Prescriptive provenance for streaming analysis of workflows at scale

Line Pouchard
Center for Data Driven Discovery
Brookhaven National Lab
Upton, NY
pouchard@bnl.gov

Kevin Huck
OACISS
University of Oregon
Eugene, OR
khuck@cs.uoregon.edu

Gyorgy Matyasfalvi
Computer Science and Math
Brookahven National Lab
Upton, NY
gmatyasfalv @bnl.gov

Dingwen Tao
Computer Science Department
University of Alabama
Tuscaloosa, AL
tao@cs.ua.edu

Li Tang
Computer Science and Math
Brookhaven National Lab
Upton, NY
ltang@bnl.gov

Huub Van Dam
Condensed Matter Physics
Brookhaven National Lab
Upton, NY
hvandam@bnl.gov

Shinaje Yoo
Computer Science and Math
Brookahven National Lab
Upton, NY
sjyoo@bnl.gov

***Abstract*— We extend our approach capturing and relating the provenance and performance metrics of computational workflows as a diagnostic tool for runtime optimization and placement. One important challenge is the volume of extracted data, both for performance metrics and provenance, even when specifying filters and focusing on quantities of interest in a simulation. We reduce this data by performing anomaly detection on streaming data and store provenance for the detected anomalies, an approach we call *prescriptive provenance.* This paper discusses the Chimbuko architecture enabling the approach. We present the use of a protein structure propagation workflow based on NWChemEx. We are testing algorithms for anomaly detection and present preliminary results here obtained with Local Outlier Factor. While scaling remains a challenge, these results show that our robust Chimbuko architecture for streaming analysis with prescriptive provenance is a promising approach.**

Keywords—computational workflows, prescriptive provenance, performance analysis, Chimbuko.

I. Introduction

The provenance of scientific workflows has traditionally been used to explicate scientific results - for instance, how a dataset was derived, what transformations have been applied to data, what dependencies exist in the workflow graph and its execution on distributed systems. Provenance is used to provide information enabling quality control, re-run computational workflows, and reproduce results. In this paper, we extend the use of provenance to encompass the performance of workflows on distributed systems. Performance in the execution of scientific workflows on shared systems can fluctuate at runtime due to many factors, including system workload, placement of the execution, shared libraries, etc. When cloud resources are used to execute workflows, provenance can be useful to predict the minimum amount of resources needed given time or resource constraint executions. In High Performance Computing environments, such as the DOE Leadership Class Facilities, scientists spend much time debugging and optimizing their codes with the goal of achieving performance improvements. These efforts are often impeded because, in these environments, many underlying libraries and variables that affect implementation change frequently. In this context, extracting detailed provenance for the optimization of workflow performance becomes advantageous to highlight latencies, possible contention for resources in workflows, and other bottlenecks that can slow down or crash execution. In previous work we have designed the Chimbuko system to capture, analyze and visualize the detailed provenance of a workflow and relate the extracted information to performance metrics [1, 2]. We have been using the Tuning and Analysis Utilities (TAU [3]) suite of tools modified to extract provenance variables of interest and extract the performance metrics for workflows, not just of single applications. The TAU + Chimbuko system captures performance metrics and provenance characteristics in a single trace file.

One important challenge encountered when extracting provenance and performance metrics for distributed workflow applications is the volume of extracted data [4]. Even when specifying filters and groups of variables - for instance, extracting only the execution events of MPI communication and suppressing performance extraction of all sub-routines, provenance and performance traces are very large and verbose. Given that simulation codes may run for hours on thousands of

Notice: The authors gratefully acknowledge the funding support from the U.S. Department of Energy Office of Science/ Office of Advanced Scientific Computing Research as part of the Exascale Computing Project (17- SC-20-SC) "Co-Design Center for Online Data Analysis and Reduction." This was authored in part by employees of Brookhaven Science Associates, LLC under Contract No. DESC0012704 with the U.S. Department of Energy.

cores or more, saving unreduced provenance and performance metrics for post-analysis is intractable. We address this challenge by analyzing the performance trace at runtime to detect anomalies (the events of interest) using anomaly detection algorithms. Once anomalies have been detected, we save all the provenance and a detailed trace for a time window of interest preceding the detected anomalies and make it available for future inspection (Fig. 1). We call *prescriptive provenance* the provenance of these events selected for retention by anomaly detection and retained for a time window preceding the occurrence of the events. As streaming performance trace data is produced during the course of a simulation, the amount of trace available for download and inspection preceding the interesting events is constrained by the memory resources of the system.

In this paper, we discuss the architecture enabling the collection, aggregation and saving of provenance and performance data and the provenance window. We present a use case of extracting performance metrics and provenance for an NWChem application. Global runtime performance traces are provided by TAU integrated with the SOSFlow architecture that provides aggregation of the traces for distributed applications [5]. We are testing several algorithms to detect anomalous events in performance data and support data reduction. We present preliminary results for one such algorithm.

II. Defining Prescriptive provenance

Prescriptive Provenance includes the traditional lineage of an application session (code names, versions, static metadata) and the provenance of the execution trace of scientific workflows at runtime, including application behavior, software stack, runtime environment, and system information.

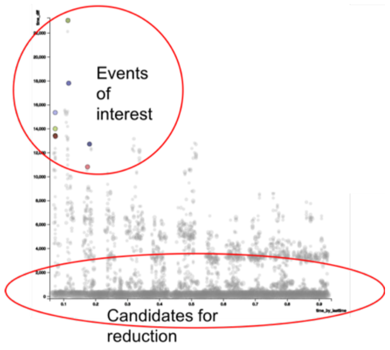

Figure 1: Schematic of data reduction approach - the events of interest (outliers) are saved, the non-events are discarded. The x axis are start timestamps of code functions; y represents execution time.

We extract the following performance metrics for each workflow component: start and end timestamp, call stack, memory allocation, I/O in network or disk, communication time and volume including between functions and nodes, and number of synchronization points. For each workflow, we extract the number of components, the amount of communication for each pair of components and size, the aggregated number of communication calls, and communication execution time. More traditional static provenance information such as executable names and versions, github hash of the codes, standard libraries and units of measurement are also extracted.

The time window to collect prescriptive provenance and the performance trace preceding the detected anomalies must be defined. Based on the purpose for which provenance is collected, this window will vary. For instance, in order to build training sets for anomaly detection algorithms, the entire provenance and the performance trace must be kept to train algorithms before they can be used with a streaming application. For the test sets, when anomalies are detected, only the prescriptive provenance is stored. For both training sets and in situ analytics, the rolling window will be algorithm dependent and can be specified for a particular algorithm. The window can be defined by time, steps in a simulation, or any other variable of interest to the event detection (a feature in ML parlance). If collecting prescriptive provenance is more generic than event detection and includes the goal of testing various event detection algorithms for forensic analysis, the window will need to be larger in order to accommodate varying factors. In addition, the amount of prescriptive provenance stored and the size of the rolling window will be bound by the available resources for computation and memory.

III. NWChem Use Case

We are using a protein structure propagation workflow based on classical Molecular Dynamics simulations with the NWChem application [6] which is the precursor to the current NWChemEx development. The NWChemEx project targets two different science challenges, one in catalysis, and one in biology. The biology science challenge considers the function and regulation of a transmembrane protein that acts as a calcium channel. The simulations need to include the protein structure itself but also the membrane, and the cell plasma and the aqueous environment outside the cell. To provide a realistic description of this environment it is expected that about one million atoms should be included in the model. In addition the regulation of the protein activity involves conformational changes to the channel. For one of the proteins of interest a change of protonation of one of the residues in the channels drives the protein to open or close the channel [7]. These conformational changes may take as long as a microsecond to complete. Given that the typical timesteps in these kinds of simulations are on the order of a femtosecond, accurate simulations may require on the order of a billion timesteps. This means, in principle, these calculations produce on the order of petabytes of data. Currently this data is reduced largely by using smaller models, storing the structure only every hundred or thousand timesteps, and running much shorter simulations. In fact, today, microsecond long simulations are typically one-off demonstration calculations and not routine simulations [8]. As compute capacity grows both larger models and longer timescales can be explored, but still the data storage capacity imposes practical limits if a conventional post simulation data analysis approach is used. If streaming data analysis is used the total volume of data produced is no longer a limiting factor, instead the size of the final analysis results becomes the critical factor. Dependent on the analysis techniques used these data sets may be considerably smaller as they often provide statistical measures and in addition may provide samples of trajectories that are selected based on features of particular interest rather than an interval sampling method.

Provenance is important in this context in two different ways. The target simulations are to be performed on extreme scale compute facilities. The facilities have a considerable number of factors that may affect the performance of the code. These factors include the hardware architecture, as well as various software aspects such as the versions of compilers, libraries, and the simulation source code. Even the particular workload on the machine when a calculation is run may affect the performance if different concurrent simulations cause contention for resources. Hence understanding the performance of the simulations requires storing a class of provenance data of key aspects that may affect performance. This data is needed both to understand the current run as well as a reference point for future comparison. The other way in which provenance data is important is that all computational campaigns involve a significant number of simulations run with different parameters, such as temperatures, pH, initial starting structures, etc. In addition the streaming data analysis will reduce the total amount of output data, for example, by selecting particular molecular configurations. To facilitate the interpretation of the results an additional set of provenance data is required that records key parameters of how the data was generated initially, and how the data was analyzed or reduced.

The simulations in this study use the AMBER95 force field [9]. The NWChem data analysis approach is modified to demonstrate streaming analysis methods. This was achieved by having the simulation write the trajectory file (subroutine SP_WTTRJ) while, in another process, the analysis module reads the file as new data becomes available, thus transforming data analysis subroutine (ANA_RDFRAM) into a data producer/consumer model. The performance instrumentation used in this case is the TAU source instrumentation - the only TAU method providing an executable for trace and profile data with readable symbol names in our case, extended to record POSIX data transfer through the trajectory file. This way we are able to demonstrate the performance implications of data transfer between workflow components. In addition, MPI job execution time was also recorded. While the approach used here still passes all data from one workflow component to another through a file, ongoing work will redirect this data stream through ADIOS [10] using in memory data staging.

IV. Architecture Overview

A. The Chimbuko system

We developed the Chimbuko system for streaming data analysis composed of three major components: provenance data collection, online anomaly detection, and provenance storage (Fig.2). The provenance data collection is implemented by using the Tuning Analysis Utilities (TAU) and Scalable Observation System (SOSFlow)[3,5] and they dynamically collect performance trace data for the workflow components and communication layer (green lines). Online anomaly detection parses the trace data collected by TAU and SOSFlow

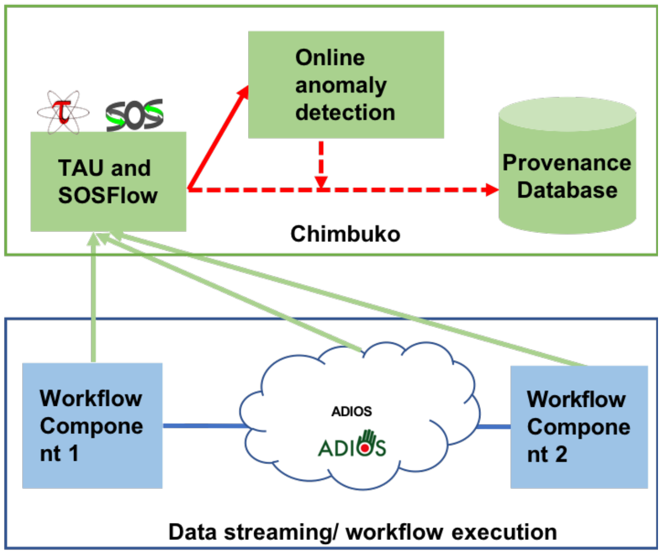

Figure 2: Overview of Streaming Architecture

and then analyzes the performance data by using the LOF algorithm (red lines). The input of the anomaly detection are trace data and the output are indices that lead to the trace data with anomalies. ADIOS enables data staging between components (blue lines). Provenance is saved in database for the detected anomalies. We collect metadata related to the environment, libraries, software stack, instrumentation parameters, function calls, per node, per thread, and indices for the anomalies.

Our current design (Fig. 2) is to fetch the provenance data from TAU and SOSFlow directly and save to the provenance database. Then the online anomaly detection sends signals to prune the provenance data stored in the provenance database to reduce volume. In this case, the online anomaly detection is more self-contained and efficient as it only reads trace data and dumps it whenever it does not need anymore. Also, the provenance database could keep a relatively larger size and longer time window of provenance data.

B. SOSFlow

To provide application performance measurement, we integrated the TAU Performance System into each of the applications in the workflow. TAU gathers performance data from parallel applications through several methods, including source instrumentation, binary instrumentation, profiling interfaces, runtime callbacks, and periodic sampling. For minimally intrusive experiments, TAU can simply be linked into the application or preloaded into the execution environment, providing access to a number of available runtime measurements from libraries that provide tool interfaces, such as MPI, ADIOS and OpenMP. With NWChemEx we use source instrumentation. To measure the application software, some selective instrumentation is used to capture main loops, computation routines, or suspected performance bottlenecks. TAU also captures a broad set of application metadata, including hardware, operating system, and input parameters. This metadata makes up part of the provenance available at runtime. In typical cases, the overhead from TAU measurement is less than 1-2% because either manual or automatic selective instrumentation helps eliminate the measurement of frequent, short-duration functions. With the addition of SOSflow, the overhead can increase slightly (in the 5-10% range) because every timer event measured with TAU is also packed to be periodically and asynchronously pushed over the SOS aggregation network. In this integration, a timer event is either a timer start/stop or a send/receive event, and may include other counter data such as bytes transferred or the current heap size when the event happened.

Global, runtime access to application performance information from multiple distributed applications requires an aggregation infrastructure. TAU cannot provide this global view on its own, and so it is integrated with the Scalable Observation System (SOS) implementation called SOSflow. The SOSflow topology is shown in Fig. 3. SOS is integrated into TAU using a plugin design, and performance events are selectively aggregated over the SOS aggregation tree. Analysis is executed at runtime using an additional allocated node or on one of the SOS aggregation nodes. The analysis extracts the streaming data as an ADIOS data stream, which is used as input for the Chimbuko anomaly detection. In this integration, the ADIOS stream represents the data as a full event trace, optionally reduced as necessary to control data volume.

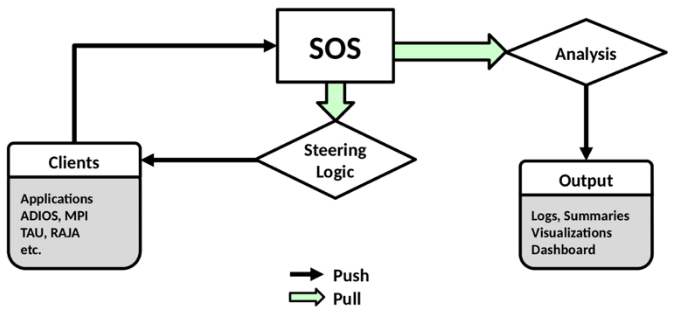

Figure 3: The SOSFlow topology

The SOS client library publishes to a data aggregation tree with an optional persistent store at each stage. Analysis clients can query the streaming or persistent store for online feedback control.

C. Anomaly detection

In this paper we will focus on detecting point anomalies in function execution times by using the Local Outlier Factor (LOF) [11] algorithm, an unsupervised outlier detection method. Our input data comes from NWChem trace files which were produced by TAU + ADIOS. Using these traces we identify the ten most frequently called functions, and extract their entry time stamps and execution times. The resulting time-series data are handled individually for each function; thus, it has only two attributes, timestamp (x-axis) and execution time (y-axis).

The LOF method computes an anomaly score called Local Outlier Factor (LOF) for each data point. LOF measures density deviation with respect to k-nearest neighbors. Data points whose local density is low compared to the local densities of its k-nearest neighbors are considered outliers. The score for a data point p is computed by the following formula:

$$LOF_k(p) = \frac{1}{k} \sum_{o \in N_{(p,k)}} \frac{lrd_k(o)}{lrd_k(p)}$$

where k is the number of nearest neighbors, $N_{(p,k)}$ is the set of k nearest neighbors of p, and $lrd_k(o)$ is the local reachability distance of a data point o.

V. Results

The performance data from TAU + ADIOS currently consists of an aggregated trace-stream which is stored in the ADIOS specific .bp file format. Using the ADIOS python module, we extract the necessary function execution time data from the .bp files for analysis, which is done by the LOF method implemented in python's scikit-learn package.

We ran NWChem for approximately 200 seconds, then examined the resulting trace to extract the timestamps and execution times for 3 frequently called functions (Fig.4).

- Function 21: int _my_isend(void *, int, MPI_Datatype, int, int, MPI_Comm, MPI_Request *) – (Fig. 4a)
- Function 23: void _make_progress_if_needed(void) – (Fig. 4b)
- Function 26: MPI_Iprobe() – (Fig. 4c)

Fig. 4 illustrates the results of the anomaly detection runs on these time series, where each figure (4a, 4b, 4c) shows data obtained for a specific function. The two most important parameters of the LOF algorithm are the number of nearest neighbors (k-nearest neighbor) and the contamination parameter. If the number of nearest neighbors is large, relative to the number of data points, then LOF detects global anomalies; if the number of nearest neighbors is small then LOF will detect local anomalies. This is why the anomalies (represented by red dots in fig. 4) sometimes appear at the bottom rather than the top, since the density of the neighbors at the bottom is higher than on the top. The contamination parameter determines what percentage of the data is considered to be anomalous. We have kept the contamination parameter at a fixed size and we will look at the top 10 anomalies i.e. the 10 most anomalous points. This approach helps application developers focus on a relatively small number of events.

Table 1 shows the data for the most anomalous points of Function 21 with k=50. It exemplifies the provenance characteristics in its columns headings. When this data is stored and discoverable through search or through the visualization afforded by Chimbuko [12] scientists can easily find details of functions that exhibit latencies, where they are executed, and what the dependencies are. The benefits of storing provenance data in this way is the ease of discovery, an improvement over the raw NWChem trace files in .bp format serving as input to anomaly detection. In these trace files, function names, rank, and other counters are only referred to as indexing positions, making it tedious to get relevant functions names and other metadata.

Table 1: data for the most anomalous points of function 21

Anomaly rank	Program	MPI rank	Thread	Entry timestamp	Execution time
1	0	0	0	1534053599305340	8486
2	0	2	0	1534053722620850	11859
3	0	3	0	1534053806325120	40220
4	0	2	0	1534053724677150	9749
5	0	0	0	1534053724825120	10419
6	0	3	0	1534053601208750	5917
7	0	3	0	1534053631189130	4689
8	0	1	0	1534053597337720	4691
9	0	1	0	1534053722381030	6684
10	0	1	0	1534053785874220	5490

VI. Discussion and Future work

The LOF algorithm can identify both local and global anomalies and has been successfully used in a wide range of applications [13]. However, the method has a complexity of O(n^2), where *n* corresponds to the number of data points, and requires that all the data are stored in file system before the computations begin. Therefore, applying a vanilla version of the LOF algorithm for outlier detection on data streams with a high data rate is not feasible. In such cases we are investigating other anomaly detection algorithms in future work, for instance the memory efficient incremental local outlier (MiLOF) detection algorithm designed for high-rate data streams [14]. The MiLOF algorithm does not require storing all the data and works in a streaming fashion. Thus, it is a good alternative for situations where computations need to be executed online and the amount of data exceeds the storage capacity of the underlying system.

In the Chimbuko architecture, there are two possible design options for properly storing the prescriptive provenance data into the provenance database. In our current option, provenance data is collected by TAU/SosFlow and directly saved to the provenance database. This streamlines anomaly detection that operates only on trace data. Another design option is that the online anomaly detection reads all the provenance data from TAU and SOSFlow directly and processes it internally. The provenance database stores the output that contains the provenance data and the selected performance trace window from the online anomaly detection. Since the online anomaly detection might need to read provenance data it does not need for calculations and store it in a time window, the performance degrade could be a potential issue. But one advantage of this design is that the overall Chimbuko framework is simpler than the other design.

VII. Conclusion

In this paper we have presented a system for streaming analysis of performance informed by detailed provenance about function execution, location, and call stack that enables culling performance data while retaining events of interest selected by anomaly detection. Our plug-in architecture allows collecting performance and provenance data from distributed processes for workflow components executing on high performance systems in a streaming fashion. We demonstrated that the prescriptive provenance approach is useful for extracting provenance related to job execution from computational workflows.

References

[1] Pouchard, L., S. Baldwin, T. Elsethagen, C. Gamboa, S. Jha, B. Raju, E. Stephan, L. Tang, and K. Kleese van Dam, Use cases of computational reproducibility for scientific workflows at pre-exascale. Workshop on Computational Reproducibility CRE 2017. Colocated SC17: Denver, CO. https://arxiv.org/abs/1805.00967

[2] Pouchard, L., Malik, A., Van Dam, H., Xie, C., Xu, W., and Kleese van Dam, K.: 'Capturing provenance as a diagnostic tool for workflow performance evaluation and optimization'. Proc. New York Scientific Data Summit (NYSDS), New York, NY, 2017, https://doi.org/10.1109/NYSDS.2017.8085043

[3] Shende, S.S., and Malony, A.D.: 'The TAU parallel performance system', The International Journal of High Performance Computing Applications, 2006, 20, (2), pp. 287-311

[4] Wang, J., D. Crawl, S. Purawat, M. Nguyen, and I. Altintas. Big data provenance: Challenges, state of the art and opportunities. in Big Data (Big Data), 2015 IEEE International Conference on. 2015. IEEE. https://doi.org/10.1109/BigData.2015.7364047

[5] Wood, C., Sane, S., Ellsworth, D., Gimenez, A., Huck, K., Gamblin, T., & Malony, A. (2016). A scalable observation system for introspection and in situ analytics. Paper presented at the 5th IEEE Workshop on Extreme-Scale Programming Tools *(ESPT), Workshop on*. 2016, IEEE. p. 42-49.

[6] NWChem: M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, and W.A. de Jong. NWChem: A comprehensive and scalable open source solution for large scale molecular simulations. Computer Physics Communications , 181(9):1477-1489, 2010.

[7] Chang, Y., R. Bruni, B. Kloss, Z. Assur, E. Kloppmann, B. Rost, W.A. Hendrickson, and Q. Liu, Structural basis for a ph-sensitive calcium leak across membranes. Science, 2014. 344(6188): p. 1131-1135.

[8] Shaytan, A.K., G.A. Armeev, A. Goncearenco, V.B. Zhurkin, D. Landsman, and A.R. Panchenko, Coupling between histone conformations and DNA geometry in nucleosomes on a microsecond timescale: Atomistic insights into nucleosome functions. Journal of molecular biology, 2016. 428(1): p. 221-237.

[9] Cornell, W.D., et al., A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. Journal of the American Chemical Society, 1995. **117**(19): p. 5179-5197.

[10] Liu, Q., et al., Hello adios: The challenges and lessons of developing leadership class i/o frameworks. Concurrency and Computation: Practice and Experience, 2014. 26(7): p. 1453-1473

[11] Breunig, M.M., H.-P. Kriegel, R.T. Ng, and J. Sander. Lof: Identifying density-based local outliers. in *ACM sigmod record*. 2000. ACM. https://doi.org/10.1145/335191.335388

[12] Xie, C., W. Xu, K. Huck, S. Shende, H.J. Van Dam, K. Kleese van Dam, and K. Mueller, Performance visualization for TAU instrumented scientific workflows, in 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (IVAPP'18). 2018: Funchal, Madeira, Portugal.

[13] Huang, H., H. Qin, S. Yoo, and D. Yu, Physics-based anomaly detection defined on manifold space. ACM Trans. Knowl. Discov. Data, 2014. **9**(2): p. 1-39. 10.1145/2641574

[14] Salehi, M., Leckie, C. A., Moshtaghi, M., & Vaithianathan, T. (2014). A relevance weighted ensemble model for anomaly detection in switching data streams. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2014).

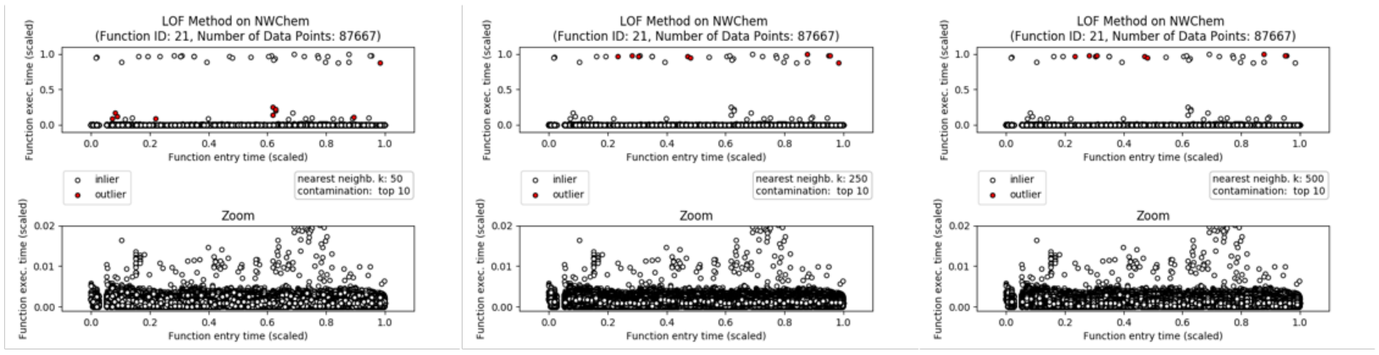

Figure 4a: Function 21

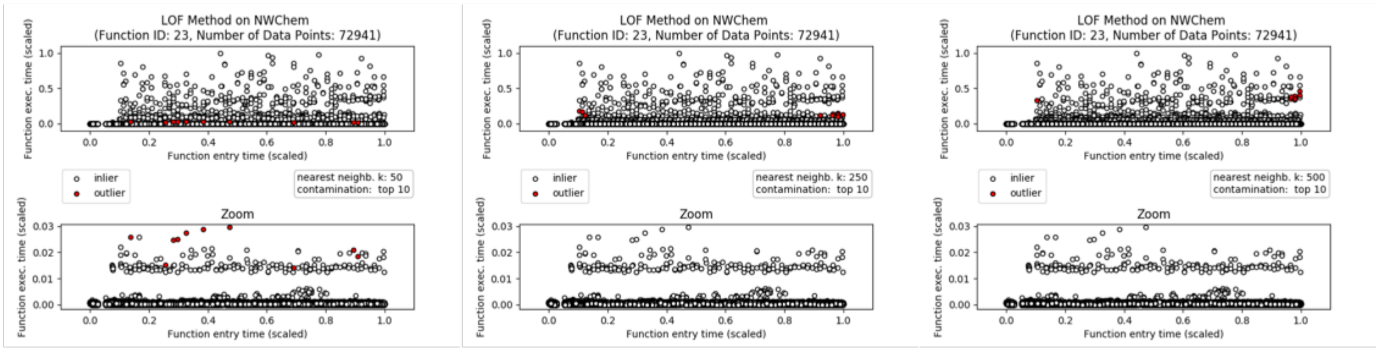

Figure 4b: Function 23

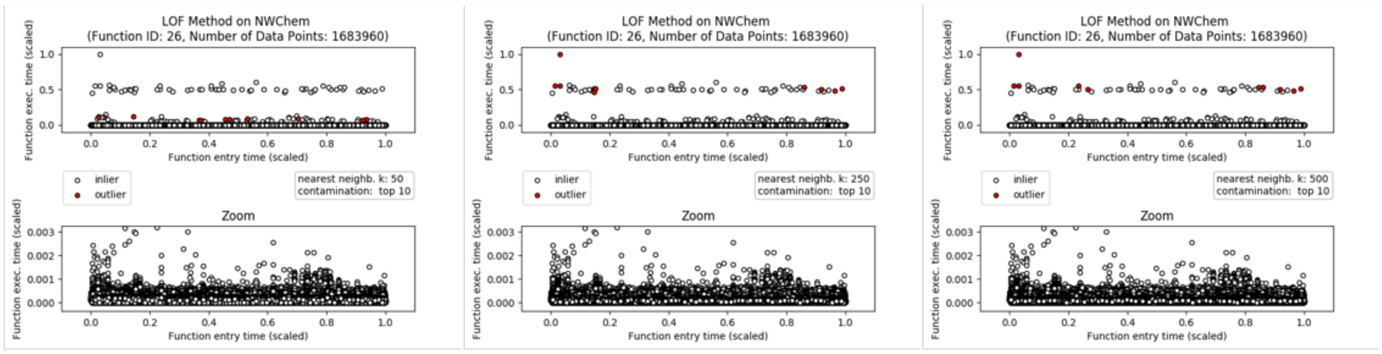

Figure 4c: Function 26

Figure 4: Anomalies detected for functions 21, 23, and 26, with k=50, 250, and 500 shown as red dots. Execution time for each function 21, 23, 26 is the y axis and entry time into the function the x axis. The data were scaled to have values between [0,1] for the function execution time.