Exploratory Visual Analysis of Anomalous Runtime Behavior in Streaming High Performance Computing Applications

Cong Xie[1], Wonyong Jeong[1], Gyorgy Matyasfalvi[2], Hubertus Van Dam[2], Klaus Mueller[1 2], Shinjae Yoo[2], and Wei Xu[2]

[1] Stony Brook University, Stony Brook NY 11790, USA
[2] Brookhaven National Laboratory, Upton NY 11973, USA

Abstract. Online analysis of runtime behavior is essential for performance tuning in streaming scientific workflows. Integration of anomaly detection and visualization is necessary to support human-centered analysis, such as verification of candidate anomalies utilizing domain knowledge. In this work, we propose an efficient and scalable visual analytics system for online performance analysis of scientific workflows toward the exascale scenario. Our approach uses a call stack tree representation to encode the structural and temporal information of the function executions. Based on the call stack tree features (e.g., execution time of the root function or vector representation of the tree structure), we employ online anomaly detection approaches to identify candidate anomalous function executions. We also present a set of visualization tools for verification and exploration in a level-of-detailed manner. General information, such as distribution of execution times, are provided in an overview visualization. The detailed structure (e.g., function invocation relations) and the temporal information (e.g., message communication) of the execution call stack of interest are also visualized. The usability and efficiency of our methods are verified in a real-world HPC application.

Keywords: Anomaly Detection · High Performance Computing · Streaming Analysis · Trace Events · Visual Analytics.

1 Introduction

Performance analysis is a critical task for the diagnosis of parallel High Performance Computing (HPC) applications. In particular, domain scientists are typically interested in detecting abnormal runtime behaviors during the execution of HPC applications. Since the supercomputer resources in use are limited and costly, the timely identification of the causes of adverse performance events (e.g., abnormal communication latencies) is essential. We have been working with a group of chemists who use an HPC cluster to solve complex molecular equations. The development of the system presented in this paper was driven by their need to monitor and identify computation latencies at runtime.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

Anomalous function executions are usually identified by examining the detailed traces collected in the HPC cluster. A trace is essentially a log of a sequence of specific events (determined by program instrumentation) that occur in a computing core during execution (e.g. function entry, function exit, or message passing). Fig. 1 (a) shows an example sequence of trace events inside one execution of the `main` function in an HPC core. It represents the call stack information (Fig. 1 (b)) inside the execution of the root function. Domain scientists typically detect anomalous function executions based on the extracted temporal information [18] [2] (e.g., execution time and exit timestamps) or structural information [22] (e.g., call relations from parent function to children) from the trace events. While existing detection approaches achieve good performance

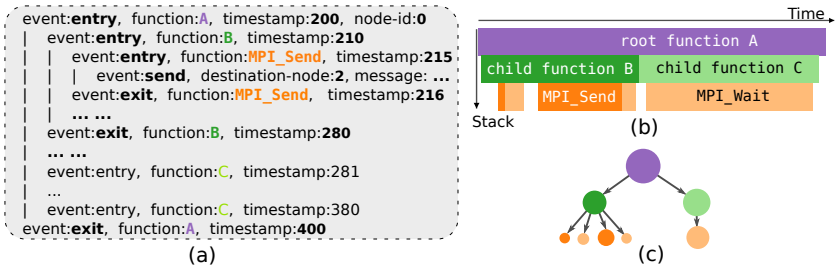

Fig. 1. (a) Example trace events during an execution of `main` in a compute core. (b) The call stack of the execution reconstructed from the trace events. (c) The call stack represented as a Call Stack Tree (CSTree), which is a directed tree with vertex weights.

and scalability in offline analysis, detecting abnormal runtime behavior in online analysis remains challenging. One reason for this is that the complexities of most feature extraction and anomaly detection algorithms are too high to support online training. It would be prohibitively slow to use a standard algorithm like Local Outlier Factor (LOF) [3] to update the learning result every time a new trace event is generated. Conventional anomaly detection models do not support continuous updates, rather they need to be re-trained in each time window. Furthermore, in order for human experts to stay aware and gain insights into the performance data an interactive visual interface will be most appropriate. However, offline visual analysis tools are not designed to provide visualizations sufficiently responsive in a streaming data environment. For example, standard visualization algorithms such as Multidimensional Scaling (MDS) [12] or t-Distributed Stochastic Neighbor Embedding (t-SNE) [13] are not fast enough to plot tens of thousands of points at the rate of streaming data.

We recently devised a visual analysis approach for the detection of abnormal HPC runtime behavior [22]. This tool, however, was designed for offline analysis and cannot be used for online analysis of streaming data. In the current work we build on this approach to create a new method for streaming trace event data. Similar to our earlier work we make use of the call stack tree (CSTree) representation we devised there (Fig. 1 (c)), but we now use it to encode the structural as well as the temporal information inside a function execution via

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

a directed tree. Abnormal behavior of function execution can then be detected by identifying abnormal trees in a call stack forest. We also modify our original tree feature extraction process to now support online tree representation and online anomaly detection. We propose a set of new online visualizations to assist the scientist in the understanding and exploration of the function executions. In particular, we visualize the feature vectors and the learned anomaly labels of the function executions in an overview projection. The structural and temporal information of the CSTree are visualized in a structure and timeline visualization, respectively, capable of dealing with large amounts of data in real time.

The remainder of our paper is structured as follows. Section 2 reviews related work. Section 3 defines the problem and gives an overview of our approach. Section 4 introduces our online algorithms for vector representation and anomaly detection of CSTrees. Section 5 describes our visual analytics approach for anomalous CSTree exploration. Section 6 presents a case study to validate our approach. And section 7 concludes the paper and discusses the future work.

2 Related Work

Domain scientists usually employ instrumentation and measurement tools [1] [5] [10] [16] to generate performance data, such as trace events, and associated metrics. Many techniques have been proposed for the evaluation and diagnosis of these data [4]. In this paper, we focus on the most important two tasks in the diagnosis of the HPC application performance: detection of abnormal runtime behavior and visualization of performance.

2.1 Abnormal Runtime Behavior Detection

For anomalous runtime behavior detection, most existing approaches identify candidates based on temporal information [7]. For example, a disproportionately large function execution time has a very high probability of being abnormal since it has a large negative impact on the downgraded performance. However, diagnosis without the execution context makes it difficult to determine the source of this major latency. For example, the cause in the delay may be triggered by a child function or by another node in the HPC system (due to delayed communication). Furthermore, semantics of the executions are also necessary for the diagnosis. For example, the initialization function call may execute more computations and communicates more than other functions and as result takes more time to complete than other function calls. However, this phenomenon should not be identified as abnormal runtime behavior.

In contrast, the Call Stack Tree representation (CSTree) [22] can encode the temporal information as well as the context structure obtained from the call stacks to identify the potential anomalous executions. The embedding vectors are then generated from the CSTrees. Each vector encodes the structural information from the call stack of one function execution. A conventional anomaly detection

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

algorithm such as the One-Class Support Vector Machine (OCSVM) [15] is then able to take the vectors and identify the anomalies.

However, both the tree representation and the anomaly detection are too complex for online analysis. To deal with that, in this paper, we modify the training strategy of these learning algorithms for the analysis of streaming data.

2.2 Performance Visualization

Performance visualization makes problem detection and diagnosis of HPC applications [7] [21] more transparent to domain experts. Most visualization approaches support the comprehension of different levels and aspects of the performance data, including the trace timeline, call stack structures, and the messages.

For the trace events, a common practice is to visualize the events along a time axis, as is done in Vampir [9] and Jumpshot [23]. Most existing temporal visualizations provide level-of-detail explorations. Users can zoom into different time window granularities to see detailed events [21]. Other temporal visualizations are also capable of presenting the relationships between threads, such as SyncTrace [8].

Call path visualization (e.g., the call relationship between parent and children functions) is critical for understanding the behavior of the runtime execution. Existing approaches employ a directed tree or graph to present the structure in a call stack, such as Vampir [9]. These visualizations usually use the visual properties of the tree to encode detailed information from the call path. For example, CSTree [22] utilizes the color and size of a tree node to encode the type and the execution duration of a function.

Communication delay is usually the main reason for application latency. A straightforward approach to encode the message passing is to draw a directed line between the sending and receiving functions, which is adopted in the Jumpshot [23] implementation. Since the messages can also be regarded as directed edges, the communication between threads or processes can also be summarized in terms of an adjacency matrix [9].

One major issue with the existing visualizations is their limited capability for online performance evaluation. Some offline analysis tools focus on the complete event or message passing structure which is available only when the communication is finished. In addition, most visualization paradigms, such as MDS are too complex for real time streaming data. On the other hand, we also need to deal with online data reduction and sampling to prevent overdraw. In our approach, we adjust the common projection, timeline, and tree visualizations for streaming to facilitate incremental updates and data visualizations.

3 Problem Formulation and Approach Overview

We focus on the following problem: Given a set of functions of interest (FOIs) {`A`} and all of their invoked executions in an HPC cluster, determine which executions are associated with anomalous runtime behavior. An FOI `A` is usually

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

a key function for computation or communication, which is specified by domain experts. An anomalous runtime behavior is then in most cases indicated by temporal or structural features. For example, a deadlock will cause large execution time and an infinite loop will generate unexpected call path structure.

3.1 Call Stack Tree Representation and Problem Formulation

As mentioned in Section 2, the Call Stack Tree (CSTree) (Fig. 1 (c)) representation provides a comprehensive way to encode the execution of an FOI `A` since it takes advantage of the execution's context information. Each execution of `A` is converted to a CSTree T where a vertex in T is a function invoked in the call stack and a directed edge represents the call from the parent to the child function. All executions of `A` collectively give rise to a forest $\mathcal{T} = \{T_i\}$ where each tree T represents a single execution of `A`. The runtime behavior can be observed directly from the features of a CSTree. For example, if the volume of a vertex in the CSTree is proportional to the execution time of that function, then a very large vertex in the tree can represent a delayed function execution. Furthermore, a large set of child vertices of the same function type indicates that the parent function invokes the child function multiple times in a loop.

Given this representation the detection of anomalous behavior is then formulated as the problem of finding anomalous tree structures in the call stack forest. Our visual interface exposes the candidate anomalies, which are those CSTrees whose structures differ most.

3.2 Approach Overview and System Architecture

Based on the CSTree representation, our online visual analytics approach for detecting anomalous CSTrees uses the following four steps with the architecture[3] in Fig. 2:

Step 1 Data processing: First, the data analysis server (Fig. 2 (b)) pairs and orders trace events (Fig. 2 (a)) generated during the execution of an HPC application by their type (i.e., message or function event). Second, given a set of functions of interest {`A`}, the data analysis server can also generate the call stack trees rooted at `A` and insert them into the call stack forest $\mathcal{T} = \{T_i\}$.

Step 2 Tree feature extraction and anomaly detection: Each new CSTree in the forest is converted into a feature vector. For this paper the feature vector consists of the temporal features of the root functions (i.e. total duration of execution). However, other options for feature extraction, such as the graph kernel, are also provided in the code. Anomaly detection is then performed based on these features, resulting in a set of candidate anomalous CSTrees.

Step 3 Overview visualization: An overview projection is calculated by the visualization server (Fig. 2 (c)). The visualization platform is the web browser (Fig. 2 (d)) showing the feature vector distribution and their learned labels.

[3] Please visit https://github.com/CODARcode/ChimbukoVisualization to see the project page and the source code.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

Step 4 Detailed visual exploration: The detailed visualizations enable the user to investigate the execution's context and make a decision whether a candidate anomalous CSTree is truly anomalous. In specific, the user can select a feature vector from the overview to view the associated CSTree structure and communication patterns (both provided by the visualization server) to understand the execution's context.

Steps 3 and 4 can be performed repeatedly for better insights into the anomaly detection results.

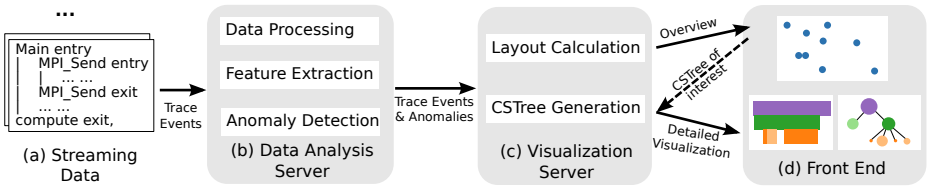

Fig. 2. Our system architecture for online anomalous CSTree detection. (a) The streaming trace events. (b) Feature vectors of the new CSTrees are generated in the data analysis server. Then the anomaly labels (normal or abnormal) of the CSTrees are learned online based on their feature vectors. (c) The visualization server calculates the layout of the overview and generates the CSTree of interest for detailed exploration. (d) On the browser end, the user is allowed to interact with the overview of the tree distribution in a 2D view. The candidates of interest in the overview can be further investigated via the detailed visualizations provided by the server.

4 Online Anomaly Detection of Call Stack Trees

Our approach begins with updating the call stack forest in the data server. For a FOI `A`, a new CSTree is generated and inserted into the call stack forest when an exit event of `A` is received in the trace stream. Since the user will focus on the recent data, the CSTrees older than a given threshold (e.g., 1 hour) will be removed from the forest. Then the feature extraction and anomaly detection are performed for the updated call stack forest in the data analysis server.

4.1 Feature Vector Representation

Our approach provides different options for temporal and structural feature extraction from the CSTree for anomaly detection, including time-based and tree-based extraction methods.

Time-Based Representation A straightforward way to represent runtime behavior is to utilize the information from the root function of the CSTrees. The domain expert is able to customize the specific set of features to be extracted, such as the execution duration, message frequency, and the exit time of the execution. With this pre-designed feature set, a vector of each CSTree is constructed and used as the input for anomaly detection.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

Tree-Based Representation Thus far there is no contextual information in the above temporal feature construction, however, structural representation is critical for the diagnosis of the runtime behavior. To provide an option for structural feature extraction, we follow the Graph Kernel [22] approach for the vector representation, which uses an analogy to document analysis. A tree is analogous to a document while the subtrees are analogous to the words in a document. The subtrees (Fig.3 (b)) are extracted using Weisfeiler-Lehman Graph Kernels [17]. In the initialization of the algorithm, each node is considered as a subtree of depth 0 (e.g., `A` - `E` in Fig. 3 (b)). Then in the following iteration, each subtree is expanded towards the children to find sub-structures of larger depths, such as `F` - `J` in Fig. 3 (b). At last, the CSTree (Fig.3 (a)) is represented as a bag-of-subtrees (Fig.3 (c)). The duration of each subtree in a CSTree is synonymous to "word frequency" in a document.

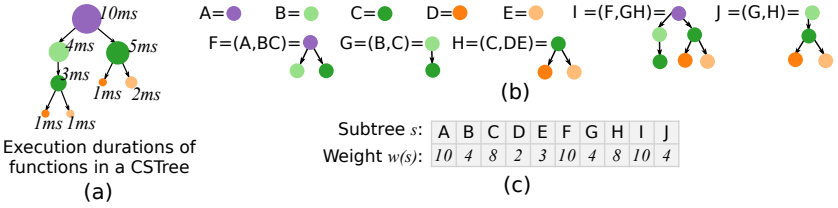

Fig. 3. (a) An example CSTree with the function execution time of the vertices. (b) An example of the result substructure generated from a CSTree using Weisfeiler-Lehman Graph Kernels. (c) The bag-of-subtree representation of the CSTree.

4.2 Online Anomaly Detection

The extracted feature vectors of the CSTrees are used as the input of the anomaly detection. Conventional algorithms such as Local Outlier Factor can be too complex for online training. For better online performance in the presence of a very large amount of streaming data we chose a fairly straightforward statistical approach. Using a Gaussian distribution to model the feature vector distribution of the CSTrees, we calculate the rolling mean (μ) and standard deviation (σ) of the feature vectors in the call stack forest using Welford's method [14]. We then label all CSTrees at 3σ (i.e., 0.003%) of the data distribution as anomaly candidates. In our own experiments we found this to be a good threshold, but the confidence level can be user-adjusted based on the estimated percentage of anomalies in the dataset.

5 Visual Exploration of Anomalous Call Stack Trees

It is necessary that human experts can verify the learning results to make sure the identified candidates are true anomalies. In addition, exploration of the temporal

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

and structural patterns of the candidates also helps the user understand why they are potentially anomalous. In this section, we describe our level-of-detail visualization system designed for the exploration and verification of the CSTrees in different granularities.

5.1 Overview Visualization

The overview visualization is generated by the visualization server and displayed in a browser-based interface. The overview shows the general distribution of the feature vectors of CSTrees. As noted above, the most common approach available for this purpose is a low-dimensional embedding method such as MDS. However, after some experimentation we determined, as we also mentioned above, that these methods are not sufficiently fast and scalable for large streaming datasets.

To deal with this problem, we resorted to a standard bivariate projective scatterplot approach. In this visualization the x and y positions are calculated by the basic attributes of the CSTrees specified the user. For example, in Fig. 4 (a), the CSTrees are visualized as points in the scatter plot. They are projected by the execution time and exit time of their root functions. The color of each points represent the FOI type. The points highlighted with thick borders are candidate anomalies which are detected in the previous step.

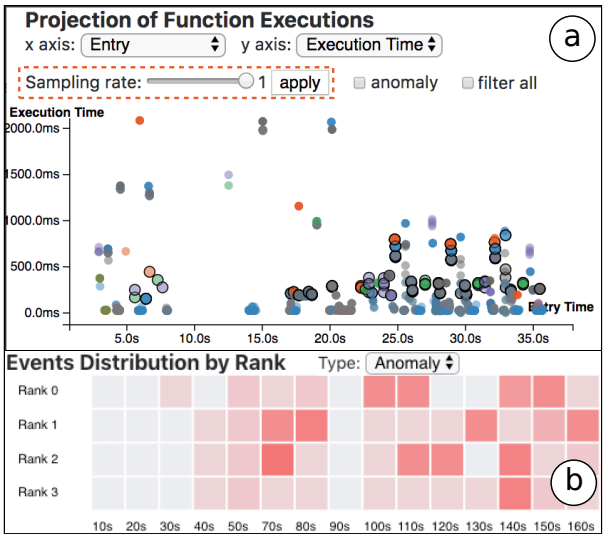

Fig. 4. (a) The CSTrees are projected by the execution durations and the entry timestamps of their root functions. The user can specify other axis-encoding schema for projection. Filtering and negative down sampling are supported to reduce visual clutter. (b) The heatmap visualizes the temporal distribution of all the history data.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

Display scalability can still be a problem when the data volume is massive. After discussing this with a domain expert we decided to adopt a negative down sampling strategy to reduce the data that are displayed. Since users are typically mainly interested in the candidate anomalies, they can set a down sampling rate (see the dashed box in Fig. 4 (a)) to keep only a portion of the normal points in the projection. At the extreme, when the down sampling rate is set to 0, the projection will only visualize the candidate anomalies. In addition, users can also filter the displayed data by FOI type. This helps users to put more focus on a subset of more important FOIs.

In an online analysis, as opposed to a postmortem analysis, users will usually want to focus on the latest trace data. Hence, the visualization server will only maintain a subset of the data, namely those collected within a given time period (e.g., 1 hour). The scatter plot will also be updated whenever new CSTrees are processed and generated from the server end. On the other hand, even in online analysis, keeping a historical perspective is desirable. We provide this view by visualizing the distribution of all history data in form of a heatmap (see Fig. 4 (b)). In this map the user is free to specify the color encoding used for highlighting the number of anomalies or trace events.

Further, the user can specify the points of interest in the scatter plot, which will send requests to the visualization server for more detail on these selections. The structure and timeline visualizations will then provide the corresponding CSTree structures and events sequences for further exploration.

5.2 Structure Visualization

The details of the selected CSTree are visualized in the Structure Visualization, as shown in Fig. 5 (a). The tree vertex size and color respectively represent the execution time and function name of the corresponding tree node. The directed edge shows the call relationship from a parent to a child function. Force-directed Layout [11] is employed to calculate the position of the tree vertices. For clearer representation, we set a maximum depth to only keep important parent functions which influence the tree the most. This reduces the visual clutter, which is helpful especially for CSTrees with a large number of descendant functions. On the other hand, in order to provide complete anomaly information, the abnormal substructures are preserved in the visualization even if they are beyond the maximum depth limit.

5.3 Timeline Visualization

While the structure visualization is effective, the message communication between different HPC cores is not visualized. To deal with this problem, we propose the Timeline Visualization (Fig. 5 (b)), which basically follows the visual design of Vampir [9] and Jumpshot [23]. Our visualization shows the event sequence as well as the message passing in a stack timeline. The x and y axes of the timeline encode the timestamp and the growing direction of the call stack. Since communication is one of the main reasons for HPC application delays, messages

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

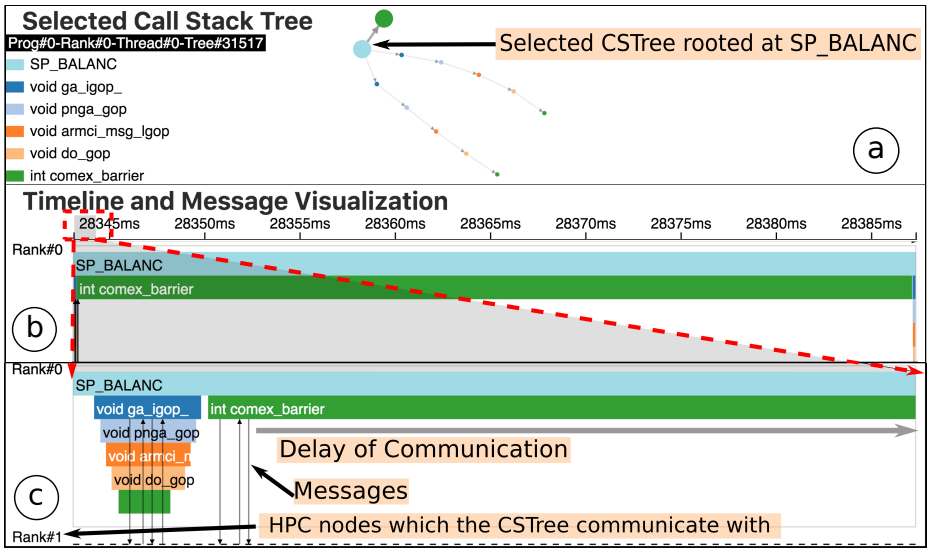

Fig. 5. (a) The call stack tree structure visualization shows the tree at a limited depth except for abnormal substructures. (b) The timeline visualize the event sequence and message communications. (c) The timeline can be zoomed in for detailed exploration.

are also visualized, via lines between different cores, as indicated in Fig. 5 (b). Users can zoom in and out of the x-axis (see red box in Fig. 5 (b)) and so explore the detailed call stack in different time ranges, as shown in Fig. 5 (c).

6 Case Study

We conducted a case study[4] with an NWChem [19] developer at Brookhaven National Laboratory (BNL). NWChem is a massively-parallel computational chemistry application deployed on BNL's HPC cluster. Our performance analysis study presented here focuses on analyzing NWChem's the molecular dynamics functions.

6.1 Experiment Settings and Online Performance Evaluation

Our participant is a chemist and not an expert in visualization or machine learning and sought to use our system to find anomalous runtime behaviors in the function executions. In the onset, we had a number of thorough discussions with this domain scientist to learn about functions of interest and possible anomalous behavior patterns. We also held a short training session to introduce our visualization system.

[4] Please see our video demo here and case study design details here.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

We applied the temporal feature extraction described in Section 4.1 since it was faster for large scale datasets. We also employed the anomaly detection by standard deviation described in Section 4.2. After studying the dataset in our interface, the participant set the significance level defining a candidate anomaly to be roughly 3σ (i.e., 0.003%) to reduce false positives.

The online analysis will not affect the HPC application execution since it only takes the trace events output. The streaming trace events of NWChem were generated by SOSFlow [20] and ADIOS [6] in real time. We tested different NWChem application settings with different molecular system sizes (small and large in the first columns of Table 1) as well as the scalability in different HPC settings (the second columns of Table 1). The details for the six datasets are shown in Table 1. We found that our feature extraction and anomaly detection algorithms did not cause significant delays for the streaming analysis. The throughput of our system was acceptable according to our user.

NWChem Setting	Number of HPC cores	Number of trace events	Number of trace events per second	Number of anomalies	Throughput (MBps)
small	2	393,542	58.5k	350	5.2
small	4	1,201,533	128.1k	445	11.5
small	8	4,024,651	224.2k	2235	20.5
large	2	784,122	52.9k	818	4.7
large	4	2,386,634	101.0k	1121	9.1
large	8	7,972,872	172.9k	3683	15.8

Table 1. Summary of experiment datasets and the throughput.

6.2 Case 1: Delay of Communication

During the online analysis, our participant first examined the overview distribution of the CSTrees. He was interested in the `SP_BALANC` function, which was designed to redistribute the work over the processors to minimize the time spend waiting in communication functions. He suggested that it was a critical factor to the overall performance of the code. In the scatter plot, he noticed that most of the points in the projection were normal. To put more focus on the potential abnormal CSTrees, he reduced the negative sampling rate (Fig. 4 (a)). He noticed an abnormal point, which was execution `#31517` of the Thread `#0` in Rank `#0` of the program `#0`, as shown in the upper left of Fig. 5 (a).

From the Structure Visualization, he found that there was a big green function of `comex_barrier` which spent the majority of the time in `SP_BALANC`. `comex_barrier` was a function responsible for the communication between different HPC cores. He made an assumption that the barrier function was the major reason for the latency of this execution.

To learn about the temporal pattern and message communication, our participant examined the Timeline Visualization. He found that barrier function

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

invoked some communications, as shown in the message passing visualizations with Rank `#1` in Fig. 4 (c). After comparing with other regular executions of `SP_BALANC`, he concluded that this execution waited for a long time for the response of Rank `#1`. As a result, the communication delay to other computing cores made this execution a candidate anomaly. He concluded that our system helped him understand one of the reasons for the performance fluctuation of `SP_BALANC`, which provided insights of how to optimize the source code to improve the overall program performance.

6.3 Case 2: Delay of Computation

Our participant continued to explore other functions of interest. In the scatter plot, he only visualized `CF_CENMAS` function, which computed the center of mass coordinates of individual molecules in the NWChem simulation. He located another candidate anomaly and showed its detailed structure and timeline. From the structure visualization (Fig. 6 (a)) he learned that it was abnormal since the root node in the CSTree was very huge. He zoomed into the timeline (Fig. 6 (b)) and found that all of the child functions were executed as expected; however, they were invoked after waiting a long time for `CF_CENMAS`.

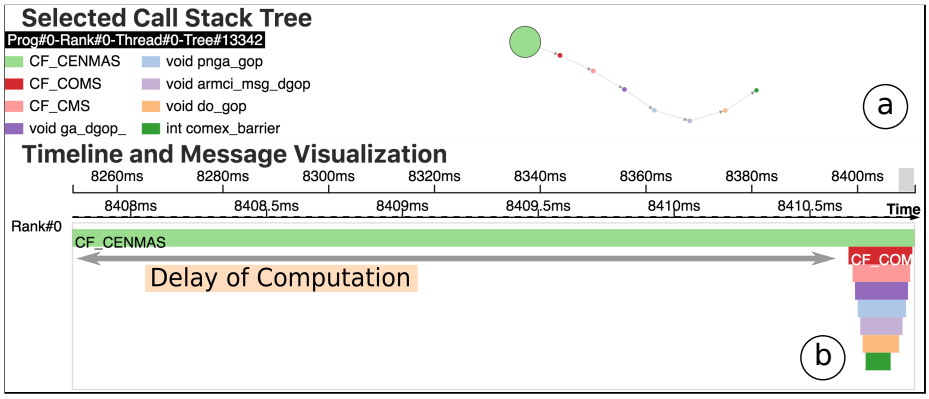

Fig. 6. (a) The CSTree structure of the `CF_CENMAS` function. (b)The user zoomed in the timeline to see the detailed temporal sequence at the beginning of the execution, as shown in the gray range in the time axis.

He expressed that this could happen when the computation in `CF_CENMAS` took a long time. The descendant functions (e.g., `comex_barrier`) had to wait for the computation to finish to communicate with other HPC cores. Our expert noted that the visual exploration our system provided was a critically important supplement to the learned labels (i.e., normal or abnormal) of the automatic algorithm. He stated that our system provided the much needed comprehensive analysis support for understanding and diagnosing the runtime behavior.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

6.4 Feedback and Discussion

After the case study with the chemist, we invited additional 5 scientists from BNL to use our system. We conducted an interview session where we asked them to rate our system and give feedback. Their average ratings were 4.8 for usability (1=not useful, 5=very useful) and 4.7 for learning cost (1=hard to learn, 5=easy to learn). One participant mentioned that the visualizations were easy to understand since they are also commonly used by the existing performance analysis tools [7] [4]. He also compared our system with the commonly available tools in his community. He indicated that with current tools such as Jumpshot [23] he would only be able to learn about the executions after a lengthy session with the system. He would first manually locate the time window of an anomaly candidate and then look at a detailed view on the respective call paths and messages. Conversely, he said that our interface was more intuitive, enabling him to quickly discover the anomaly and succinctly explain it by using the three linked views.

7 Conclusion and Future Work

We described a visual analytics approach for the online detection of anomalous function executions in HPC clusters and their visual exploration. Our approach is based on the CSTree representation. It provides effective anomaly detection and visualization tools that address challenges with streaming performance evaluation for parallel computation at scale. We demonstrated our approach with a real world NWChem application.

In the case study, we learned that our bag-of-subtree vector can be too sparse since the subtree corpus will be massive. To cope with this problem, a Stack2vec embedding [22] can be a viable option to generate the embedding vectors from the sparse bag-of-subtree vector. We also plan to integrate machine log analysis and source code examination into our system which will provide more insights into the execution scheduling and code design optimization.

Acknowledgments

This research was partially supported by NSF grant IIS 1527200, BNL LDRD grant 16-041 and 18-009, ECP CODAR project 17-SC-20-SC, and the MSIP (Ministry of Science, ICT and Future Planning), Korea, under "IT Consilience Creative Program (ITCCP)" supervised by NIPA.

References

1. Adhianto, L., et al.: Hpctoolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience **22**(6), 685–701 (2010)
2. Arnold, D.C., Ahn, D.H., De Supinski, B.R., Lee, G.L., Miller, B.P., Schulz, M.: Stack trace analysis for large scale debugging. In: IPDPS. pp. 1–10. IEEE (2007)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12

3. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: ACM SIGMOD. vol. 29, pp. 93–104 (2000)
4. Ezzati-Jivan, N., Dagenais, M.R.: Multi-scale navigation of large trace data: A survey. Concurrency and Computation: Practice and Experience **29**(10) (2017)
5. Geimer, M., Wolf, F., et al.: The scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience **22**(6), 702–719 (2010)
6. Gu, J., Klasky, S., Podhorszki, N., Qiang, J., Wu, K.: Querying large scientific data sets with adaptable io system adios. In: Yokota, R., Wu, W. (eds.) Supercomputing Frontiers. pp. 51–69. Springer International Publishing, Cham (2018)
7. Isaacs, K.E., Giménez, A., Jusufi, I., Gamblin, T., Bhatele, A., Schulz, M., Hamann, B., Bremer, P.T.: State of the art of performance visualization. EuroVis 2014 (2014)
8. Karran, B., Trumper, J., Dollner, J.: Synctrace: Visual thread-interplay analysis. VISSOFT 2013 **00**, 1–10 (2013)
9. Knüpfer, A., Brunst, H., Doleschal, J., et al.: The vampir performance analysis tool-set. In: Tools for High Performance Computing, pp. 139–155. Springer (2008)
10. Knüpfer, A., et al.: Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In: Tools for High Performance Computing 2011, pp. 79–91. Springer (2012)
11. Kobourov, S.G.: Spring embedders and force directed graph drawing algorithms. arXiv preprint arXiv:1201.3011 (2012)
12. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika **29**(1), 1–27 (1964)
13. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov), 2579–2605 (2008)
14. Salonen, J.: (2013), http://jonisalonen.com/2013/deriving-welfords-method-for-computing-variance/
15. Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J.: Support vector method for novelty detection. In: NIPS. pp. 582–588 (2000)
16. Shende, S.S., Malony, A.D.: The tau parallel performance system. Int. J. High Perform. Comput. Appl. **20**(2), 287–311 (2006), http://dx.doi.org/10.1177/1094342006064482
17. Shervashidze, N., Schweitzer, P., Leeuwen, E.J.v., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(Sep), 2539–2561 (2011)
18. Sigovan, C., et al.: A visual network analysis method for large-scale parallel i/o systems. In: IEEE IPDPS. pp. 308–319 (2013)
19. Valiev, M., et al.: Nwchem: a comprehensive and scalable open-source solution for large scale molecular simulations. Computer Physics Communications **181**(9), 1477–1489 (2010)
20. Wood, C., Sane, S., Ellsworth, D., Gimenez, A., Huck, K., Gamblin, T., Malony, A.: A scalable observation system for introspection and in situ analytics. In: Proceedings of the 5th Workshop on Extreme-Scale Programming Tools. pp. 42–49. ESPT '16, IEEE Press, Piscataway, NJ, USA (2016)
21. Xie, C., Xu, W., Ha, S., et al.: Performance visualization for tau instrumented scientific workflows. In: VISIGRAPP (3: IVAPP). pp. 333–340. SciTePress (2018)
22. Xie, C., Xu, W., Mueller, K.: A visual analytics framework for the detection of anomalous call stack trees in high performance computing applications. IEEE transactions on visualization and computer graphics (2018)
23. Zaki, O., Lusk, E., Gropp, W., Swider, D.: Toward scalable performance visualization with jumpshot. Int. J. High Perform. Comput. Appl. **13**(3), 277–288 (Aug 1999), http://dx.doi.org/10.1177/109434209901300310

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22734-0_12