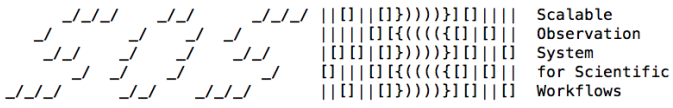
Online Monitoring and Control of HPC Applications using SOSflow

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

***Abstract*—SOSflow provides a flexible, scalable, and programmable framework for observation, introspection, feedback, and control of HPC applications. The Scalable Observation System (SOS) performance model used by SOSflow allows a broad set of online and in situ capabilities including remote method invocation, data analysis, and visualization. SOSflow can couple together multiple sources of data, such as application components and operating environment measures, with multiple software libraries and performance tools, efficiently creating holistic views of performance at runtime. SOSflow can can extend the functionality of HPC applications by connecting and coordinating accessory components, such as in situ visualization tools that are activated only when the primary application is not performing its work. This paper describes the use of SOSflow to provide application-focused monitoring, feedback, and control.**

***Index Terms*—hpc; monitoring; performance; in situ; workflow; sos; sosflow;**



I. Introduction

A. The Scalable Observation System

SOSflow can be used for a variety of purposes:

- Aggregation of application and performance data at runtime
- Providing holistic view of multi-component distributed scientific workflows
- Coordinating in situ operations with global analytics
- Synthesizing application and system metrics with scientific data for deeper performance understanding
- Extending the functionality of existing HPC codes using in situ resources
- Resource management, load balancing, online performance tuning, etc.

There are many more ways to use the SOSflow platform than the scope of this paper can contain. This paper is principally concerned with describing the architecture and use of SOSflow, and concludes with a set of representative experiments. The experiments selected here are abstract, though they were performed on real-world HPC machines. This choice was deliberate, to encourage the reader to imagine how SOSflow could facilitate their own use cases, rather than the reader seeing SOSflow as addressing only a specific case that is not of interest to them.

Fig. 1. An overview of the SOS model.

II. Related Work

Monitoring systems can generally be separated by their purpose into three independent sets: system, application, and science. System-oriented monitoring focuses on machine metrics such as bandwidth, queue depth, resource contention, power, and thermal measurements. Exclusively system-oriented monitoring platforms primarily serve the needs of machine and site administrators, reporting the health and utilization of the nodes, network, and storage systems. Application-oriented monitoring looks at the performance of software components, exploring both single-process performance characteristics as well as behaviors that emerge in parallel distributed contexts. These tools are often of use to low-level application developers and library writers, but traditionally not helpful

to administrators or end users. Science-oriented monitoring involves processing information related to the simulation phenomenon itself. This information can be used for anomaly or convergence detection, visualizations, validation of accuracy, and many other domain-specific ends. While science-related information is important to end-users, it has in the past not been of significant value to developers or administrators. This is no longer true.

REFER TO PAPERS ON PRECISION, POWER vs. PERFORMANCE, REPLICATION, ETC.

SOSflow is designed to address a variety of purposes for monitoring platforms, including those described above. By integrating various layers and categories of information, operating in situ adjacent applications at run time, and supporting online query, analysis, and feedback, SOSflow is able to facilitate both current and future HPC needs. The following sections describe existing and related HPC monitoring efforts, contrasted with SOSflow.

A. System Monitoring

...

B. Application Monitoring

...

C. Science Monitoring

...

To the authors' knowledge, no application-oriented online monitoring systems exist that are feasible for deployment to modern architectures at large scale and can provide runtime interactivity, are programmable, and capable of serving purposes beyond simple migration of performance metrics. SOSflow makes a novel contribution by synthesizing application and system monitoring perspectives into a holistic architecture. SOSflow goes much further than merely interleaving two data sources, as it:

- Operates during the application runtime
- Allows for local and global interactivity
- Couples user-defined analytics with applications
- Automatically archives data for offline analysis
- Provides full-support for complex, loosely-coupled, multi-component scientific workflows, as well as simple single-binary applications
- Enables online query and feedback to both applications and their execution environment
- Is engineered to scale from single-node deployments to future-scale clusters

SOSflow is a fully-programmable system designed to be tailored to individual use-cases with a minimuim amount of additional development or effort.

III. The Architecture of SOSflow

To better understand what can be done with SOSflow, it is useful to examine its architecture. SOSflow is composed of four major components:

- **sosd** : Daemons
- **libsos** : Client Library
- **pub/sql** : Data
- **sosa** : Analytics & Feedback

These components work together to provide extensive runtime capabilities to developers, administrators, and application end-users. All of SOSflow runs in user-space, and does not require elevated priveledges to use any of its features.

A. SOSflow Daemons

Online functionality of SOSflow is enabled by the presence of a user-space daemon. This daemon operates completely independently from any applications, and does not interfere with or utilize any of their resources or communications. The SOSflow daemons are launched from within a job script, before the user's applications are initialized. These daemons discover and communicate amongst each other across node boundaries within a user's allocation. When crossing node boundaries, SOSflow uses the machine's high-speed communication fabric. Inter-node communication may use either **MPI** or **EVPath** as needed, allowing for flexibility when configuring its deployment to various HPC environments.

Fig. 2. SOSflow in situ and off-node daemons.

The traditional deployment of SOSflow will have a single daemon instance running in situ for each node that a user's applications will be executing on. Additional resources can be allocated in support of the SOSflow runtime as-needed to support scaling and to minimize perturbation of application performance. One or more nodes are usually added to the user's allocation to host SOSflow aggregation daemons that combine the information that is being collected from the in situ daemons. These aggregator daemons are useful for providing a holistic unified view at runtime, especially in service to online

analytics modules. Because they have more work to do than the in situ listener daemons, and also are a useful place to host analytics modules, it is advisable to place them on their own dedicated node[s].

1) In Situ: Data coming from SOSflow clients moves into the in situ daemon across a light-weight local socket connection. Any software that connects in to the SOSflow runtime can be thought of as a client. Clients connect only to the daemon that is running on their same node. No socket connections are made across node boundaries, so no special permissions are required to use SOSflow.

The in situ daemon offers the complete functionality of the SOSflow runtime. At startup, it creates a local persistent data store that gathers in all data that is submitted to it. This local data store can be queried and updated via the SOSflow API, with all information moving over the local socket, avoiding any dependence on or contention with the filesystem resources. Providing the full spectrum of data collected on node to clients and analytics modules on node allows for distributed online analytics processing. Analytics modules running in situ can observe a manageable data set, and then exchange small intermediate results amongst themselves in order to compute a final global view. SOSflow also supports running analytics at the aggregation points for direct query and analysis of global or enclave data, though it is potentially less scalable to perform centrally than in a distrubted fashion, depending on the amount of data being processed by the system.

SOSflow's internal data processing utilizes unbounded asynchrous queues for all messaging, aggregation, and data storage. Pervasive design around asynchronous data movement allows for the SOSflow runtime to efficiently handle requests from clients and messaging between off-node daemons without incurring synchronization delays. Asynchronous in situ design allows the SOSflow runtime to scale out beyond the practical limits imposed by globally synchronous data movement patterns.

2) Aggregate: A global perspective on application and system performance is often useful. SOSflow automatically migrates information it is given into one or more aggregation targets. This movement of information is transparent to user's of SOS, requiring no additional work on their part. Aggregation targets are fully-functional instances of the SOSflow daemon, except that their principle data sources are the distributed in situ daemons rather than local clients. The aggregated data contains identical information as the in situ data stores, it just has more of it, and it is assembled into one location. The aggregate daemons are useful for performing online analysis or information visualization that needs to include information from multiple nodes.

SOSflow is not a publish-subscribe system in the traditional sense, but uses a more scalable push and pull model. Everything sent into the system will automatically migrate to aggregation points unless it is explicitly tagged as being node-only. Requests for information from SOSflow are ad hoc and the scope of the request is constrained by the location where the request is serviced: in situ queries are resolved against the in situ database, aggregate queries are resolved against the aggregate database. If the node-only information is potentially useful for offline analysis or archival, the in situ data stores can be collected at the end of a job script, and their contents can be filtered for that node-only information, which can be simply concatenated together with the aggregate database[s] into a complete image of all data All information in SOSflow is tagged with a globally unique identifier (GUID). This allows SOSflow data to be mixed together while preserving its provenance and preventing data duplication or namespace collision.

B. SOSflow Client Library

Clients directly interface with the SOSflow runtime system by calling a library of functions (libsos) through a standardized API. All communication between the SOSflow library and daemon are transparent to users, they do not need to write any socket code or introduce any state or additional complexity to their own code.

TABLE OF INIT/PACK/PUB API FUNCTION SIG.S W/DESCRIPTIONS.

SOSflow forms its own communication channels, and does not interfere with or rely on any communication overlay formed by the application it is linked into. Information sent

Fig. 3. SOSflow client library.

through the libsos API is copied into internal data structures, and can be freed or destroyed by the user after the SOSflow API function returns. Data provided to the API is published up to the in situ daemon with an explicit API call, allowing developers to control the frequency of interactions with the runtime environment. It also allows the user to register callback functions that can be triggered and provided data by

user-defined analytics function, creating an end-to-end system for both monitoring as well as feedback and control.

To maximize compatibility with extant HPC applications, the SOSflow client library is presently implemented in C99. The use of C99 allows the library to be linked in with a wide variety of HPC application codes, performance tools, and operating environments. There are various custom object types employed by the SOSflow API, and these custom types can add a layer of complexity when binding the full API to a language other than C or C++. SOSflow provides a solution to this challenge by offering a "Simple SOS" (ssos) wrapper around the full client library, exposing an API that uses no custom types. The ssos wrapper was used to build a native Python module for SOSflow. Users can directly interact with the SOSflow runtime environment from within Python scripts, acting both as a source for data, but also a consumer of online query results. SOSflow users can extend the utility of their applications using **numpy** for analytics and **pyplot** for information visualization, as a simple example. HPC developers can capitalize on the ease of development provided by Python, using SOSflow to observe and react to online data from complex applications written in legacy languages and data models.

C. SOSflow Data

The primary concept around which SOSflow organizes information is the "publication handle" (pub). Pubs provide a private namespace where many types and quantities of information can be stored as a key/value pair. SOSflow automatically annotates values with a variety of metadata, including a GUID, timestamps, origin application, node id, etc. This metadata is available in the persistent data store for online query and analysis. SOSflow's metadata is useful for a variety of purposes:

- Performance analysis
- Provenance of captured values for detection of source-specific patterns of behavior, failing hardware, etc.
- Interpolating values contributed from multiple source applications or nodes
- Re-examining data after it has been gathered, but organizing the data by metrics other than those originally used when it was gathered

The full history of a value, including updates to that value, is maintained in the daemon's persistent data store. When a key is re-used to store some new information that has not yet been transmitted to the in situ daemon, the client library enqueues it up as a snapshot of that value, preserving all associated metadata alongside the historical value. The next time the client publishes to the daemon, the current and all enqueued historical values are transmitted. SOSflow is built on a model of a global information space. Aggregate data stores are guaranteed to provide eventual consistency with the data stores of the in situ daemons that are targeting them. SOSflow's use of continuous but asynchronous movement of information through the runtime system does not allow for strict quality-of-service guarantees about the timeliness of information being available for analysis. A parametric scaling study is included in this paper that demonstrates the performance that can reasonably be expected on a real-world HPC cluster.

Fig. 4. SOSflow data store.

SOSflow does not require the use of a domain-specific language when pushing values into its API. Pubs are self-defining through use: When a new key is used to pack a value into a pub, the schema is automatically updated to reflect the name and the type of that value. When the schema of a pub changes, the changes are automatically announced to the in situ daemon the next time the client publishes data to it. Once processed and injected into the persistent data store, values and their metadata are accessible via standardized SQL queries. SOSflow's online resolution of SQL queries provides a high-degree of programmability and online adaptivity to users. Examples and further documentation of SOSflow's SQL schema are provided within the standard SOSflow repository.

D. SOSflow Analytics & Feedback

INSERT SOURCE CODE EXAMPLE W/CALLBACK REGISTRATION/TRIGGERING

IV. Experiments

A. Evaluation Platform

1) Cori:

2) Titan:

B. Experiment Setup

1) Scaling Study:

- Message Size: 1, 64, 1024, 16384, 131072
- Message Size (or): 1, 10, 100, 1000, 10000, 100000

Fig. 5. SOSflow analytics interface.

- Message Frequency: 10 seconds, 1 Hz, 10 Hz, 100 Hz, 1k Hz, 10k Hz
- Application Instances per Listener: 1, 4, 8, 16, 32

2) Load Balance Proxy:

3) Global Power Cap:

4) Producer-Consumer "workflow":

V. Results

A. Observations

B. Discussion

1) Limitations of SOSflow:

2) Strengths of SOSflow:

VI. Conclusion

A. Future Work

B. Acknowledgments

The research report was supported by a grant (DE-SC0012381) from the Department of Energy, Scientific Data Management, Analytics, and Visualization (SDMAV), for "Performance Understanding and Analysis for Exascale Data Management Workflows."

References

[1] 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 *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools*, pages 42–49. IEEE Press, 2016.