Applying Frequency Analysis Techniques to DAG-based Workflows to Benchmark and Predict Resource Behavior on Non-Dedicated Clusters

Vivek K. Pallipuram, Jeffrey DiMarco, and Michela Taufer
University of Delaware
Newark, DE 19716
Email: kpallip, jdimarco, taufer@udel.edu

***Abstract*—Today, scientific workflows on high-end non-dedicated clusters increasingly resemble directed acyclic graphs (DAGs). The execution trace analysis of the associated DAG-based workflows can provide valuable insights into the system behavior in general, and the occurrences of events like idle times in particular, thereby opening avenues for optimized resource utilization. In this paper, we propose a bipartite tool that uses frequency analysis techniques to benchmark and predict event occurrences in DAG-based workflows; highlighting the system behavior for a given cluster configuration. Using an empirically determined prediction window, the tool parses real-time traces to generate the cumulative distribution function (CDF) of the event occurrences. The CDF is then queried to predict the likelihood of a given number of event instances on the cluster resources in a future time frame. Our results yield average prediction hit-rates as high as 94%. The proposed research enables a runtime system to identify unfavorable event occurrences, thereby allowing for preventive scheduling strategies that maximize system utilization.**

I. Introduction

The scientific workflows of petascale applications executed on non-dedicated clusters resemble directed acyclic graphs (DAGs), where the DAG nodes represent the computational tasks and DAG edges represent the dependencies and communication between tasks [1]. The dynamic analysis of DAG-based workflows can provide the runtime system with important insights when it is time to assign resources among different applications. For instance, the runtime system can detect the limited task-level parallelism in a scientific application and react by reallocating the expected idle resources to a different application. As the DAG-based workflows become increasingly complex, the analysis and prediction of event occurrences become more demanding in terms of both time for the analysis and memory for storing the traces. Because of the DAG-based workflow complexity, a visual analysis may be error-prone, while traditional analysis may face memory constraints. For the sake of efficient resource utilization, a runtime system must keep both time and memory costs under control and still reliably predict the occurrence of computation, communication, and idle events at runtime. The runtime system can ultimately use this knowledge to take precautionary actions and avoid unfavorable use of resources in the near future.

This work addresses this challenge for a diverse set of DAG-based workflows by seeking a general, scalable, and accurate solution that embraces diverse workflows with different sizes, level of dependencies, and complexities. Our solution relies on the frequency-based analysis of event traces. The analysis is performed by a bipartite tool that enables the study of event occurrences on the cluster resources including computation, communication, and idle events. Specifically, our tool analyzes DAG-based workflows in the Paje trace format [2] that are generated by a cluster using a popular runtime scheduling system such as StarPU [3]. The functionality of our tool is two-fold. First, it allows us to benchmark the DAG-based workflow in a single-pass approach and collects frequency-based statistics that are much smaller than the traces themselves. Second, it uses the statistics to identify a sweet-spot size for prediction windows that enables the estimation of future event occurrences with a certain confidence level.

From the implementation point of view, the tool includes two analysis phases. A first *benchmarking* phase uses the empirical cumulative distribution function (ECDF) of event occurrences to benchmark a set of diverse DAG-based workflows and produces a suite of window-size models. A second *prediction* phase uses the window-size models to generate the ECDF of event occurrences for a real-time DAG-based workflow at runtime. At a given time during the workflow execution, the ECDF predicts the likelihood that a given number of event instances occur on the cluster resources with a certain confidence and within a given window of time. This likelihood information can be used by the runtime system to take important scheduling decisions, such as scheduling a second queue of back-end tasks on the idle resources, thus increasing the resource utilization of the whole non-dedicated cluster. The main contributions of our work are:

1) A tool that uses frequency analysis to track, study, and predict event occurrence rates for real-time DAG-based workflow executions within a given window size and with a certain confidence.
2) An empirical validation of the tool for a large set of DAG-based workflows and for clusters with different numbers of available resources.

Results in this paper yield an average of 94% prediction hit-rates for the DAG-based workflows considered in our empirical study. Although we apply our analysis to Paje traces, our tool can be easily extended to analyze event traces in other formats with minor changes to the tool's initial parser module.

The rest of the paper is organized as follows. Section II provides an overview of StarPU used to generate DAG-based workflows and the Paje execution traces. Section III describes in detail our tool and the methodology. Section IV presents tests on multiple DAG-based workflows and our analysis results. Section V summarizes the state of the practice and related work. Section VI concludes the paper and lists future work directions.

II. Background

In this section, we present background on the StarPU runtime system and the Paje execution traces used in this work.

A. StarPU

StarPU is an open-source software that supports the scheduling of computational tasks across the nodes of a non-dedicated cluster using task-based paradigms [3]. Aspects such as scheduling policies or resource allocation in StarPU are outside the scope of this paper. Still we provide a short overview of the runtime system because it is used in this work to generate the DAG-based workflows on non-dedicated clusters that we use for benchmarking and predictions. A StarPU task is a module or kernel that has been written using a heterogeneous programming paradigm such as OpenCL [4]. StarPU supports both single and multiple nodes in a cluster. Each cluster node considers its own sub-problem of the overarching application, meaning that each node uses StarPU on a sub-graph of the entire workload graph. StarPU decides on which resources a task runs. When tasks are repeated multiple times, StarPU either estimates their execution time, or profiles a single task and uses its execution time as a reference for other task executions. In these circumstances, our tool can provide StarPU with valuable insights into likely event occurrences across all nodes of the cluster that the runtime systems can use to transform speculation into accurate predictions.

B. Paje Execution Traces

StarPU returns execution traces including resource utilization, communication among resources, and information on resource blocking and/or dependencies to the user in the Paje format. Specifically, the Paje format is a trace file format that is self-defined, textual, and generic to describe the behavior of computer programs that are executed in distributed systems [2]. This format consists of a single file collecting the information of multiple nodes in the non-dedicated cluster. The file has three components: event definitions, events, and comments. The event definitions describe the parameters for different types of events (e.g., running, idle, communication) while the events themselves describe the values for each of the parameters (e.g., event type, resource ID, start time). Comments start with the symbol #. Each trace maps multiple events to a resource (e.g., a node); and each event is associated with a certain amount of time.

The trace format can be either generated directly or converted from other conventional trace formats. For example, the StarPU scheduling system uses the Paje format to properly capture the events of distributed systems to profile the scheduling. Other trace formats such as .tau and .otf can be converted to Paje using programs such as tau2paje or otf2paje [1]. This property of the Paje trace format makes our tool amenable for analyzing other trace formats, thereby broadening its scope for use. In this research, we use ViTE trace explorer [5] to visualize the Paje traces, when the number of resources makes the visualization process feasible.

III. Methodology

In this section, we describe the bipartite tool and its components. While the tool can be used to model a broad range of events including computation, communication, and idle events, we specifically focus on idle event occurrences and their predictions as our proof-of-concept. Prior to describing the tool framework, we show how we adapt the cumulative frequency analysis technique for event likelihood estimations.

A. ECDF-based Frequency Analysis

Cumulative frequency analysis is a popular statistical technique whose frequency information is often used to devise preventive strategies. In this work, we adapt the technique to predict event occurrences in DAG-based workflows. Specifically, we extend the cumulative frequency analysis theory to model commonly occurring events such as computation, communication, and idle resource instances on non-dedicated clusters. The frequency analysis is applied to a dataset containing the numbers of event occurrences in a given analysis window (i.e., $x_0, x_1, ..., x_{N-1}$, where N is the number of samples in the analysis window). We define the cumulative frequency CF_r as the frequency with which a number of event occurrences x takes on a value less than or equal to x_r. The cumulative frequency is used to obtain the probability P_C for the value x to be smaller than x_r. The probability is given by the equation:

$$P_C = \frac{CF_r}{N} \tag{1}$$

To automate the probability analysis, we use the *ecdf* function from MATLAB that transforms the dataset of the numbers of event occurrences into the empirical cumulative distribution function (ECDF). The ECDF can be used to predict the likelihood of a given number of resource instances undergoing a specific event in the future. As discussed in detail in the following sections, ECDF is used in our work for both the benchmarking and prediction phases in the bipartite tool. The two phases rely on comparing two ECDFs obtained from two successive modeling windows using Kolmogorov-Smirnov (KS) test [6]. The null hypothesis H_0 for the KS test states that the samples are drawn from the same distribution. The two-sample KS test results in a miss if the null hypothesis is rejected at 5% significance level (i.e., the two CDFs are not similar), a hit otherwise (i.e., the two CDFs are statistically similar).

B. Tool Framework

As shown in Figure 1, the framework of our bipartite tool consists of two phases: a benchmarking phase and prediction phase. The benchmarking phase builds a characterization of a

[1] http://paje.sourceforge.net/tracers.html

large suite of DAG-based workflows when executed on different numbers of resources on a non-dedicated cluster, whereas the prediction phase facilitates the likelihood prediction of an event occurrence on a certain set of resources as the execution of a real-time workflow evolves.

In the benchmarking phase, the Paje traces of benchmarking DAG-based workflows are supplied to the benchmark core of our tool as seen in Figure 1. These traces describe the occurrences of computation, communication, and idle events on each resource of the cluster and are obtained from the execution of scientific DAG-based workflows on a non-dedicated cluster using StarPU as the runtime system. The benchmark core uses ECDF-based frequency analysis technique described in Section III-A to generate an optimal window-size model for the prediction of future event occurrences.

In the prediction phase, a DAG-based workflow is in execution and its runtime Paje trace is input to a prediction core which uses the optimal window-size model to generate the ECDF for predicting the likelihood a certain number of resources will experience a given event occurrence in a given interval of time as large as the modeled window size. For example, the prediction core can answer a query such as What is the likelihood that 25% of resources are idle in the next time window of length 10 seconds? In general, the likelihood information is used by the runtime system to optimize the resource utilization among several applications executed on the non-dedicated cluster as described in more detail in Section III-D.

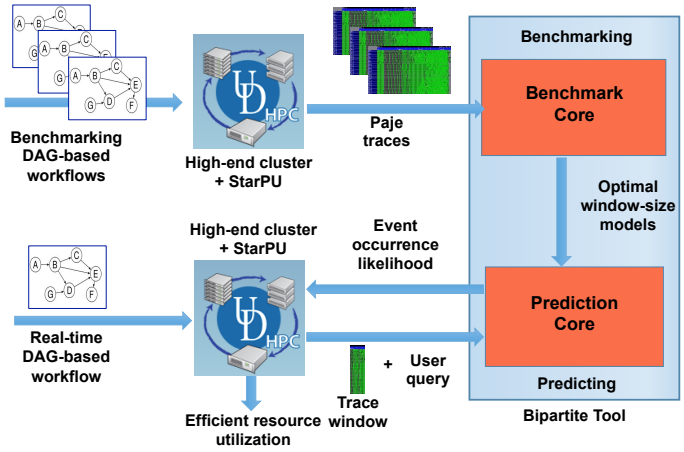


Fig. 1: Bipartite tool and its two-phase framework including the benchmarking phase and the prediction phase.

C. Benchmarking Phase

The benchmarking phase of our tool analyzes a set of Paje traces to generate a suite of optimal window sizes suitable for likelihood predictions, where the size is measured in number of sampled event occurrences. The traces are obtained from the execution of DAG-based workflows of varying task sizes (i.e., with a number of tasks ranging from 1000 to 4000) on a high-end, non-dedicated cluster with varying number of resources (i.e., with a number of nodes ranging from 8 to 256). The benchmarking DAGs are homogeneous, meaning all the tasks execute the same type of computation. For the sake of completeness, we consider a large and diverse suite of DAG-based workflows that is described in detail in Section IV-A.

The implementation of the benchmarking phase is modular as showed in Figure 2. It consists of two stages, namely a surface builder stage and a surface analyzer stage. The surface

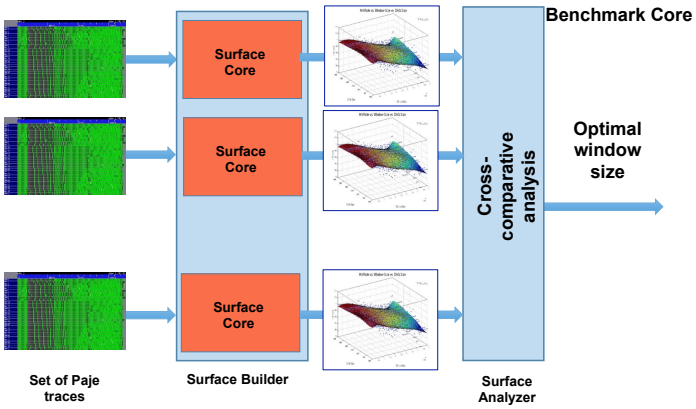


Fig. 2: The benchmarking phase and its two modular stages, i.e., the surface builder stage and the surface analyzer stage.

builder runs multiple instances of the surface core, each one performing an ECDF-based frequency analysis on one of the Paje traces. Figure 3 shows the components constituting the surface core. The Paje trace first enters the event occurrence generator. The generator parses the trace and counts the number of event occurrences, creating an event occurrence signal. To discretize the trace continuum into sampled event occurrences, we refer to the Nyquist frequency theory [7] and set the sampling frequency equal to twice the inverse of the minimum time elapsed between two successive event changes.

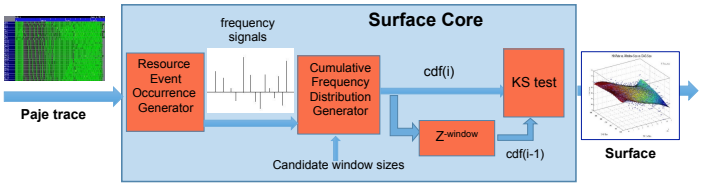


Fig. 3: The surface core with its steps that transform a Paje trace into a parametric surface.

The event occurrence signal is filtered using a suite of defined candidate window sizes and is supplied to the cumulative distribution function (CDF) generator to build an ECDF model. It should be noted that the window size in this work is defined as the number of event occurrence signal samples. For instance, a window size of 5000 collects 5000 samples of the event occurrence signal. The accuracy of the candidate window size is determined by performing the statistical Kolmogorov-Smirnov (KS) test on two successive, non-overlapping CDFs (i.e., cdf_{i-1} and cdf_i) from two consecutive sampling windows. If the two CDFs are statistically similar, the KS test results in a prediction hit for the window size; otherwise it is a prediction miss. For each window in the suite of candidate

window sizes, we traverse the entire trace using a sliding factor equal to the window size to determine the trace's prediction hit-rate, as shown in Figure 4. The hit-rate is defined as the

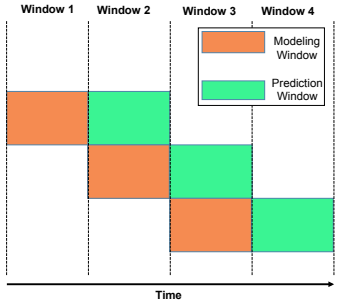


Fig. 4: Modeling and prediction windows over the trace lifespan.

percentage of prediction hits observed for the complete trace traversal. By performing the above process, the tool builds sets of 3-D samples, each showing the hit-rate with respect to the window size and DAG-based workflow size for a given cluster configuration (i.e., with a given number of resources). Each set of sampled 3-D points is used to build a hit-rate surface. Figure 5 shows an example of surface generated by the surface core with our testing workflows. The surface shows the hit-rate with respect to different window sizes (i.e., 1000 to 50000 samples) and DAG-based workflow sizes (i.e., from 1000 to 4000 tasks) for a cluster of 64 modes.

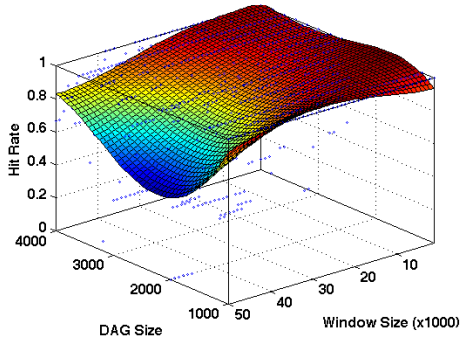


Fig. 5: Example of 3-D surface predicting the hit-rate with respect to different window and DAG-based workflow sizes for a cluster of 64 nodes.

The 3-D surfaces from the surface builder are then input to the surface analyzer that performs cross-comparative analysis to model the suite of optimal window sizes for our runtime predictive analysis. The 3-D surface plots are interpolated into a single 3-D surface by combining all of the data points from the individual surfaces. The interpolation is performed using MATLAB *fit* function that fits a polynomial surface. We use fourth-order polynomial surfaces to prevent over-fitting the models with too high a polynomial order or under-fitting the models with too low a polynomial order. For a given cluster configuration, the interpolated surface is cut at the 95% hit-rate to identify the modeled window sizes with 95% prediction hits for each DAG-based workflow. The suite of optimal window sizes, comprising of model windows for all resource configurations and DAG-based workflows, is supplied to the prediction phase for likelihood estimations.

D. Prediction Phase

The prediction phase uses the optimal window-size model from the benchmarking phase to perform ECDF-based frequency analysis on the real-time Paje execution traces. This phase is operated by the prediction core shown in Figure 6 that predicts the likelihood of certain event occurrences when using a given set of resources. The inputs to the prediction core are: (1) a real-time trace frame of length equal to the optimal window size obtained by the benchmarking phase and (2) a user query. The event occurrence generator samples the incoming trace in the selected model window and produces the event occurrence signals for the trace frame. The signals are used by the CDF generator to generate the ECDF. Based on the user query, the ECDF provides the likelihood prediction for the event occurrence. The model building and predictions are performed on frame-by-frame basis.

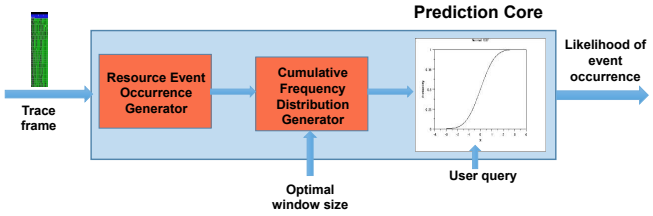


Fig. 6: The prediction core with its inputs (i.e., the trace frame and the user query) and its outputs (i.e., the likelihood an event occurrence under a given resource configuration).

IV. Tool Testing

This section presents the testing results for the bipartite tool. In Section IV-A, we present the characterization of the synthetic DAG-based workflows studied in this research. In Sections IV-B and IV-C, we provide the testing results for the *benchmarking* and *prediction* phases, respectively.

A. DAG Characterization and Associated DAG-based Workflows

To cover a diverse set of workflows, we use a dataset of 1600 DAGs generated as part of the work presented in [1]. The DAGs have different properties (i.e., different number of tasks, number of edges, maximum outdegree, and graph level). The nature of the DAGs' properties across the chosen dataset results in diverse execution profiles, making the DAGs suitable for benchmarking and prediction studies. Specifically, the diverse computational density in DAG-based workflows is captured by the number of tasks in the DAGs, whereas the diverse dependencies and level of parallelism are explained by the number of edges and outdegree of tasks, respectively. The maximum graph level from source tasks (roots) to sink tasks (leaves) highlights the critical path in the workflow. The above mentioned graph properties for characterization are motivated by the work of Ostermann and co-workers [8]. The 1600 DAGs are organized into 16 groups, each containing 100 DAGs.

TABLE I: Characterization of DAGs used to build the DAG-based workflows in the benchmarking phase.

#Group	#Tasks	#Edges	Max. Outdegree	Graph Level
1000	1001-1006	1724-2011	7-11	28-70
1200	1201-1208	2137-2417	7-15	30-69
1400	1401-1402	2501-2755	8-11	26-65
1600	1600-1603	2878-3112	7-11	41-94
1800	1800-1814	3278-3617	9-15	38-86
2000	2000-2021	3490-3877	8-11	28-80
2200	2200-2222	3943-4285	9-13	36-132
2400	2400-2408	4343-4733	8-14	42-108
2600	2602-2609	4598-5038	8-11	54-93
2800	2801-2813	4965-5553	10-18	32-117
3000	3000-3004	5248-6096	9-17	37-125
3200	3200-3210	5735-6296	8-12	27-114
3400	3400-3425	6084-6865	9-16	40-128
3600	3600-3605	6615-7032	10-15	41-97
3800	3800-3802	7004-7310	10-13	49-109
4000	4000-4011	7368-7876	9-12	50-93

TABLE II: Characterization of DAGs used to build the DAG-based workflows in the prediction phase.

#Group	#Tasks	#Edges	Max. Outdegree	Graph Level
1000	1000-1007	1782-1922	9-13	30-46
1200	1200-1201	2146-2333	8-15	28-66
1400	1400-1412	2552-2802	11-13	30-45
1600	1600-1607	3003-3031	8-11	40-58
1800	1801-1804	3284-3365	9-11	33-68
2000	2000-2002	3724-3959	10-11	65-124
2200	2202-2205	3864-4377	8-10	32-56
2400	2400-2404	4410-4619	11-13	47-52
2600	2600-2603	4657-4887	9-11	46-125
2800	2801-2815	5182-5552	11-13	58-78
3000	3000-3004	5645-5890	10-10	49-91
3200	3201-3202	5880-5935	9-13	42-87
3400	3401-3403	6361-6704	11-14	53-70
3600	3601-3603	6542-7014	9-13	48-104
3800	3801-3804	6967-7186	11-12	69-92
4000	4000-4001	7514-7727	11-11	43-94

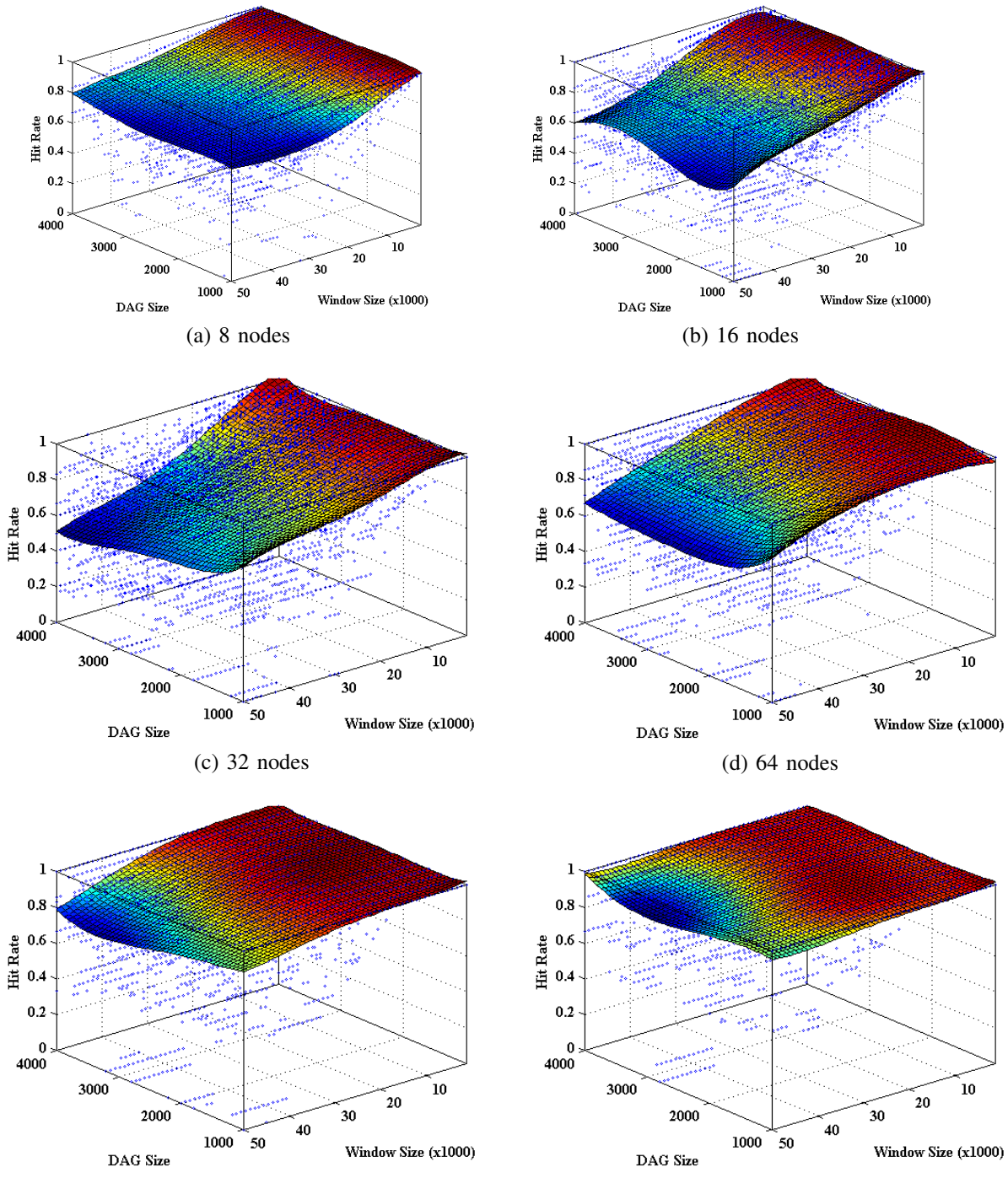
The groups have consecutively incremental numbers of tasks ranging from 1000 to 4000 tasks (i.e., 1000, 1200, 1400 and so on). In other words, the first group of 100 DAGs has $\approx$ 1000 tasks per DAG, the second group of 100 DAGs has $\approx$ 1200 tasks per DAG, and so on.

To generate the window-size models in the benchmarking phase, we build eight benchmarking DAG subsets from the entire DAG dataset. Each subset consists of 16 DAGs; each one of these 16 DAGs is randomly selected from one of the 16 groups defined above. Thus the 16 DAGs of a subset have consecutively incremental numbers of tasks (i.e., 1000 tasks, 1200 tasks, and so on). Table I presents the characterization of the DAGs across the eight subsets in terms of their properties (i.e., number of tasks, number of edges and- maximum outdegree, and graph level). As the number of tasks increases from 1000 to 4000, the number of edges and the maximum outdegree both increase, while the graph level remains similar. We use the eight benchmarking DAG subsets, each with 16 DAGs, for training the window size in the benchmarking phase. Specifically, for each subset we build the DAG-based workflows and the associated Paje traces on multiple StarPU-based non-dedicated clusters with different resource configurations (i.e., ranging from 8 to 256 nodes).

To validate the modeling process and assess the prediction accuracy in the prediction phase, we build three additional subsets of DAGs from the entire DAG dataset. Similar to the DAG subsets used in the benchmarking phase, each subset used for the prediction phase consists of randomly selected DAGs with consecutively incremental numbers of tasks. Table II presents the characterization of the three subsets of DAGs in terms of their properties. For the sake of accuracy, the eight subsets used for benchmarking and the three subsets used for prediction are disjoint. As in the benchmarking phase, we use the three subsets to build the DAG-based workflows and the associated Paje traces on multiple StarPU-based non-dedicated clusters, each using a variable number of resources. Contrary to the DAG-based traces used for the benchmarking phase that are analyzed statically for building the window-size models, traces from the three DAG subsets are analyzed at runtime for predicting the likelihood of event occurrences.

B. Building the Window-size Models: Benchmarking Phase

We build window-size models using the eight subsets of DAG-based workflows described above. We consider six cluster configurations with 8, 16, 32, 64, 128, and 256 nodes. For each subset of DAGs, a surface core in the surface builder analyzes the associated Paje traces that are generated when the DAG-based workflows are executed on the cluster configurations. With this information, the surface core builds the surface plots of the hit-rate with respect to window size and the number of DAG tasks for the six cluster configurations. The multiple surfaces generated by the surface builder using eight DAG subsets are provided to the surface analyzer for interpolation. Figures 7.a-f show the resulting interpolated surfaces of our DAG subsets for the specific case of idle resource events. In the figures, the x-axis and y-axis are the window size (in number of event occurrence samples) and the number of DAG tasks, respectively. The z-axis is the hit-rate (i.e., the percentage of prediction hits observed for the entire trace traversal). As shown in Figures 7.a-b, the surfaces are relatively flat for small cluster configurations (i.e., 8 and 16 nodes). A large flat surface means that for increasing window size and number of tasks in the DAGs, the probability of matching expected event occurrences and observed event occurrences is high. From the modeling point of view, these two surfaces imply large window sizes. From the execution point of view this can be explained as follows: when using small cluster configurations, the nodes are easily saturated with tasks. The associated uniform trace behavior with respect to computations supports a larger modeled window size that is still able to capture the trace characteristics with high prediction accuracy for the next time frame. This is not the case for moderate size clusters (i.e., with 32 and 64 resources) as shown in Figures 7.c-d, where the surfaces rapidly drop below the ideal rate of one as both window size and number of DAG tasks increase. Unlike the aforementioned cluster configurations, relatively large cluster configurations (i.e., 128 and 256 nodes) have flatter surfaces due to uniform trace behavior with respect to idle events. For these two configurations, the resources may remain idle for



(a) 8 nodes (b) 16 nodes

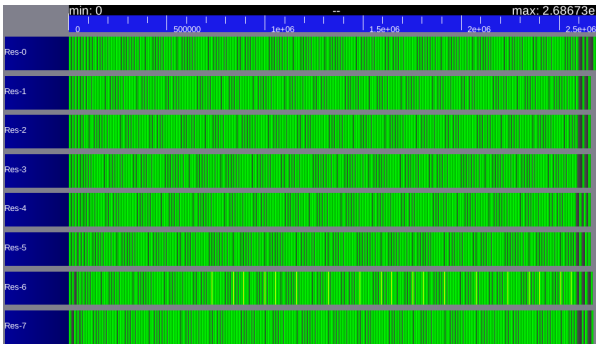
(c) 32 nodes (d) 64 nodes

(e) 128 nodes (f) 256 nodes

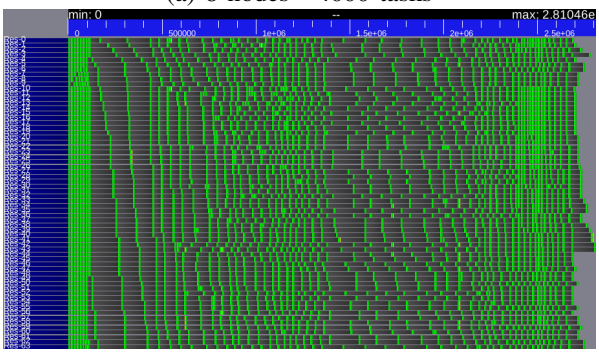
Fig. 7: Interpolated surface plot from surface analyzer for a cluster of (a) 8 resources; (b) 16 resources; (c) 32 resources; (d) 64 resources; (e) 128 resources; and (f) 256 resources.

extended periods due to insufficient computational density of the DAG-based workflows.

An empirical validation of these observations can be obtained by looking at the Paje traces. Figure 8.a shows a uniform trace behavior of an 8-node cluster configuration for a DAG-based workflow of 4000 tasks. On the other hand, Figure 8.b shows a patch-like trace behavior of a cluster with 64 nodes and for the same 4000-task DAG-based workflow. Unlike in the 8-node cluster, the nodes in larger 64-node cluster remain idle due to fewer computational tasks ready for execution at any time on the larger number of available resources. The idle periods may be further exacerbated due to limited parallelism (small node outdegree) in the DAG-based workflows. In gen-



(a) 8 nodes - 4000 tasks



(b) 64 nodes - 4000 tasks

Fig. 8: Uniform trace behavior of 8-node configuration executing 4000-task workflow (a) and patch-like trace behavior of 64-node configuration executing the 4000-task workflow (b).

eral, a comparable number of idle and computation resource instances in large cluster configurations result in patch-like trace behaviors (i.e., the computation and idle events occur in patches). Therefore, the model building window sizes become relatively short to sufficiently capture the variable behavior in the traces. Unfortunately, the visual analysis of Paje traces is not effective for larger cluster configuration; already with 128 nodes, the visual analysis is inconclusive. This makes our tool desirable in applications of large non-dedicated clusters.

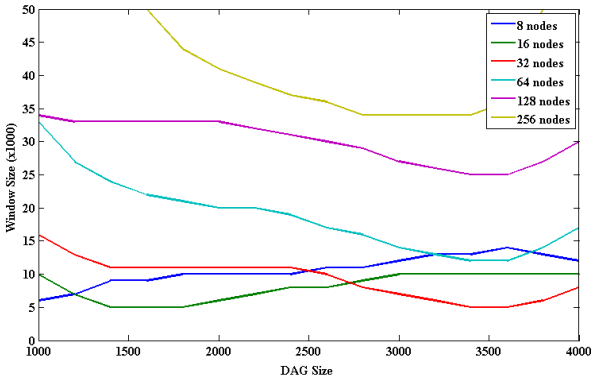


Fig. 9: Window size model for different DAG sizes and different cluster configurations.

The interpolated surfaces are used to model the window sizes as the number of tasks in the DAG increases. To do so, the surface analyzer cuts the surfaces at 95% prediction hit-rate to yield an appropriate window size for a given size of DAG-based workflow executing on a given cluster configuration. Based on this interpolated surface cutting, Figure 9 shows the variation of window sizes with respect to the DAG size and cluster configuration for the specific case of idle resource events, identifying a clear modeling pattern of the window size across DAG-based workflows as the number of resources grows. To draw inferences from this graph, we define three states of an execution trace: a *uniform idle* state in which the trace exhibits a uniformly idle pattern due to large number of idle resources; a *patch* state in which the trace exhibits a patch-like behavior due to comparable idle and computation occurrences; and a *uniform computation* state in which the trace exhibits a uniform computation pattern due to large number of computation events. For both the uniform idle and uniform computation states, the window size is expected to be large and relatively constant with respect to the workflow size. The window size decreases as the patch-like behavior of the trace increases. As the workflow size increases, the trace behavior is expected to shift from a uniform idle state to a patch state followed by a uniform computation state. As seen in Figure 9, the window size is initially large and constant for a 256-node cluster configuration, implying uniform idle state. The DAG-based workflows with less than 1600 tasks are not sufficiently dense to saturate this configuration, resulting in large number of idle events. However, the window size decreases for workflow sizes beyond 1600 tasks, indicating the transition of traces into the patch state. Moderate size cluster configurations (i.e., 32, 64, and 128) have a downward trend for the window size, indicating patch state of the execution

traces for the given workflows. The smaller cluster configurations including 8 and 16 nodes have an upward trend: their traces' behavior gradually shifts from patch state to uniform computation state as the workflow size increases, which causes the window size to increase. With our proposed bipartite tool, it is straightforward to analyze and model DAG-based workflows even beyond 4000 tasks. The window-size models obtained with the above process are used for the validation phase, which is discussed in the next section.

C. Validating the Window-size Models: Prediction Phase

To assess the accuracy of our window-size model to predict the likelihood of event occurrences (idle events as our specific case), we use the three DAG subsets described in Section IV-A. As shown in Figure 1, the prediction phase analyzes a real-time trace window to build an ECDF model using the optimal window-size models obtained in the benchmarking phase. For validation purposes, the ECDFs of two successive real-time windows $i-1$ and i are compared using the KS test and prediction hit/miss are recorded. Here, a prediction hit means that cdf_{i-1} obtained by analyzing the previous window $(i-1)$ satisfactorily predicts the likelihood of event occurrences in the current window (i). This process is continued for the real-time traces' lifespan to yield the overall prediction hit-rate for that particular trace. Figures 10.a-c show the hit-rate versus the DAG size across all the cluster configurations for all the execution traces from the three DAG subsets. Barring a few outliers, the hit-rates are observed 90% and above for most of the test cases. The outliers are due to variations in DAG properties across the 8 benchmarking DAG subsets. These outliers can be mitigated by increasing the number of benchmarking DAG subsets, thereby capturing the statistics more effectively. Using our methodology, we observed average prediction hit-rates of up to 94% across all the DAG subsets, establishing the efficacy of our tool to predict the likelihood of event occurrences on a given number of resources.

V. Related Work

There has been significant research and commercial effort to improve the performance of cluster systems, but fewer efforts include real-time predictive analysis of workflows to improve the performance dynamically. To the best of our knowledge, none of these use frequency analysis techniques on a large and diverse set of DAG-based workflows to predict events occurrences within optimal window sizes as described in our work.

More traditional approaches monitor the cluster behavior with popular tools such as Ganglia [9] and Supermon [10], lacking the capability to dynamically tune the cluster performance and leaving the prediction of future event occurrences to the system administrator. Trace analysis highlights the cluster behavior for applications execution and helps to understand the performance of scheduling algorithms. Tools such as Vampir [11] and Tau [12] provide various methods for analyzing traces of applications on distributed systems. Work in [13] analyzes traces from cloud systems to identify the performance of different scheduling algorithms. While the scheduling aspects targeted in this work are important, our trace analysis is directed towards a more dynamic runtime approach in terms of improving performance. Like Ostermann and co-authors in [8], we also rigorously characterize the DAG-based workflows. However, the work in [8] characterizes workflows on grid systems only to identify the categories of multiple workflows for post mortem analysis, while we add an additional dynamic analysis component that can be easily integrated into runtime systems other than StarPU.

The work that most resembles to our contributions is that of Sahoo et al. [14]. This work involves the idea of "self-healing" systems that predict potential problems through the analysis of real-time execution data of cluster systems. In [14] the prediction targets critical events that lead to system failure. In our work, we predict a broader range of event occurrences by using a diverse set of DAG-based workflows, with the scope of increasing the overall resource utilization rather than just predicting the failure occurrences. We also provide a comprehensive tool and a window-size model to adjust the granularity of the predictions based on the number of resources available.

VI. Conclusions

This paper presents a general, scalable, and effective method to analyze the execution traces of large DAG-based scientific workflows on non-dedicated shared clusters managed by a popular runtime system such as StarPU. The tool employs a cumulative distribution-based frequency analysis technique on Paje execution traces to study the occurrences of events including idle, computation, and communication events. The tool offers two primary functionalities. First, it systematically benchmarks the DAG-based workflows in a single pass approach and collects the statistics to estimate modeling window sizes. Second, it uses the modeling windows to predict the likelihood of future event occurrences. With our workflow benchmarking techniques, we observe average likelihood prediction hit-rate up to 94% using several diverse DAG test cases. Using our approach of workflow benchmarking and event predictions, runtime systems can make important scheduling decisions when it is time to assign resources to applications, improving the overall system utilization. Future work will include the study of theoretical distributions to model the resource events. To broaden the scope of our tool, future work will also include integrating trace format converters (e.g., .otf and tau) to Paje format.

ACKNOWLEDGMENTS

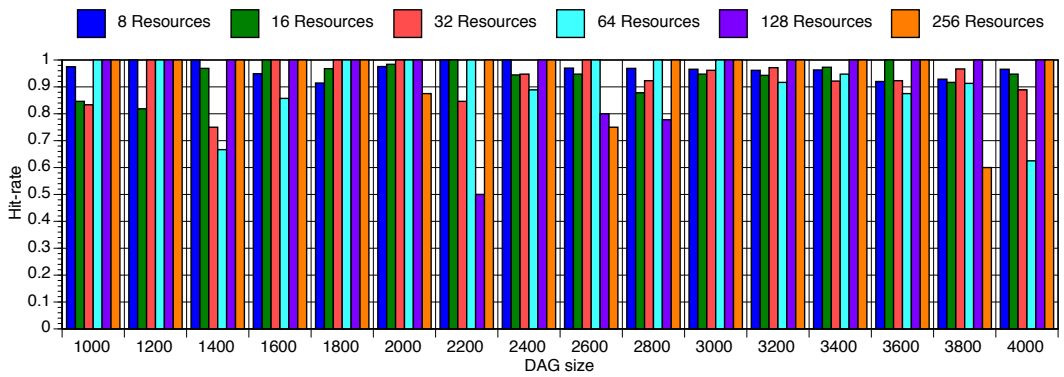
This work is supported in part by the NSF grant NSF #1217812 and the USAF, AFRL Program. The authors want to thank Drs. Eric Kelmelis and John Humphrey from EM Photonics for their support and Scott Roche for the DAG dataset.

References

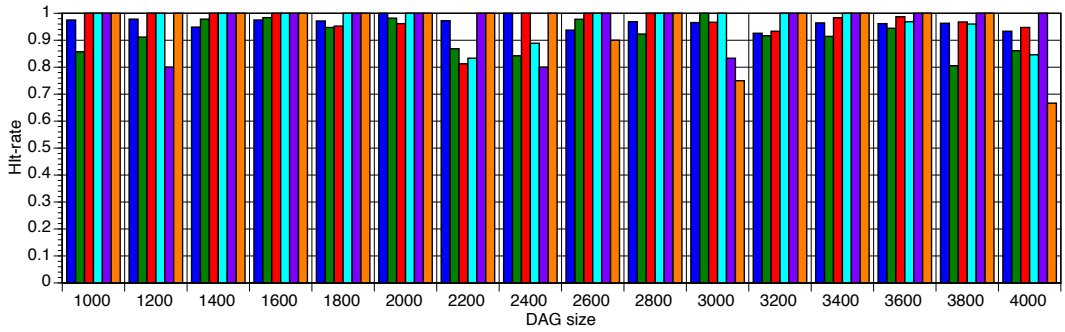
[1] Gennaro Cordasco and Arnold L Rosenberg. Area-maximizing schedules for series-parallel DAGs. *Lecture Notes in Computer Science*, 6272:380–392, 2010.

[2] Jacques Chassin de Kergommeaux, Benhur Stein, and Pierre-Eric Bernard. Pajé, an interactive visualization tool for tuning multi-threaded parallel applications. *Parallel Computing*, 26(10):1253–1274, 2000.

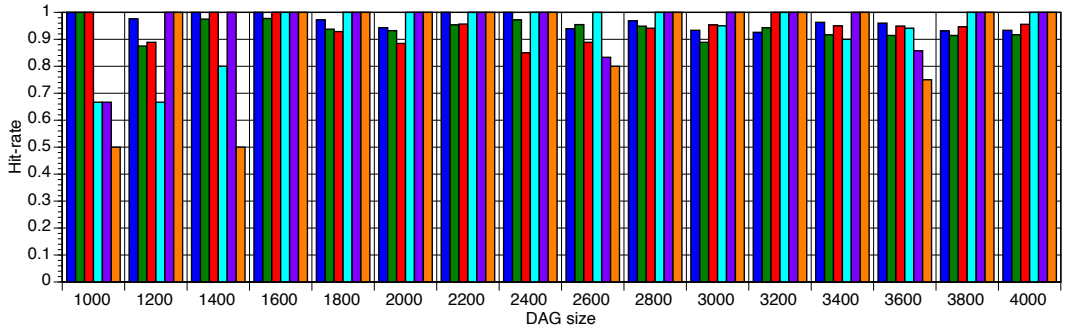
[3] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198, 2011.



(a) Case study I



(b) Case study II



(c) Case Study III

Fig. 10: Validation of the window-size models using three randomly generated case studies of 16 DAG-based workflows each.

[4] John E Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science & Engineering*, 12(3):66, 2010.

[5] Hao-wei Hsieh and Frank M Shipman III. VITE: A visual interface supporting the direct manipulation of structured data using two-way mappings. In *Proc. of the 5th International Conference on Intelligent User Interfaces*, 2000.

[6] Hubert W Lilliefors. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402, 1967.

[7] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IEEE*, 86(2):447–457, 1998.

[8] Simon Ostermann, Radu Prodan, Thomas Fahringer, Alexandru Iosup, and Dick Epema. A trace-based investigation of the characteristics of Grid workflows. *From Grids to Service and Pervasive Computing*, pages 191–203, 2008.

[9] Matthew L Massie, Brent N Chun, and David E Culler. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[10] Matthew J Sottile and Ronald G Minnich. Supermon: A high-speed cluster monitoring system. In *Proc. of the IEEE International Conference on Cluster Computing*, 2002.

[11] Wolfgang E. Nagel, Alfred Arnold, Michael Weber, Hans-Christian Hoppe, and Karl Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12:69–80, 1996.

[12] Sameer S Shende and Allen D Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[13] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of the Third ACM Symposium on Cloud Computing*, 2012.

[14] Ramendra K Sahoo, Adam J Oliner, Irina Rish, Manish Gupta, José E Moreira, Sheng Ma, Ricardo Vilalta, and Anand Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proc. of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.