description* for workflow performance. If the Simulation layer produced data representing the evolution of a system over N seconds of simulated time, and the Application layer requires M seconds of real-world computation to yield that data, the relationship between N and M should be a valid performance metric to report, compare across runs, and to attempt to generally optimize through parameter convergence.

C. Semantics

All information that is gathered by the monitoring system should be annotated as richly as possible to maximize its usefulness when performing analytics. Hand-annotated codes will have the most to offer an analytics engine, values that are tracked will be able to carry a full spectrum of high-level tags that express what that data point means and what could be expected of it, in structure preserving a human programmer or user's understanding while also being compatible with unsupervised machine-learning tests for significance and other advanced analysis techniques.

Any episodic performance measurement, such as run-time TAU instrumentation, can also be injected into the SOSflow engine, and the SOSflow runtime will be able to differentiate from information pushed directly by a layer, and information that is being captured by middle-ware tools.

[NOTE: Insert table heres of some of the enum values alongside plain-english descriptions.]

Semantic information is local to the "publication handle" (pub) created by a source that is contributing to SOSflow. Sources can create multiple pubs to distinctly represent potential compound or complex roles. Pubs carry their own pub-wide semantic markups, including the origin layer, a role within that layer, and information about the node that the source process is running on. Semantic markups are then nested inside of the pub handle, as each value that is pushed into the SOSflow system through a pub handle also comes with a rich set of high-level semantic tags that stay affiliated with it over time. While deep off-line data analytics can reveal unforseen correlations between various aspects of the workflow or the data set it is operating over, the interest in real-time analytics and performance tuning gives value to expressing "relationship hints" between values tracked by the system. These hints can be used to direct in situ analytics and identify deviations from expectations; anything that can be overtly identified as an expectation for a value can be used to narrow the search spacec when doing unsupervised machine learning over gathered workflow performance data.

IV. SOSflow

Applying a novel semantic workflow performance model to actual run-time environments necessitated the development of new infrastructure software, and so the second contribution to the general challenges of monitoring scientific workflows is the SOSflow software artifact.

The initial design goals of SOSflow are:

1) Facilitate capture of scientific workflow performance data and events in situ (i.e. "on node") from a variety of sources and perspectives at the same time.
2) Annotate the gathered data's semantic meaning with expressive "high-level" tags that facilitate contextualization and introspective analytics.
3) Store the captured data on node in a way that can be searched with dynamic queries in real-time as well as being suitable for long-term centralized archival.

...the SQLite3 open-source DB engine has been selected during the initial development phase for on-node storage, but the API is modularized and a different format/mechanism could be used instead. On-node storage is logically separated off-node data transport mechanisms, but they could be the same backend if the enabling technology supports that.

SOSflow is divided into two main parts in it's current incarnation:

- sosd - Daemon process running on each node
- libsos - Library of routines for interacting with the daemon

The sosd daemon launches before a scientific workflow begins, and passively listens on a socket. The port that is available to all of the sources on any given node is found in the "SOS_CMD_PORT" environment variable. Many programs and layers of programs can connect to the daemon and send information in. Library-to-daemon communication is invisible to the SOSflow user and happens entirely on-node using a simple stateless protocol. When the workflow is completed, the sosd_stop tool is executed on each node, it signals the daemon to flush buffers and close down.

A. Behavior

[NOTE: Here I can introduce a narrative or a figure or both, to explain how a typical SOSflow session should happen on the cluster. I am skipping this for now, to be able to release this document for revision and comment. This section is obvious and easy to both write and edit, but I want to get this out to you guys.]

B. Implementation

1) Language: C

Table I
KEY SOURCE FILES FOR SOSFLOW

sos.h / sos.c	Becomes libsos, the core functions of SOSflow
sosd.h / sosd.c	SOSflow daemon
sos_debug.h	Debugging off / on (level) knobs
demo_app.c	The "Hello, world." of SOSflow value injection
sosd_cloud_mpi.c	Off-node transport using simple MPI calls
sosd_db_sqlite.c	On-node DB creation and value-injection

2) External Requirements:

- cmake
- pthreads
- MPI
- Sqlite3 (*May change across implementations)

3) Key Source Files: See Table 1.

C. Limitations and Concerns

The value of SOSflow is directly proportional to the quantity and quality of the sources that are pushing semantically-annotated data into it. At this time the use-case development has centered on the ADIOS scalable I/O library, due to its deep integration with many existing scientific workflows and industry standard tools like VisIt. ADIOS instrumentation is a source of significant observable data and an actionable target for effective feedback and control.

The more sources that are instrumented with SOSflow the less limited the workflow performance model will be. When only one or two layers are instrumented, the benefits of the semantic annotation and the workflow performance model are naturally limited.

The SOSflow software artifact is in its initial development cycle and has room for improvement in various software engineering facets: memory subsystem interaction, diversity of storage and transport mechanisms, and quality-of-service guarantees to both producers and consumers of SOSflow data.

D. Results of Initial Benchmarking

[NOTE: We need to decide what we can put in here (or if we should even have this section at this time) ... it could be something as simple as PERCENTAGE of difference in the execution time of a synthetic workflow when the SOSflow system is activated compared to when it is not activated. I fully expect the perturbation to be EXTREMELY LOW, especially if values are being injected non-continuously and at natural pauses in computation like at the top of loops]

V. CONCLUSION

Given a diverse set of motivations for variability studies, and a seemingly intractable problem space when classical performance models are applied to scientific workflows at extreme scale, we argue that a new performance model is required. SOSflow was developed to enable the exploration and validation of new performance models, especially those built for reasoning over high-level and human-understandable semantic annotation that is affixed to all captured data and events.

A. Recommendations

Our research initiative is oriented towards on line and in situ monitoring and analytics. A system should be designed that is efficient enough that it need not be disabled for full-scale production runs of scientific workflows. This creates the important case that the lessons learned through the use of such a monitoring system are in fact applicable to future full-scale production runs, and do not only apply to smaller sample test runs where the workflow's behavior has been heavily perturbed by invasive monitoring technologies. The conclusion goes here. this is more of the conclusion

B. Future Work

ACKNOWLEDGMENT

Research has been conducted under the following grants:
TODO: INSERT GRANT INFORMATION HERE

The authors would like to thank:
NAME of Georgia Tech University
Hasan Abassi of Argonne National Lab
Todd Gamblin of Lawrence Livermore National Lab

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.