SOSflow: A Scalable Observation System for Introspection and In Situ Analytics

Lawrence Livermore National Laboratory

UNIVERSITY OF OREGON

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."
Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-751722).

Efficiently observing and interacting with complex scientific workflows at scale presents unique challenges.

SOSflow helps meet them.

- Distributed components with ***data flow***
- Complex interactions with ***dynamic behavior***
- **Variability is inherent** in machine components
- Shared resource utilization at runtime is a significant factor in performance
- Offline measurements provide limited insights
- An **in situ (online) runtime** is needed for scalable measurement, analysis, and application steering

Design and API

- SOSflow written in C99 for high-performance w/small footprint
- Several communication backends are supported, including EVPath, MPI, sockets, and ZeroMQ
- Asynchronous design focuses on minimizing overhead and time spent in API calls within client applications
- Flexible, programmable interface
- Provides a distributed key/value store with full SQL query support
- Offers a low-latency value cache with adjustable depth
- Highly-configurable daemons
- Integrated support for UID/GID authentication (Munge)

"Hello, SOS" w/C:

```c
#include "sos.h"

int main(int argc, char **argv) {

    // Initialize the client, registering it with the SOS runtime.
    // In an MPI application, this is usually called immediately
    // after the MPI_Init(...) call.
    SOS_runtime *sos = NULL;
    SOS_init(&argc, &argv, &sos,
        SOS_ROLE_CLIENT, SOS_RECIEVES_NO_FEEDBACK, NULL);

    SOS_pub *pub = NULL;
    SOS_pub_create(sos, &pub, "demo",
        SOS_NATURE_CREATE_OUTPUT);

    int someInteger = 256;
    SOS_pack(pub, "examplevalue", SOS_VAL_TYPE_INT, &someInteger);

    SOS_announce(pub);
    SOS_publish(pub);

    for (someInteger = 1024; someInteger <= 2048; someInteger++) {
        // All these pack'ed values will accumulate within the
        // client until the next SOS_publish(...) is called on the
        // publication handle.
        SOS_pack(pub, "examplevalue",
            SOS_VAL_TYPE_INT, &someInteger);
    }

    SOS_publish(pub);

    // This is called at the end of an application, when it will no
    // longer be contributing to the SOS environment or responding
    // to feedback directives from SOS.
    // In an MPI application, the client usually will call this
    // immediately before the call to MPI_finalize().
    SOS_finalize(sos);

    return 0;
}
```

Online query w/Python:

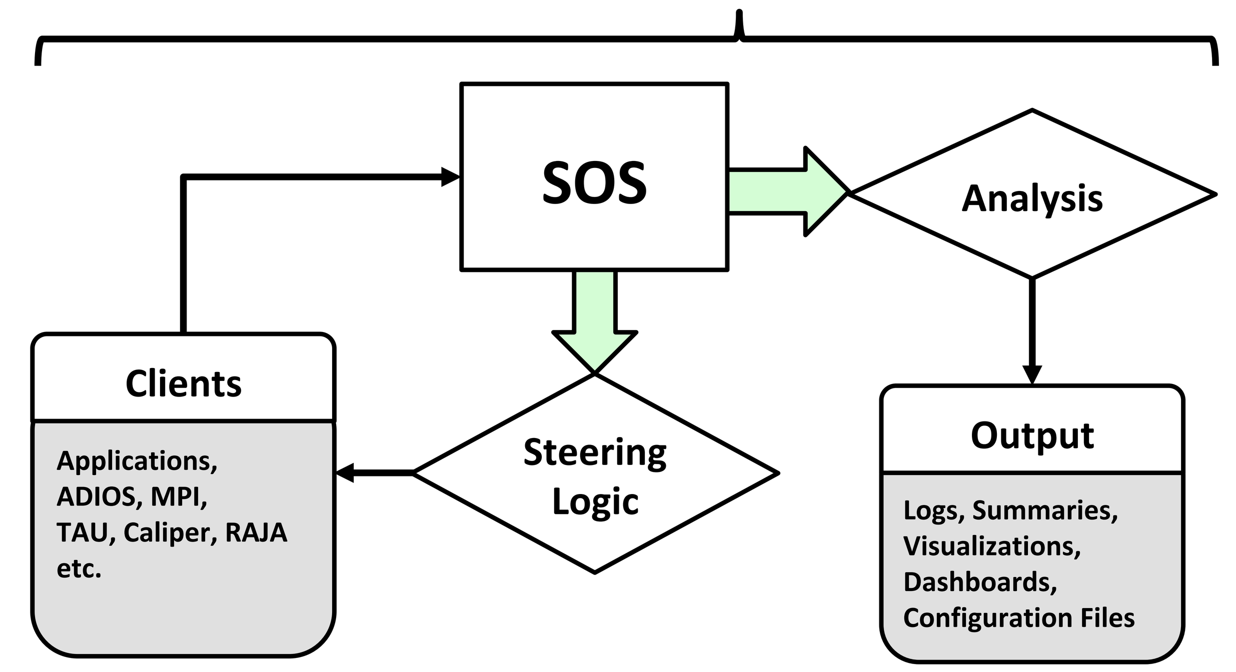
```python
#!/usr/bin/env python

import os
from ssos import SSOS

def demonstrateSOS():
    SOS = SSOS()

    sos_host = "localhost"
    sos_port = os.environ.get("SOS_CMD_PORT")

    SOS.init()
    SOS.pack("somevar", SOS.STRING, "Hello, SOS.  I'm a python!")
    SOS.announce()
    SOS.publish()

    sql_string = "SELECT * FROM tblVals LIMIT 10000;"

    results, col_names = SOS.query(sql_string, sos_host, sos_port)

    print "Results:"
    print "    Output rows....: " + str(len(results))
    print "    Output values..: " + str(results)
    print "    Column count...: " + str(len(col_names))
    print "    Column names...: " + str(col_names)
    print ""

    SOS.finalize();


if __name__ == "__main__":
    demonstrateSOS()
```

Download SOS:
github.com/cdwdirect/sos_flow.git

SOS Model

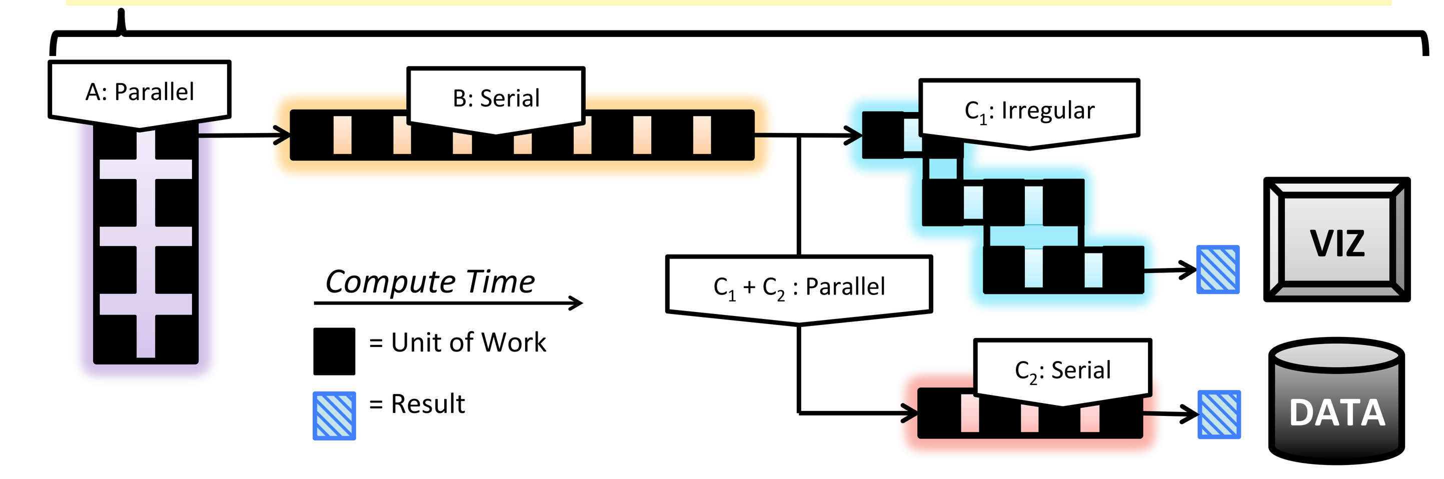
SOSflow functions as a hub for collecting, aggregating, and acting on a variety of information at runtime

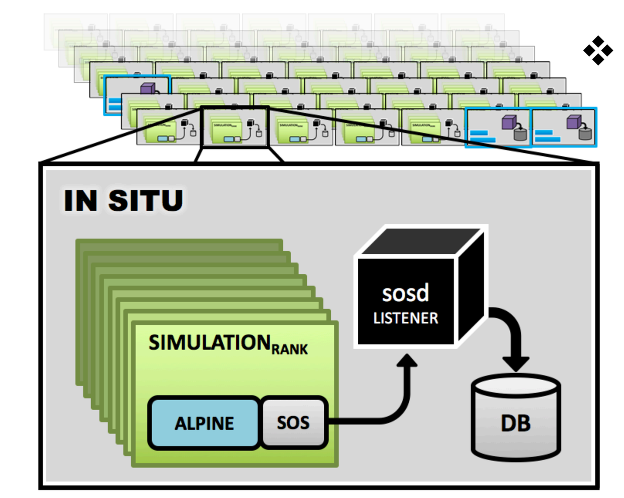
SOSflow's in situ (online) services work together to provide global views and online data analytics within an HPC environment



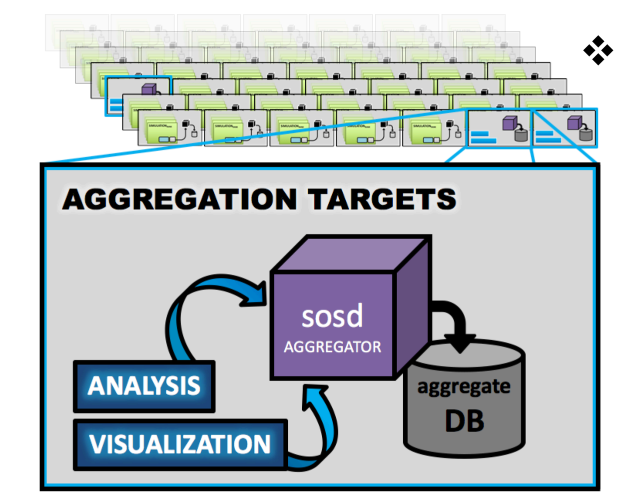
SOS Daemons

SOSflow daemons provide **an integrated context for information** from all components of a distributed workflow, for the entire duration of a job.



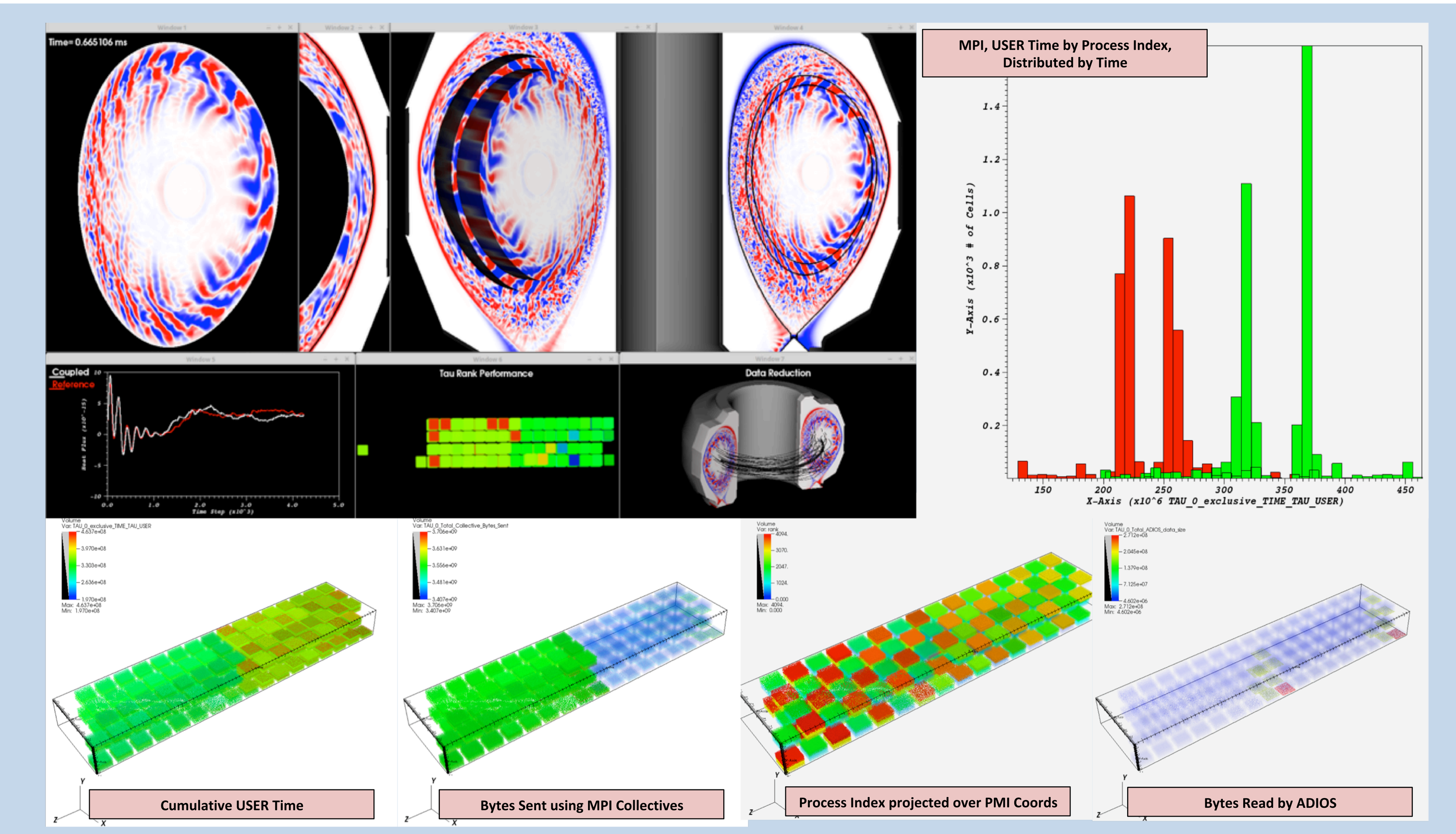


- **In Situ Listeners**
  - Runs entirely in user-space
  - Minimally invasive
  - In-memory SQL database per daemon
  - Efficient push/pull data flow
  - Provides feedback/control mechanism for analysis and steering scripts
  - Integration with performance tools
  - APIs for C/C++ and Python

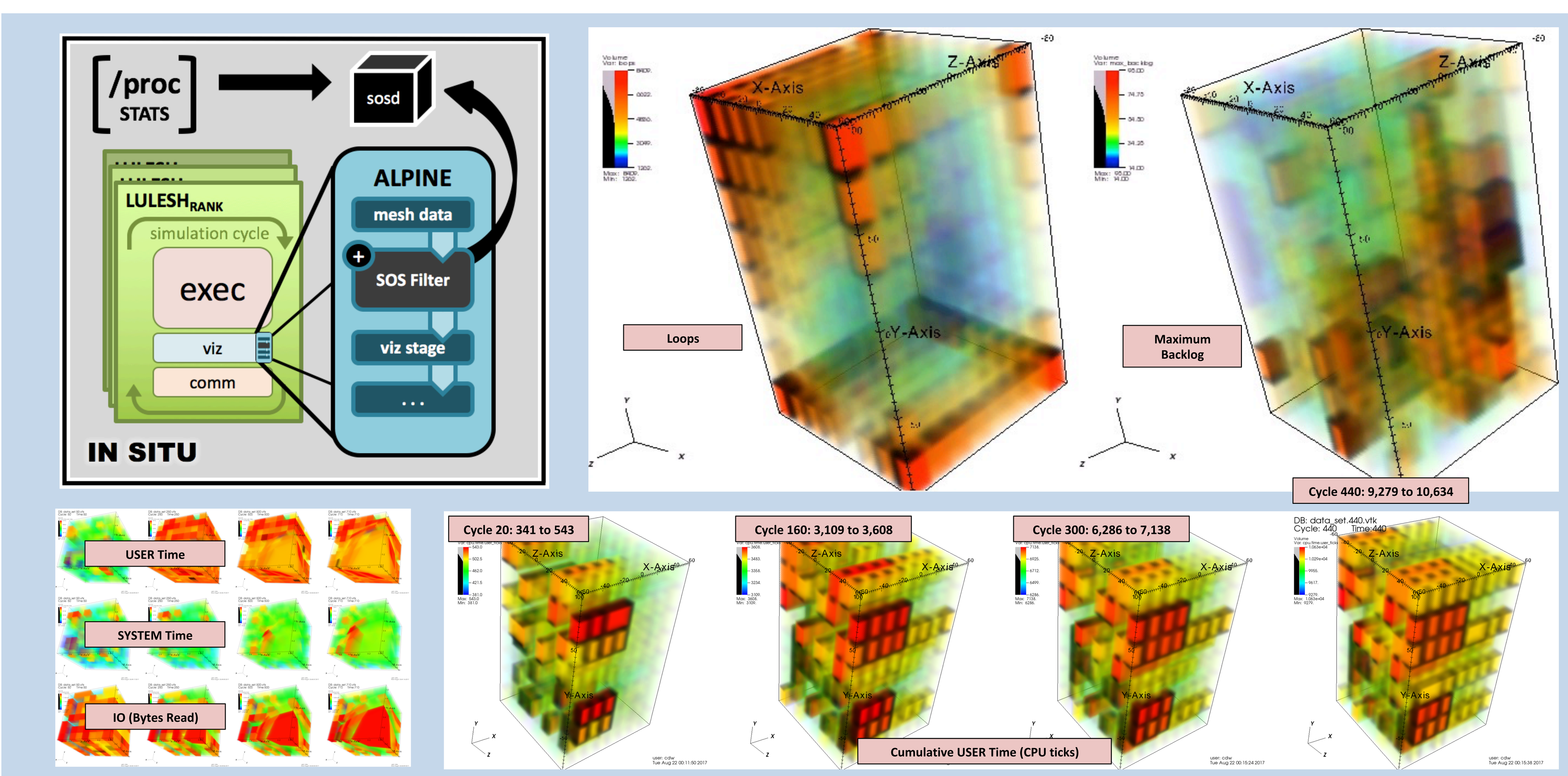


- **Off-Node Aggregators**
  - Can be run on dedicated nodes
  - SQL store contains all the data captured by the listeners
  - Can launch many aggregators and run queries on them in parallel
  - Send feedback/control data to listeners
  - Online aggregation can be disabled or deferred to offline processing

Results: Performance Understanding



- 4,096 ranks of XGC on TITAN
- Data collected and aggregated online from TAU measuring ADIOS, MPI, and user code
- Python script queried SOSflow during the run and assembled VTK files with performance metrics projected over server rack and node coordinates
- SOSflow integrated performance measurements from all parts of the workflow
- Dynamic visualizations were rendered and displayed live during the run
- Any TAU-collected performance metrics could be selected for display



- 512 ranks on 32 nodes on QUARTZ and CATALYST
- SOSflow filter added to ALPINE Ascent pipeline
- KRIPKE: 3D deterministic neutron transport proxy application that implements a distributed-memory parallel sweep solver over a rectilinear mesh.
- LULESH: 3D Lagrangian shock hydrodynamics proxy application that models Sedov blast test problem over a curvilinear mesh.
- No ad hoc instrumentation needed
- Updated geometry is automatically captured during the run to observe metrics projected over a changing mesh
- Anything published to SOSflow can be projected into these online views
- SOS runtime overhead within system noise
- Enable/disable without recompilation

Future Work

Apollo Performance Portability

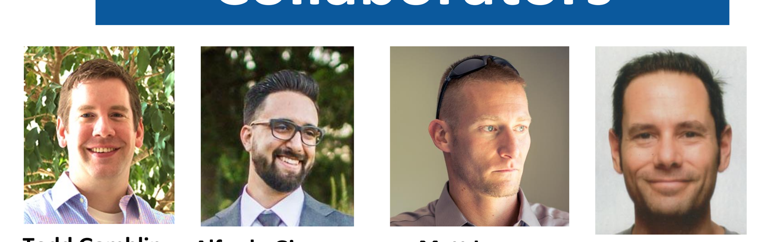
- Next Generation of LLNL's Apollo Project
- Intelligent RAJA policy configuration
- Caliper and SOSflow collect metrics at runtime and facilitate distributed analysis and steering
- Online machine learning adapts to changes over time
  i. Physics changes over time in a run
  ii. Code changes w/new commits and merges
  iii. System utilization changes during jobs

Author

Chad Wood
cdw@cs.uoregon.edu
ix.cs.uoregon.edu/~cdw

Chad Wood is a fourth-year Computer & Information Science PhD student at the University of Oregon. His research focus is on monitoring, introspection, feedback, and control for HPC systems, emphasizing online in situ operations and scalability.

Collaborators



Todd Gamblin
Alfredo Gimenez
Matt Larsen
Kevin Huck

David Boehme
David Beckingsale
David Poliakoff