Prescriptive provenance for streaming analysis of workflows at scale

Line Pouchard, Li Tang, Huub Van Dam, Shinjae Yoo - CSI

Dingwen Tao - CMU

Kevin Huck - University of Oregon

BROOKHAVEN NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

Definition of provenance

Provenance in Computer Science is defined as the record of data lineage and software processes operating on this data that enable interpreting, validating and reproducing results. In experimental science, provenance also includes experimental conditions, calibrations, notebooks, etc.

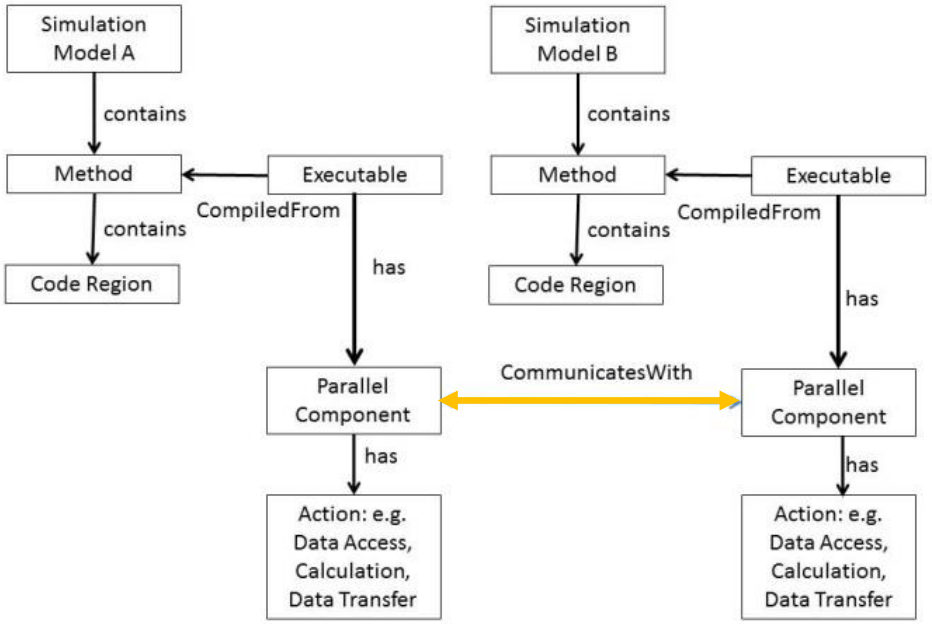
```
:bar_chart
   a prov:Entity;
   prov:wasGeneratedBy   :illustrationActivity;
   prov:wasDerivedFrom   :aggregatedByRegions;
   prov:wasAttributedTo :derek;

:graduation
   a prov:Activity, :Graduation;
   prov:startedAtTime "2012-04-15T13:00:00-04:00"^^xsd:dateTime;
   prov:used           :ms_smith;
   prov:generated      :doctor_smith;
   prov:endedAtTime    “2012-04-15T14:30:00-04:00"^^xsd:dateTime;
```

Uses of Provenance in HPC

- **Performance Investigation and optimization**
  - Where are the performance anomalies coming from?
    - Which workflow component, node, function?
  - Can I quickly find the environmental conditions where anomalies are detected?
  - Has a similar error been encountered before?
    - In previous runs?
  - Have the HW and SW stack on this system changed since my last run?
- **Reproducibility**
  - What version of the application/workflow did this run use?
  - Which libraries did I use previously?
  - What configuration switches did I use?
  - What results do I obtain if I run the same workflow again?
    - What happens to performance if I run the same configuration on another system?
    - Where are the trade-offs between accuracy and performance?
    - Can I establish thresholds of reproducibility?

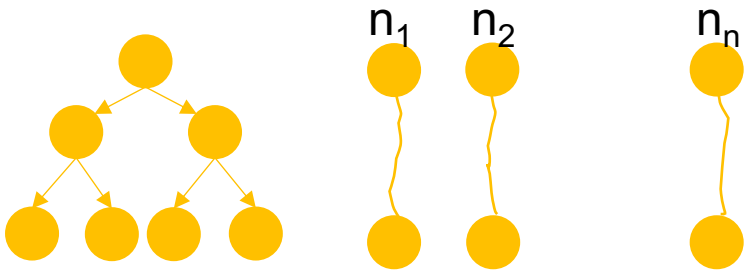
What is a workflow?



Interdependencies between parallel components at runtime

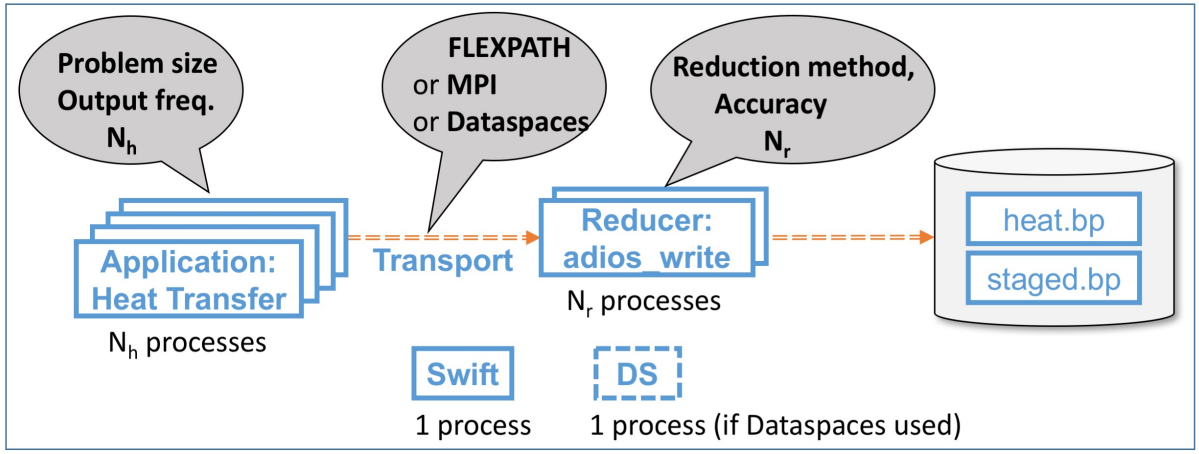
Performance of each individual component can affect overall performance of a workflow

Performance metrics are measurements from workflow and system resources



Courtesy: Kleese van Dam, K., et al., Enabling Structured Exploration of Workflow Performance Variability in Extreme-Scale Environments, Proc. 8th Workshop in Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS) collocated with SC 2015.

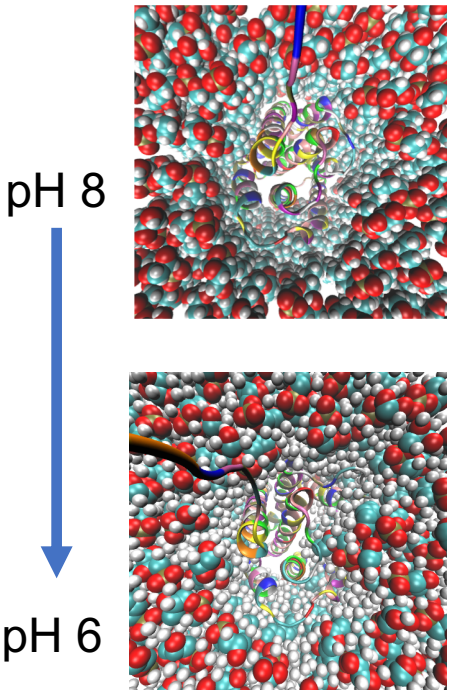
Basic example and mini-app: the heat transfer equation

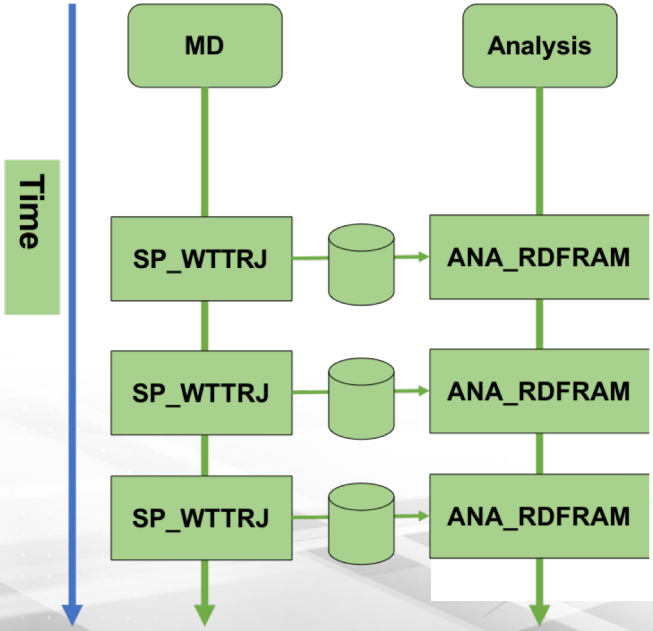


ADIOS

Use Case: NWChem MD

- MD simulates
  - Conformational changes of biomolecules
  - Transport processes
  - Try to find statistics and key points
- Processes involve
  - Many atoms (~1 million)
  - Long time scales (μs)
  - Many time steps (~1 billion)
- Shown
  - Transmembrane Calcium channel
  - Regulated by pH
  - Plays a role in plant draught response





Challenges

- Volume of performance data is too large manual inspection
- Provenance is very verbose, hard to extract and understand
- Memory is limited – not all data points can be saved
- Streaming methods for anomaly detection are needed

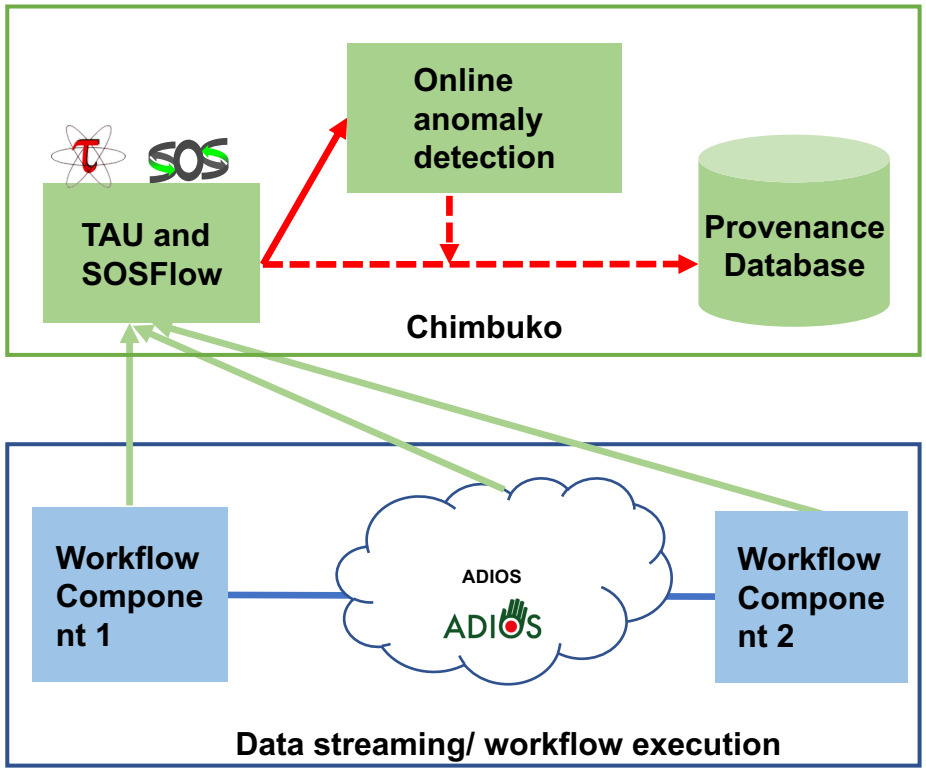
Prescriptive Provenance

Events (anomalies) and a time series rolling window of preceding events are extracted and stored

The provenance of these events selected for retention by the anomaly detection and retained for a time window preceding the occurrence of the events

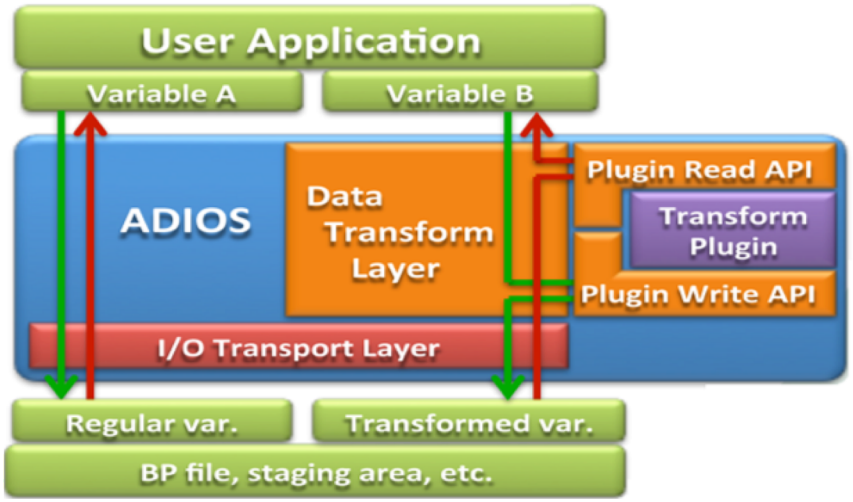
Chimbuko overview architecture

- Chimbuko monitors workflow execution, extracts provenance and visualize data
- ADIOS orchestrates workflows (blue line) and provides data streaming
- TAU provides performance metrics for instrumented components 1 & 2
- TAU extracts provenance metadata and trace data (green lines)
- SOSFlow stores and aggregates the data at each node
- Trace data is dynamically analyzed to detect anomalies (solid red line)
- Selected metadata and trace data is stored (e.g., time window for which trace event is interesting) (dashed red lines)



The Adaptable IO System (ADIOS)

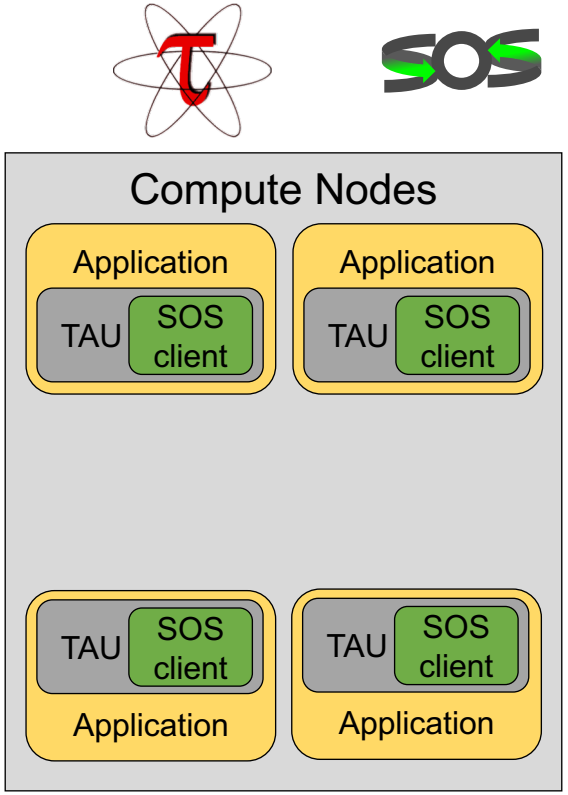
- Provides portable, fast, scalable, easy-to-use, metadata rich output
- Key feature: I/O abstraction
  - Optimized high-performance write and read
  - Data transformation: indexing and compression
  - Data connection via staging, WAN, etc.



Courtesy of Jong Choi, Workflow Case Studies, EPSi Data Processing and Experimental Data Processing Workflows

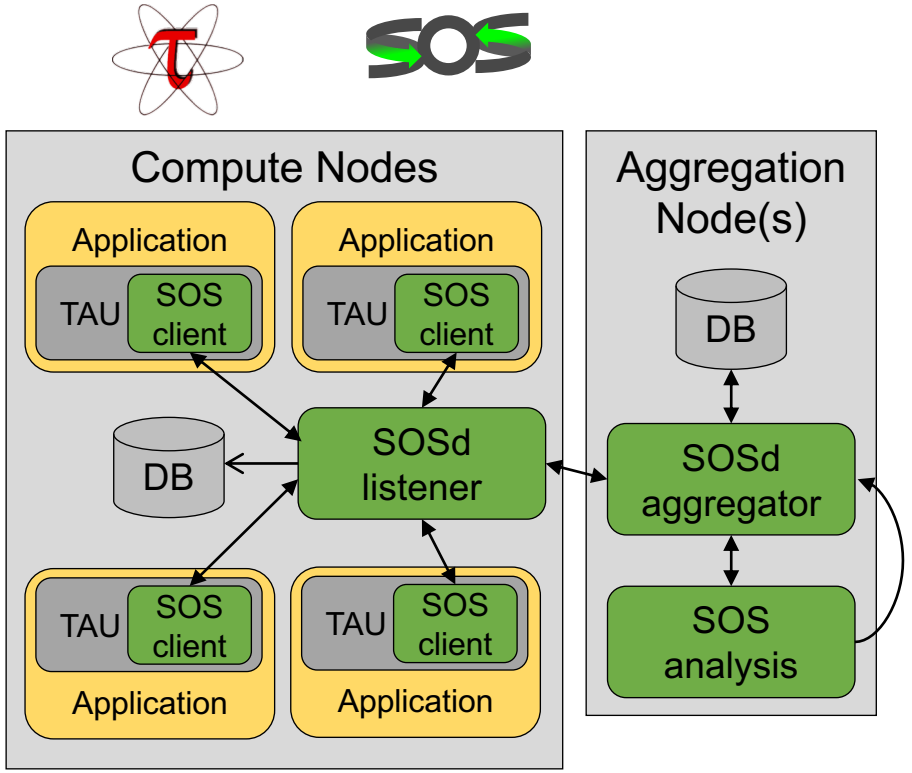
TAU performance system

- TAU Performance System® measures instrumented application regions, library calls
  - OpenMP regions, MPI functions, I/O libraries (POSIX, ADIOS)
  - Time and hardware counters of interest (cache misses, FLOPs)
  - Metadata from the execution



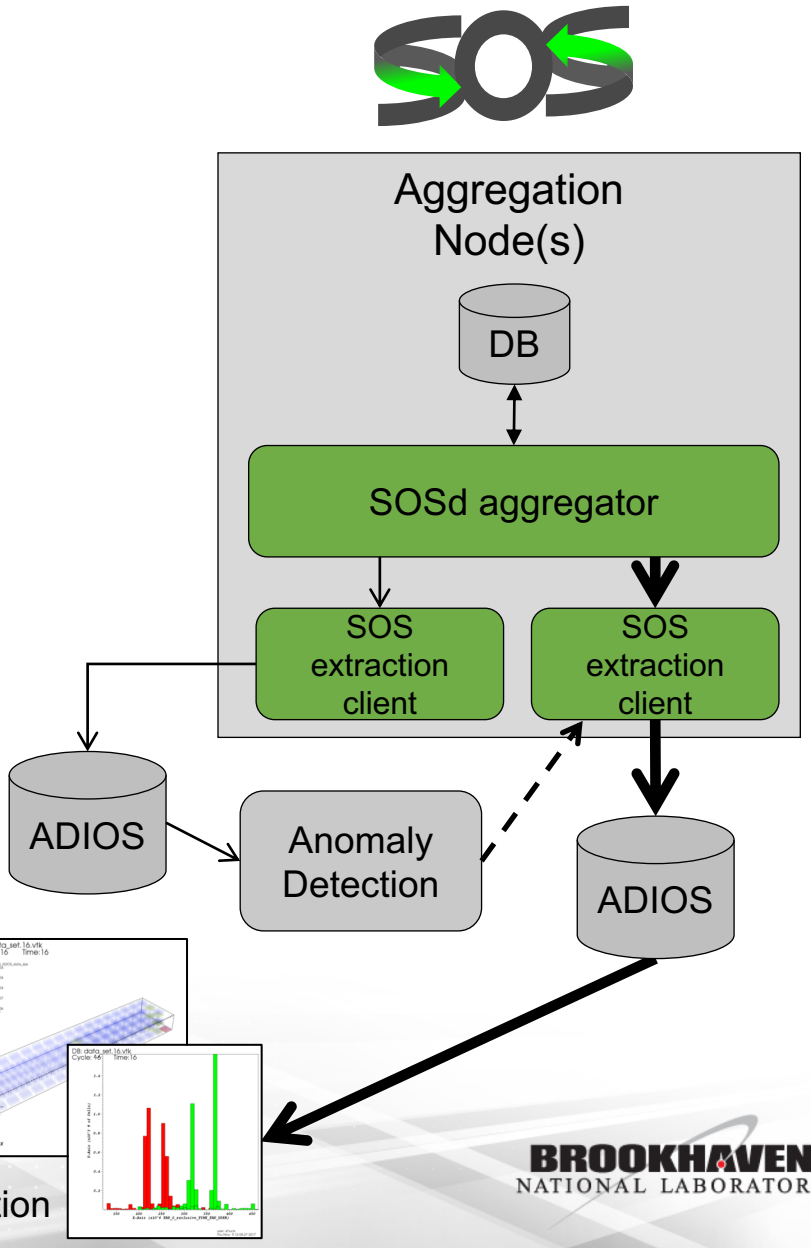
SOSFlow aggregation

- SOS client plugins, listeners and aggregators periodically aggregate performance data from all applications
  - Maintains cache of *N* most recent frames of data
  - Optional storage to key/value store database



SOSFlow and Anomaly Detection

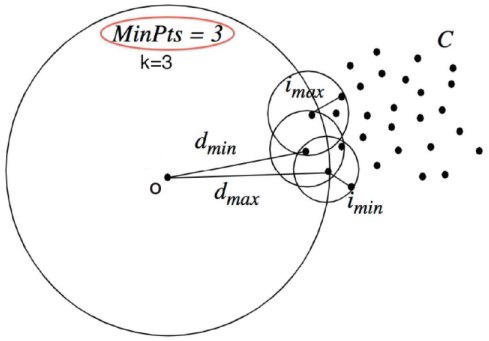
- SOS extraction client periodically reads trace data and exports it over ADIOS to be read by anomaly detection analysis
- If/when anomaly detected, a second extraction client is launched to export full window of trace data leading up to anomalous event



Prescriptive Provenance Window

- We keep all the (static) provenance metadata for each run
- We keep all the performance metrics for a window preceding the anomalies
- Prescriptive provenance is the provenance of events selected for retention by anomaly detection
  - Anomaly detection prescribes the events to be stored.

Offline Anomaly Detection: Local Outlier Factor (LOF)



- Reachability distance from o' to o:

$$reachdist_k(o \leftarrow o') = \max\{dist_k(o), dist(o, o')\}$$

  - where k is a user-specified parameter

- Local reachability density of o:

$$lrd_k(o) = \frac{\|N_k(o)\|}{\sum_{o' \in N_k(o)} reachdist_k(o' \leftarrow o)}$$

k

$dist_k(o)$: the distance between data point o and its kth nearest neighbor (kth-NN)

$N_k(o)$: the set of k nearest neighbors of data point o

- LOF (Local outlier factor) of an object o is the *average of the ratio of local reachability of o and those of o's k-nearest neighbors*

$$LOF_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{lrd_k(o')}{lrd_k(o)}}{\|N_k(o)\|} = \sum_{o' \in N_k(o)} lrd_k(o') \cdot \sum_{o' \in N_k(o)} reachdist_k(o' \leftarrow o)$$

- The lower the local reachability density of o, and the higher the local reachability density of the kNN of o, the higher LOF
- This captures **a local outlier** *whose local density is relatively low comparing to the local densities of its kNN*

Higher LOF value ➔ Higher probability to be outlier

Online Anomaly Detection: Incremental Local Outlier Factor (iLOF)

- LOF: Requires the entire data set to compute LOF values
- iLOF: Can compute the outlier factor for each incoming data point (can be used for data streams)
- For each incoming data point *p,* perform an iLOF insertion:
  - Find *p*'s *k-nearest neighbors* (*k-NNs*)
  - Compute LOF of *p* based on the outlier factors of its *k-NNs*
  - Update the k-NNs of past data points along with their LOFs (if needed)

Algorithm 1. iLOF Insertion

1. **Input:** a data point p_t at time t
2. **Output:** LOF value $LOF(p_t)$
3. Compute $N_{(p_t,k)}$ and $k-distance(p_t)$
4. **for all** $o \in N_{(p_t,k)}$ **do**
5. Compute $reach-dist(p_t, o)$ using Equation (1)
6. **end for**
7. $S_{update} \leftarrow RN_{(p_t,k)}$ {the set of reverse k-NNs of p_t}
8. **for all** $o \in S_{update}$ and $q \in N_{(o,k)}$ **do**
9. Update $k-distance(o)$ and $reach-dist(q, o)$
10. **if** $o \in N_{(q,k)}$ **then**
11. $S_{update} \leftarrow S_{update} \cup \{q\}$
12. **end if**
13. **end for**
14. **for all** $o \in S_{update}$ **do**
15. Update $lrd(o)$ and $LOF(\{RN_{(o,k)}\})$
16. **end for**
17. Compute $lrd(p_t)$ and $LOF(p_t)$
18. **return** LOF

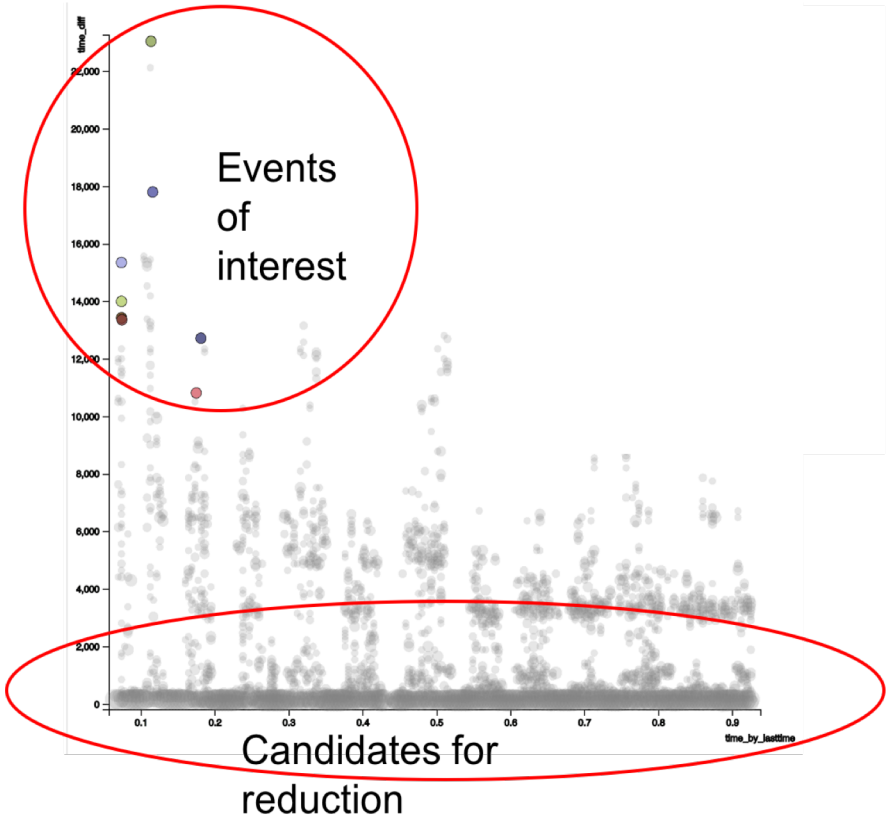
However, iLOF still suffers from large memory requirements, since all past data points need to be retained to compute the outlier factor for each new incoming data point (for high accurate of outlier factor value)

Online Anomaly Detection: Limited Memory incremental LOF (MiLOF)

1. Summarization
   - After incoming points reaches memory limit
     - Build a summary over the past data points (i.e., k-distance, lrd, LOF) based on K-means
     - Delete these data points from the memory
   - Not necessarily to find clusters exactly, but aims to find similar data points to represent average of original data points
2. Merging
   - After summarization, new generated cluster centers will be merged with existing cluster centers by using *weighted clustering* algorithm
   - Weights are based on their numbers of points before summarization
3. Updating
   - Update k-distance, lrd, LOF based on merged clusters

MILOF: Fast Memory Efficient Local Outlier Detection in Data Streams, Mahsa Salehi, Christopher Leckie, James C. Bezdek, Tharshan Vaithianathan, and Xuyun Zhang. *IEEE Transactions on Knowledge and Data Engineering*, 28:12, 2016

Data reduction with provenance



- Data set: LAMMPS data subset
- Event: function: voidLAMMPS_NS::Run::command(int,char **):voidLAMMPS_NS::Verlet::run(int):MPI_Wait()
- Parameters
  - Anomalies rate: 1%
  - No. of nearest neighbors: 4
- Memory limit size: 128
- Summarization size: 64

Extracted Provenance

What does it mean?

What do we keep?

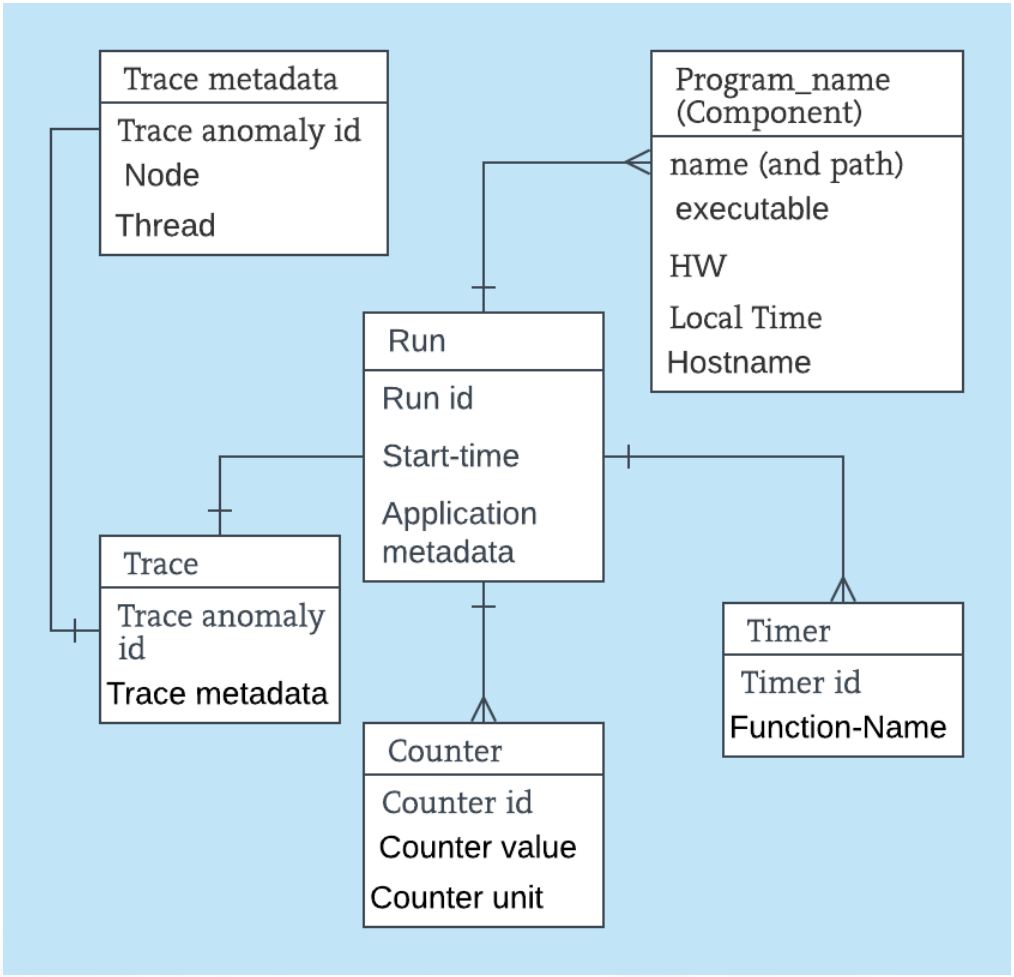
```
[1 1 1 0 0 nan nan nan nan nan nan 1529097196656293 3]
[1 1 1 0 1 nan nan nan nan nan nan 1529097196656433 4]
[1 1 1 0 2 nan nan nan nan nan nan 1529097196656630 5]
[1 3 1 0 0 nan nan nan nan nan nan 1529097196660985 6]
[1 3 1 0 1 nan nan nan nan nan nan 1529097196661109 7]
[1 3 1 0 2 nan nan nan nan nan nan 1529097196661243 8]
[1 2 1 0 0 nan nan nan nan nan nan 1529097196663663 9]
[1 2 1 0 1 nan nan nan nan nan nan 1529097196663759 10]
[1 2 1 0 2 nan nan nan nan nan nan 1529097196663882 11]
[1 0 1 1 2 nan nan nan nan nan nan 1529097196694382 12]
[1 3 1 1 2 nan nan nan nan nan nan 1529097196694490 13]
[1 0 0 nan nan 0 91802 nan nan nan nan 1529097196694678 14]
[1 0 0 nan nan 1 16484 nan nan nan nan 1529097196694812 15]
[1 3 0 nan nan 0 91802 nan nan nan nan 1529097196694826 16]
[1 1 1 1 2 nan nan nan nan nan nan 1529097196694855 17]
[1 2 1 1 2 nan nan nan nan nan nan 1529097196694878 18]
[1 0 0 nan nan 2 16484 nan nan nan nan 1529097196694891 19]
[1 3 0 nan nan 1 16396 nan nan nan nan 1529097196694972 20]
[1 0 0 nan nan 3 0 nan nan nan nan 1529097196695028 21]
[1 3 0 nan nan 2 16396 nan nan nan nan 1529097196695040 22]
[1 3 0 nan nan 3 0 nan nan nan nan 1529097196695157 23]
[1 1 0 nan nan 0 91804 nan nan nan nan 1529097196695262 24]
[1 2 0 nan nan 0 91804 nan nan nan nan 1529097196695389 25]
[1 1 0 nan nan 1 16508 nan nan nan nan 1529097196695411 26]
[1 1 0 nan nan 2 16508 nan nan nan nan 1529097196695491 27]
```

```
**** Print info ****
>>> Number of attributes = 506
>>>506 names and values of attributes =
program_name 0     b'/home/khuck/src/Example-Heat_Transfer/stage_write/st
MetaData:0:0:0:CPU Cores     b'4'
MetaData:0:0:0:CPU MHz     b'2000.000'
MetaData:0:0:0:CPU Type     b'Intel(R) Xeon(R) CPU X5355 @ 2.66GHz'
MetaData:0:0:0:CPU Vendor     b'GenuineIntel'
MetaData:0:0:0:CWD     b'/home/khuck/src/Example-Heat_Transfer/test_sos'
MetaData:0:0:0:Cache Size     b'4096 KB'
MetaData:0:0:0:Command Line     b'../stage_write/stage_write heat.bp staged.
MetaData:0:0:0:Executable
b'/home/khuck/src/Example-Heat_Transfer/stage_write/stage_write'
MetaData:0:0:0:Hostname     b'ktau'
MetaData:0:0:0:Local Time     b'2018-06-15T14:13:17-07:00'
MetaData:0:0:0:Memory Size     b'8172400 kB'
MetaData:0:0:0:Node Name     b'ktau'
MetaData:0:0:0:OS Machine     b'x86_64'
MetaData:0:0:0:OS Name     b'Linux'
MetaData:0:0:0:OS Release     b'4.4.0-127-generic'
MetaData:0:0:0:OS Version     b'#153-Ubuntu SMP Sat May 19 10:58:46 UTC 2018'
MetaData:0:0:0:Starting Timestamp     b'1529097197678517'
MetaData:0:0:0:TAU Architecture     b'default'
MetaData:0:0:0:TAU Config     b' -iowrapper -pdt=/home/khuck/install/pdtoolkit-3.25
-papi=/usr/local/papi/5.5.0 -sos=/home/khuck/install/sos_flow -mpi
```

Where are the anomalies?

```
>>> Indices of anomalies in terms of entry: [348, 450, 691, 1086,
1343, 1605, 1867, 1974, 2397, 2777]
```

Provenance database



Querying provenance

What are the anomalies for this run?

For a given anomaly in a run, what is:

- the workflow component?
- the node, thread, function call?
- the time elapsed since start of simulation?

What are the trace data for this node/component/function before the anomaly?

- we save all trace data in memory preceding the detected anomaly
- the window is constrained by the total amount of memory
  - in the system
  - at each node

Conclusion

The extraction of provenance and performance metrics at a detailed level for a simulation run

The use of provenance for code optimization and debugging

The provenance system complements visualization of the performance trace

Publications

L. Pouchard, A. Malik, H. Van Dam, K. Kleese, C. Xie, and W. Xu, Capturing provenance as a diagnostic tool for workflow performance evaluation and optimization, NYSDS 2017

I. Foster, et al., Computing just what you need: Online data analysis and reduction at extreme scales, Europar 2017

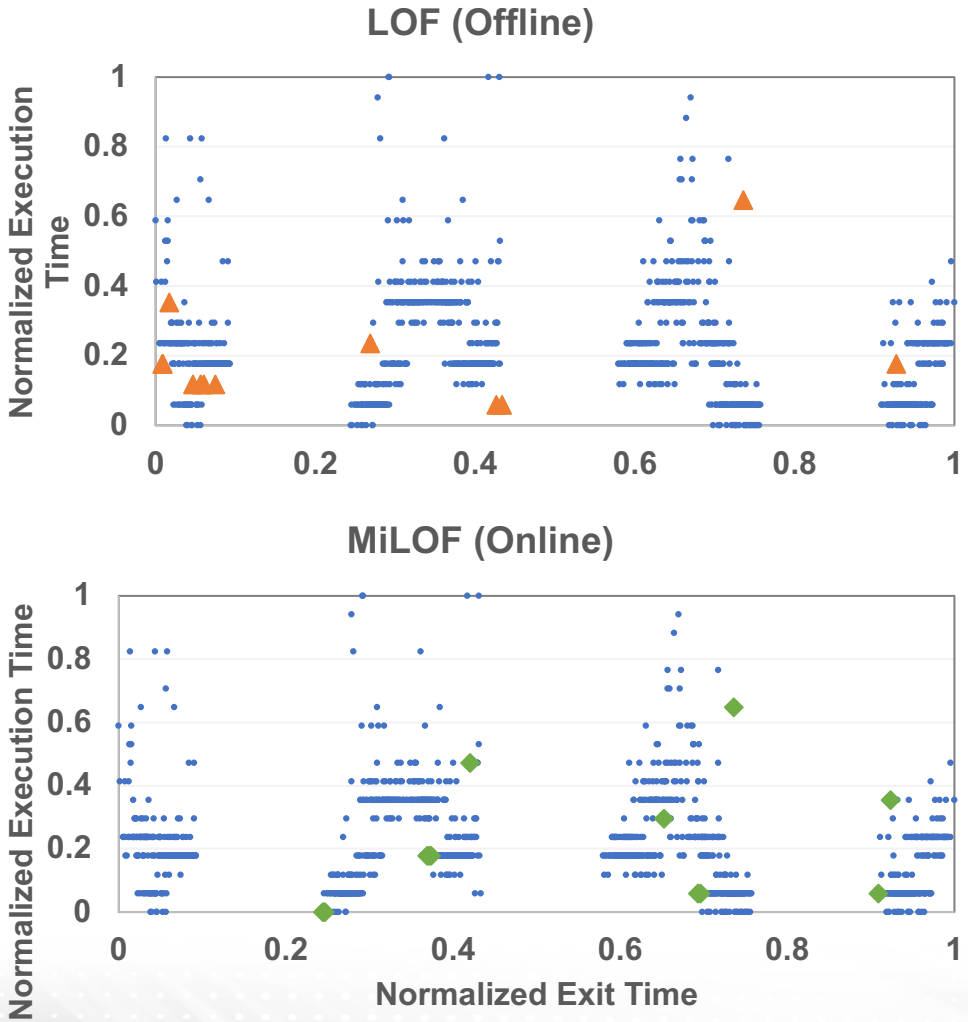
L. Pouchard, S. Baldwin, T. Elsethaggen, C. Gamboa, S. Jha, B. Raju, E. Stephan, L. Tang, and K. Van Dam, Computational reproducibility for scientific workflows in large-scale environments, SC 2017 (presentation) and IJHPCA (submitted)

Acknowledgements

This research was supported by the Exascale Computing Project (ECP), a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration. The Project Number for the **Co-design center for Online Data Analysis and Reduction (CODAR)** that supported this research is 17-SC-20-SC.

We acknowledge the participation of Mohammed Endris, Behiru Shita, and Dr. Mulugeta Dubda, Morgan State University and NSF Summer Program.

Anomaly Detection Results on LAMMPS Data



- Data set: LAMMPS data subset
- Data size: 1280 2D data points
- Event: function: voidLAMMPS_NS::Run::command(int,char**):voidLAMMPS_NS::Verlet::run(int):MPI_Wait()
- Parameters
  - Anomalies rate: 1%
  - No. of nearest neighbors: 4
- Memory limit size: 128
- Summarization size: 64
- Window size: 1280