Predicting Performance Variability

Mohammed Baydoun
Department of Computer Science
American University of Beirut
Beirut, Lebanon
mb57@aub.edu.lb

Mohammad Sonji
Department of Computer Science
American University of Beirut
Beirut, Lebanon
mms158@mail.aub.edu

Pedro Bruel
Hewlett Packard Enterprise
Milpitas, CA, USA
bruel@hpe.com

Dejan Milojicic
Hewlett Packard Enterprise
Milpitas, CA, USA
dejan.milojicic@hpe.com

Eitan Frachtenberg
Hewlett Packard Enterprise
Milpitas, CA, USA
eitan.frachtenberg@hpe.com

Izzat El Hajj
Department of Computer Science
American University of Beirut
Beirut, Lebanon
izzat.elhajj@aub.edu.lb

***Abstract*—As computing systems grow increasingly more complex, application performance on these systems is becoming more variable and less deterministic. Scalar performance summaries such as mean or median run time do not adequately reflect the true behavior of an application that can only be gleaned from the complete performance distribution. However, measuring the distribution of an application's performance on a system requires running the application many times on that system, which can be an expensive process. To address this challenge, we aim to answer the question: Can the performance distribution of an application on a system be predicted by learning from other representative applications?**

We aim to answer this question in the context of two use cases: predicting the performance distribution of an application on a system from a few runs of that application on the system, and predicting the performance distribution of an application on a system from a measured distribution of that application's performance on a different system. To this end, we measure the performance distribution of a large set of representative benchmarks and use that information to train prediction models that predict the performance distribution of new applications. We consider different alternatives for formulating the prediction problem as well as different types of prediction models. Our evaluation compares these alternatives to identify the best formulation and model to use for each use case, and shows that many application performance distributions can be predicted with reasonable accuracy.

I. Introduction

Modern computing systems are becoming increasingly more complex, which hinders reliable performance evaluation. Sophisticated hardware optimizations, heterogeneity, system software, middleware, language runtimes, concurrency, and even environmental factors all present sources of nondeterminism that impact performance. It is increasingly the case that the same code running on the same system with the same input parameters may exhibit different performance with each run, a phenomenon known as performance variability [1].

Because of performance variability, analyzing an application's performance on a system by taking the mean or median of a few runs does not accurately reflect the true behavior of the application on the system. A faithful performance analysis of an application on a system requires observing the full distribution of the application's performance, which also gives insight about performance modes and performance tails, features that are not captured by a scalar performance summary. For example, Figure 1(a) shows the performance distribution of the SPEC OMP [2] benchmark 376 measured from 1,000 sample runs (see Section IV for experimental methodology). It is clear that the performance distribution has two modes, with the larger mode being the faster mode.

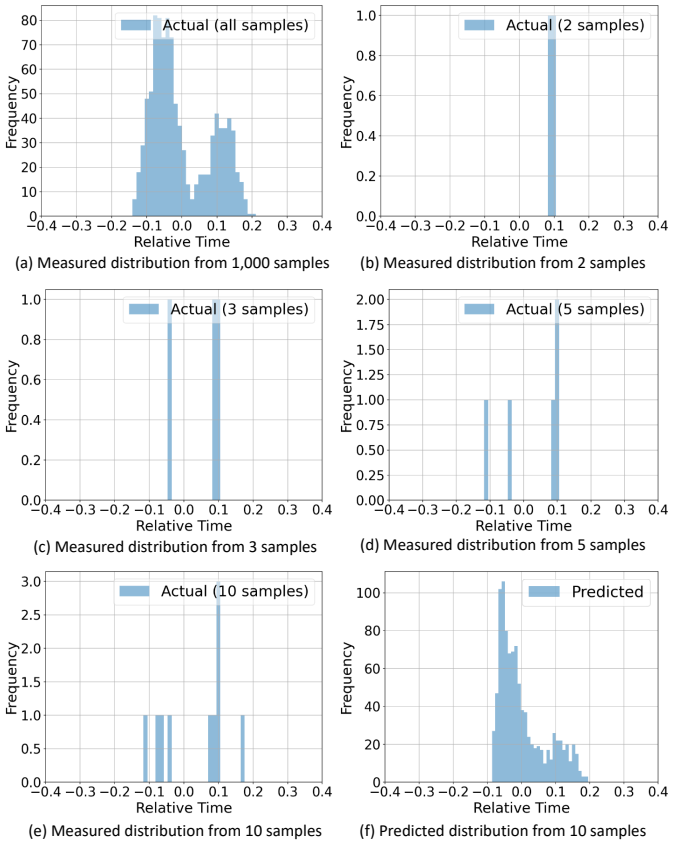

Fig. 1. Measured and predicted performance distributions of the SPEC OMP [2] benchmark 376 (see Section IV for experimental methodology)

The mean value is in between the two modes, closer to the larger mode, which is not entirely representative of how the benchmark performs.

Evaluating performance variability requires careful experimental design and statistical analysis which is difficult [1], [3], [4]. Inadequate methodology, such as measuring too few runs or using reductive summary statistics, can lead to wrong or misleading interpretations [5], which is regrettably not uncommon [6]. For example, Figures 1(b,c,d,e) show the performance distribution of the SPEC OMP benchmark 376 as measured from 2, 3, 5, and 10 samples. Clearly, the distributions measured from a small number of samples are not representative of the actual performance distribution shown in Figure 1(a). On the other hand, an overly conservative evaluation based on an excessive number of runs such as that in Figure 1(a) can lead to wasted time and resources. While some adaptive techniques assist in finding a good compromise between too many samples and too few [7], even collecting a minimally adequate number of samples can be too expensive for some applications.

To address this challenge, we aim to answer the question: Can the performance distribution of an application on a system be predicted by learning from other representative applications? We aim to answer this question in the context of two use cases. In the first use case, we aim to predict the performance distribution of an application on a system from just a few runs of the application on that system. This use case represents the scenario where a user may need to frequently inspect the application's performance distribution while optimizing it to select what optimization to apply, or to assess the fitness of an application for being used in latency-sensitive contexts. In the second use case, we aim to predict the performance distribution of an application on a system from a performance distribution of the application measured on another system. This use case represents the scenario where a user may be interested in acquiring a new system and would like to anticipate the suitability of that system for the user's application. For both use cases, we consider different alternatives regarding how to represent an application's profile as input features, how to represent an application's performance distribution for prediction purposes, and what kind of predictive model to use.

To evaluate these use cases, we measure the performance distribution of a large set of representative benchmarks and use that information to train models that predict the performance distribution of new applications. Our evaluation compares the different alternatives discussed to identify the best representations and model to use for each use case. We also show that many application performance distributions can be predicted with reasonable accuracy. For example, Figure 1(f) shows the performance distribution of the SPEC OMP benchmark 376 that we predict from just 10 measured samples. While not an exact match to the measured distribution in Figure 1(a), the predicted distribution in Figure 1(f) displays sufficiently accurate information about the number of modes as well as their relative locations and sizes. Our results can be further improved and generalized by increasing the number and diversity of benchmarks used for collecting training data, as well as using a larger number of systems for evaluation including heterogeneous systems with accelerators.

The rest of this paper is organized as follows. Section II surveys previous works on accounting for, managing, and predicting performance variability. Section III describes our use cases and alternatives for formulating the prediction problem. Section IV presents our experimental methodology, and Section V evaluates each use case. Finally, Section VI concludes and discusses future work.

II. Related Work

Modern computing systems hide many complexities under multiple layers of abstraction. These complexities, and the interactions between them, make performance less deterministic and results in performance variability [8]. This nondeterminism may come from: CPUs due to dynamic clock speed, caches, or speculation [9]; heterogeneity due to the imbalance between components [10]; system software and middleware due to power-saving trade-offs, schedulers, garbage collection, or just-in-time optimizations [11], [12]; concurrency due to unpredictable execution orderings of parallel tasks [13], [14]; and even environmental factors such as datacenter temperature.

The existence of performance variability poses a challenge for reproducibility [14]–[17]. For this reason, numerous works have proposed evaluation methodologies that take variability into account [3], [6], [13], [18], [19]. These methodologies include using sufficient repetitions and sound statistical analyses [20], resampling to estimate the sample size that results in sufficient confidence [1], integrating reproducibility into DevOps methodology [4], and using quantile regression to compare distributions [21].

A number of works aim at modeling variability for the purpose of managing it with different objectives. These objectives include reducing variability via cache-aware page allocation [22], trading off variability and performance [23], scheduling [24], and anomaly detection [25]. Our work aims at predicting performance variability, rather than modeling it from a measured distribution, which can also be used to assist these same variability management objectives.

Many works aim at predicting the performance of an application while accounting for performance variability [9], [11], [12], [26]–[29]. However, these works focus on predicting large-scale variability caused by interference from other users such as in cloud environments. On the other hand, our work aims at predicting small-scale variability caused by system nondeterminism in the absence of interference.

III. Predicting Performance Variability

In this section, we describe our approach testing if the performance distribution of applications can be predicted. We start by describing the two use cases under which we predict performance distributions (Section III-A). We then describe different alternatives for formulating the prediction problem (Section III-B).

A. Variability Prediction Use Cases

There are multiple instances where one may want to predict the performance distribution of an application rather than measure it. It may be expensive to run the application a large number of times, or the system on which one would like to obtain that distribution may not be readily available. In these cases, predicting the performance distribution instead of measuring it can be an attractive alternative for a preliminary performance analysis. In this work, we look at two use cases under which we predict the performance distribution of an application on a system: predicting from a few runs on the same system (Section III-A1), and predicting from a measured distribution on a different system (Section III-A2). These two use-cases are illustrated in Figure 2.

1) Use Case #1: Predicting Distributions from a Few Runs: Figure 2(a) shows our first use case, which is to predict the performance distribution of an application on a system from just a few runs on that system. One of the challenges of measuring the performance distribution of an application is the need to run the application many times to construct a representative distribution. This process can be expensive, particularly for long-running applications and for applications with complex performance distributions that require a large number of runs to measure accurately. To overcome this challenge, we train a system-specific prediction model that takes a low-level profile of an application from a few runs and predicts the application's entire performance distribution. The representation of an application's profile is discussed in Section III-B1 and the representation of the predicted distribution is discussed in Section III-B2. The model is trained using a large amount of previously collected data consisting of the profiles and performance distributions of many benchmarks on the system of interest.

2) Use Case #2: Predicting Distributions for New Systems: Figure 2(b) shows our second use case, which is to predict the performance distribution of an application on a new system from a measured performance distribution on another system. For example, a user may be interested in acquiring a new system and may wonder what the performance distribution of their application would look like on this new system. If the user is unable to run their application on the new system, they may instead run the application on a system they already own and use the measured distribution on the old system to predict the performance distribution on the new system. To accomplish this objective, we train a system-to-system prediction model that takes the profile and performance distribution of an application on one system and predicts the application's performance distribution on another system. The representation of an application's profile is discussed in Section III-B1 and the representation of the input and predicted distribution is discussed in Section III-B2. The model is trained using a large amount of previously collected data consisting of the profiles and performance distributions of many benchmarks on both systems of interest. For example, the vendor of the new system may publish the performance distribution of a set of benchmarks and the user may run the same benchmarks on their old system to collect data for training the model.

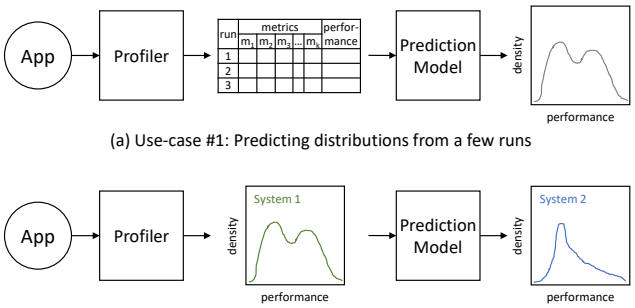

Fig. 2. Variability Prediction Use Cases

B. Variability Prediction Design Considerations

In this subsection, we look at different design considerations for predicting performance distributions. These considerations include how to represent an application's profile as an input to the prediction model (Section III-B1), how to represent an application's performance distribution as an input or output (Section III-B2), and what kind of model to use (Section III-B3).

1) Representing an Application's Profile: In both use cases, the prediction models take application profiles as input features. Since these models are trained with many different applications from different sources, the input features in the application profiles need to be represented in an application-independent way. For this reason, we abide by the following two principles when representing application profiles:

- We collect application-independent hardware and software metrics such as CPU metrics, cache metrics, TLB metrics, etc. These metrics can be collected from applications while treating the applications as black boxes. The specific list of metrics used in our evaluation is detailed in Section IV-D.
- Since the applications used for training the models have different running times, the absolute values of the metrics collected have different scales. We use relative metrics normalized per second to ensure that the metrics have the same scale across applications.

Hence, an application's profile is represented as a vector of application-independent hardware and software metrics normalized per unit time, which is fed as the input feature vector to the prediction models. In the case where multiple runs are used to construct the input feature vector, we include the mean, standard deviation, skewness, and kurtosis of each normalized metric in the input feature vector. We attempted to include higher-order moments as well, but their impact on prediction accuracy was insignificant.

2) Representing the Distribution: In both use cases, the prediction models produce a performance distribution as an output. In the second use-case, they also take a performance distribution as an input. Since these models are trained with many different applications with different absolute running

times, we predict a distribution of the relative time (i.e., normalized to mean time) so that the output has the same scale across applications. Predicting a distribution of relative time is sufficient because in our use cases, users are ultimately interested in the shape of the distribution. One important design consideration is how to represent the performance distribution in the input and output feature vector. We consider three different design approaches for representing the distribution for prediction purposes:

- **Histogram**: The first approach is for the feature vector to be the bins of a histogram of the relative time, similar to a discretized PDF.
- **PyMaxEnt**: The second approach is for the feature vector to be the moments of the distribution, which are then used to reconstruct the distribution using the principle of maximum entropy [30]. We consider the first four moments: mean, standard deviation, skewness, and kurtosis.
- **PearsonRnd**: The third approach is for the predicted output feature vector to be the first four moments of the distribution similar to the second approach. However, to reconstruct the distribution, we instead use the moments to draw random numbers from the distribution in the Pearson system with these moments (using *pearsrnd* in MATLAB [31]). These random numbers are subsequently used to reconstruct the distribution itself.

We compare these three approaches for representing the distribution in Section V.

3) Choice of Prediction Model: We consider three machine learning models for predicting performance distributions. We consider k-nearest neighbors (kNN) because of its ability to deal with noisy data, which is the case for the hardware and software metrics and the performance distribution data we collect. We set k to 15, and we use the cosine similarity as the distance metric as opposed to the Euclidean distance or other distance metrics which did not perform as well. We also consider random forests (RF) [32] and extreme gradient boosting (XGBoost) [33] because of their ability to deal with a large and diverse set of input features, which is the case for our application profiles. We compare the accuracy of these different models in Section V.

IV. Experimental Methodology

In this section, we detail the tools, benchmarks, systems, and profiling metrics used in our evaluation.

A. Libraries and Tools

The prediction workflows for training and testing the machine learning models are implemented in Python in addition to using Matlab [31] for the *pearsrnd* function. We use NumPy [34] and pandas [35] to prepare and process data, and Python for plotting and leave-one-group-out cross-validation from Scikit-Learn [36], in addition to SciPy [37] to evaluate the performance of our models.

TABLE I
Benchmarks used in the evaluation

Suite	Benchmarks
NPB [38]	bt, cg, ep, ft, is, lu, mg, sp, ua
PARSEC3.0 [39]	blackscholes, bodytrack, canneal, dedup, fluidanimate, freqmine, netdedup, streamcluster, swaptions
SPEC OMP [2]	358, 362, 367, 372, 376
SPEC Accel [40]	303, 304, 353, 354, 355, 356, 359, 363
Parboil [41]	bfs, cutcp, histo, lbm, mrigridding, sgemm, spmv, stencil
Rodinia [42]	backprop, bfs, heartwall, hotspot, kmeans, lavaMD, leukocyte, ludomp, particle_filter, pathfinder
MLlib [43]	correlation, dtclassifier, fmclassifier, gbtclassifier, kmeans, logisticregression, lsvc, mlp, pca, randomforestclassifier, summarizer

B. Benchmarks

The benchmarks used in the evaluation are listed in Table I. These benchmarks come from seven different suites and cover a variety of application domains.

C. Hardware Specifications

We use two different systems in our evaluation that are distinguished by their CPUs. Our *Intel* system uses an Intel Xeon Platinum 8358 CPU, and our *AMD* system uses an AMD EPYC 7543 CPU. Both systems have 512GB of DDR4 RAM, two sockets, and 32 cores per socket. The benchmarks ran on an entire node and without any interference. We use the Intel system for evaluating the first use case and both systems for evaluating the second use case.

D. Performance Metrics

We use the Linux `perf` tool [44] to profile applications and collect the metrics used as input features to our models. The metrics were selected from distinct categories, including metrics related to the operating system, hardware, and software, ensuring a comprehensive assessment of system performance. We collect a total of 68 profiling metrics from System 1, listed in Table II, and 75 profiling metrics from System 2, listed in Table III.

E. Visualization and Analysis of Performance Distributions

Throughout the evaluation, we represent performance as distributions measured from 1,000 repeated executions. We visualize the distributions as kernel density estimates (KDE), shown as smooth curves, offering a continuous representation of the data distribution. This visualization allows for a detailed examination of the characteristics of each benchmark, including central tendency, spread, and skew.

When evaluating the accuracy of a predicted distribution, we use the Kolmogorov-Smirnov (KS) [45] divergence test to assess the goodness of fit and quantify the agreement between the observed and predicted distributions. A value of 0 for the KS test indicates a perfect match, and as the value increases up to a maximum of 1, the match decreases.

TABLE II
PROFILING METRICS COLLECTED FOR THE INTEL CPU SYSTEM

ID	Metric
0	branch-instructions
1	branch-misses
2	bus-cycles
3	cache-misses
4	cache-references
5	cpu-cycles
6	instructions
7	ref-cycles
8	alignment-faults
9	bpf-output
10	cgroup-switches
11	context-switches
12	cpu-clock
13	cpu-migrations
14	emulation-faults
15	major-faults
16	minor-faults
17	page-faults
18	task-clock
19	duration_time
20	L1-dcache-load-misses
21	L1-dcache-loads
22	L1-dcache-stores
23	l1d.replacement
24	L1-icache-load-misses
25	l2_lines_in.all
26	l2_rqsts.all_demand_miss
27	l2_rqsts.all_rfo
28	l2_trans.l2_wb
29	LLC-load-misses
30	LLC-loads
31	LLC-store-misses
32	LLC-stores
33	longest_lat_cache.miss
34	mem_inst_retired.all_loads
35	mem_inst_retired.all_stores
36	mem_inst_retired.lock_loads
37	branch-load-misses
38	branch-loads
39	dTLB-load-misses
40	dTLB-loads
41	dTLB-store-misses
42	dTLB-stores
43	iTLB-load-misses
44	node-load-misses
45	node-loads
46	node-store-misses
47	node-stores
48	mem-loads
49	mem-stores
50	slots
51	assists.fp
52	cycle_activity.stalls_l3_miss
53	assists.any
54	topdown.backend_bound_slots
55	br_inst_retired.all_branches
56	br_misp_retired.all_branches
57	cpu_clk_unhalted.distributed
58	cycle_activity.stalls_total
59	inst_retired.any
60	lsd.uops
61	resource_stalls.sb
62	resource_stalls.scoreboard
63	dtlb_load_misses.stlb_hit
64	dtlb_store_misses.stlb_hit
65	itlb_misses.stlb_hit
66	unc_cha_tor_inserts.io_hit
67	unc_cha_tor_inserts.io_miss

TABLE III
PROFILING METRICS COLLECTED FOR AMD CPU SYSTEM

ID	Metric
0	branch-instructions
1	branch-misses
2	cache-misses
3	cache-references
4	cpu-cycles
5	instructions
6	stalled-cycles-backend
7	stalled-cycles-frontend
8	alignment-faults
9	bpf-output
10	cgroup-switches
11	context-switches
12	cpu-clock
13	cpu-migrations
14	emulation-faults
15	major-faults
16	minor-faults
17	page-faults
18	task-clock
19	duration_time
20	L1-dcache-load-misses
21	L1-dcache-loads
22	L1-dcache-prefetches
23	L1-icache-load-misses
24	L1-icache-loads
25	branch-load-misses
26	branch-loads
27	dTLB-load-misses
28	dTLB-loads
29	iTLB-load-misses
30	iTLB-loads
31	branch-instructions
32	branch-misses
33	cache-misses
34	cache-references
35	cpu-cycles
36	stalled-cycles-backend
37	stalled-cycles-frontend
38	bp_l2_btb_correct
39	bp_tlb_rel
40	bp_l1_tlb_miss_l2_tlb_hit
41	bp_l1_tlb_miss_l2_tlb_miss
42	ic_fetch_stall.ic_stall_any
43	ic_tag_hit_miss.instruction_cache_hit
44	ic_tag_hit_miss.instruction_cache_miss
45	op_cache_hit_miss.all_op_cache_accesses
46	fp_ret_sse_avx_ops.all
47	fpu_pipe_assignment.total
48	l1_data_cache_fills_all
49	l1_data_cache_fills_from_external_ccx_cache
50	l1_data_cache_fills_from_memory
51	l1_data_cache_fills_from_remote_node
52	l1_data_cache_fills_from_within_same_ccx
53	l1_dtlb_misses
54	l2_cache_accesses_from_dc_misses
55	l2_cache_accesses_from_ic_misses
56	l2_cache_hits_from_dc_misses
57	l2_cache_hits_from_ic_misses
58	l2_cache_hits_from_l2_hwpf
59	l2_cache_misses_from_dc_misses
60	l2_cache_misses_from_ic_miss
61	l2_dtlb_misses
62	l2_itlb_misses
63	macro_ops_retired
64	sse_avx_stalls
65	l3_cache_accesses
66	l3_misses
67	ls_sw_pf_dc_fills.mem_io_local
68	ls_sw_pf_dc_fills.mem_io_remote
69	ls_hw_pf_dc_fills.mem_io_local
70	ls_hw_pf_dc_fills.mem_io_remote
71	ls_int_taken
72	all_tlbs_flushed
73	instructions
74	bp_l1_btb_correct

V. EVALUATION

In this section, we motivate the need to predict variability by showing its occurrence across our benchmarks (Section V-A). We then evaluate how accurately variability can be predicted for each of our two use cases: predicting from a few runs on the same system (Section V-B), and predicting from a measured distribution on a different system (Section V-C). For each use case, we evaluate the accuracy of prediction for different representations of the predicted distribution and different choice of models. We also show examples of the predicted distributions to get more intuition about their accuracy.

A. Performance Variability in Benchmarks

Figure 3 shows the distribution of the relative execution time of all the benchmarks on the Intel system. The diversity in the shapes of the performance distributions demonstrates how variable performance can be, both within and across applications, even on the same hardware. This observation reiterates that performance cannot be accurately represented by a single-point summary because that could fail to capture important features of the distribution such has how wide or narrow it is, its modes, and if it has a tail. It emphasizes the importance of treating performance as a distribution, and the need for predicting performance distributions for preliminary performance analysis.

B. Evaluating Use Case #1: Predicting Distributions from a Few Runs

Figure 4 shows how the KS score of the predicted distributions varies with the representation of the predicted distribution and the choice of model for the first use case when ten runs are used. For each combination of representation and model, the violin plot shows how the KS score is distributed across the benchmarks.

1) Distribution representation: We observe from Figure 4 that the representation of the distribution that results in the best mean KS score is the PearsonRnd representation. The mean KS score of the PearsonRnd representation for the best choice of model is 0.241, in contrast with 0.278 and 0.302 for the Historgram and PyMaxEnt representations, respectively.

2) Choice of model: We observe from Figure 4 that the choice of model that results in the best mean KS score is the kNN model. The mean KS score of the kNN model for the best choice of distribution representation is 0.241, in contrast with 0.247 and 0.248 for the XGBoost and Random Forest models, respectively, noting that the improvement is more prominent with a lower number of samples.

3) Number of samples: Figure 6 shows how the KS score of the predicted distributions varies with the number of runs (or samples) made from the application that we are making predictions for. For each number of samples, the violin plot shows how the KS score is distributed across the benchmarks. We observe a significant improvement in the KS score when

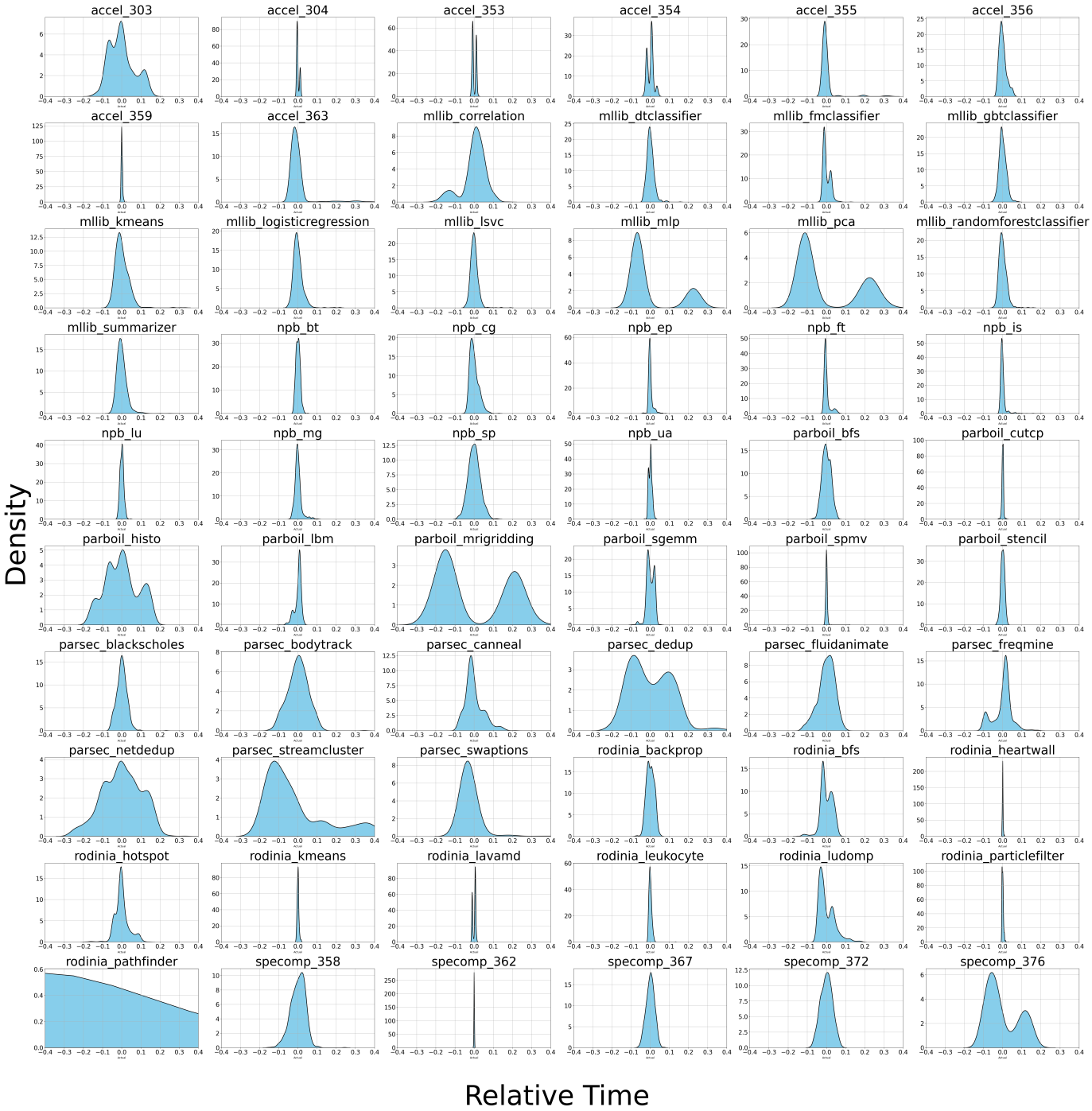

Fig. 3. Relative execution time density plots for all benchmarks on the Intel system

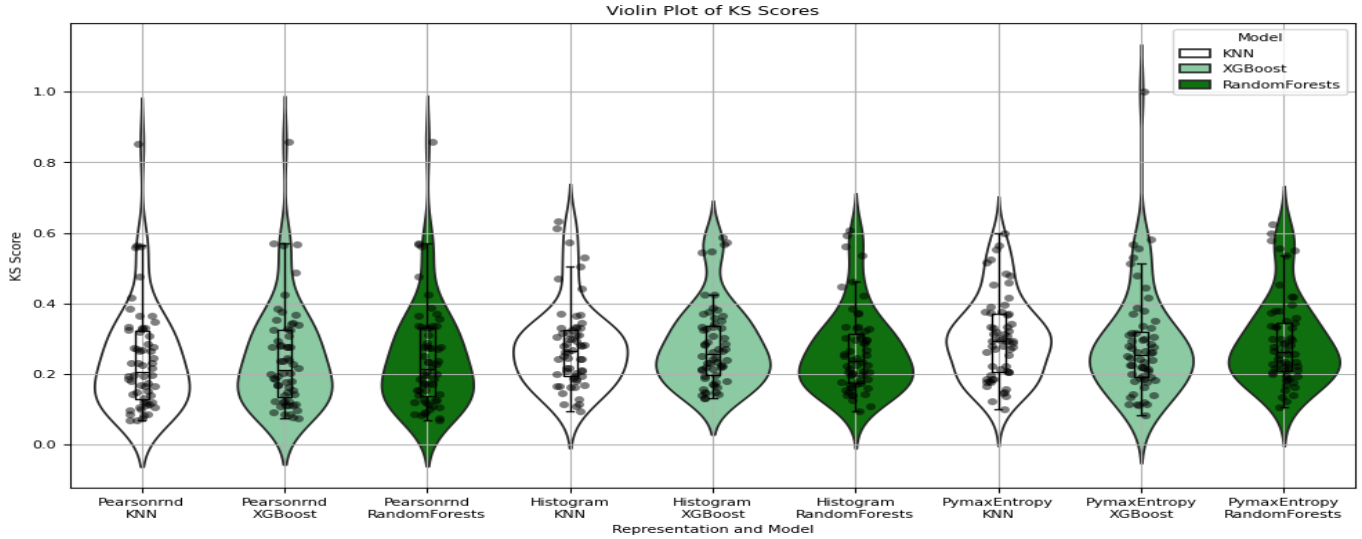

Fig. 4. KS scores of the predicted distributions for different distribution representations and choice of model (Intel System)

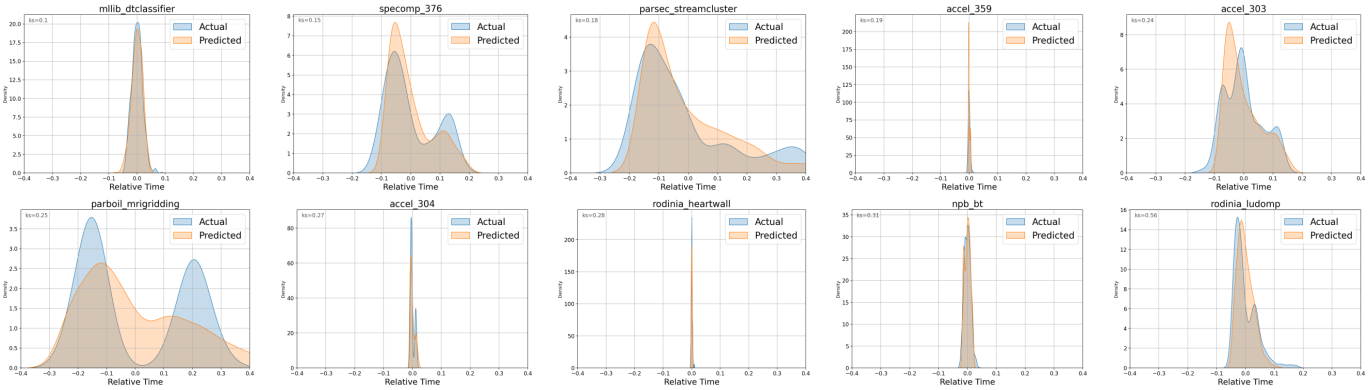

Fig. 5. Overlay of the predicted and actual distributions for selected benchmarks across the KS score spectrum when predicting with ten samples

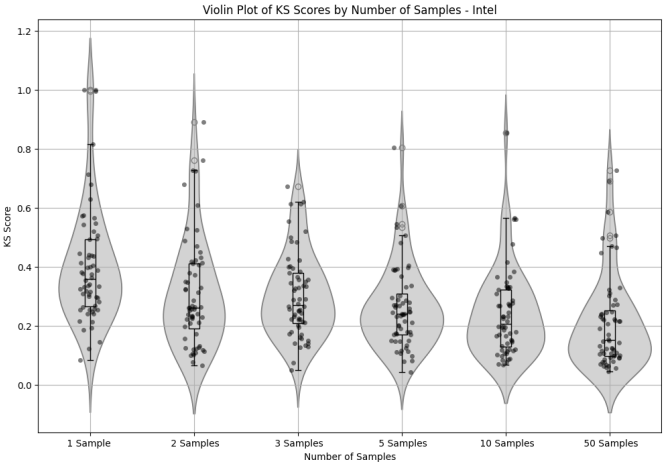

Fig. 6. KS score of the predicted distribution with different numbers of samples (Intel System)

going from one sample to multiple samples, since using multiple samples reduces the chances of the single sample being not representative. When using multiple samples, we observe a steady improvement in the KS score as the number of samples increases. This result demonstrates that users can trade off sampling time for prediction accuracy.

4) Examples: As shown in Figures 4 and 6, the KS score varies significantly across benchmarks for a fixed representation, model, and number of runs. To get more intuition about the accuracy of the predicted distributions, Figure 5 shows an overlay of the predicted and actual distributions for selected benchmarks across the KS score spectrum when using the PearsonRnd representation, the kNN model, and predicting using just ten runs on Intel system. We observe that the overall width of the distribution can be correctly predicted, including very narrow distributions (e.g., 359, 304, bt, heartwall), distributions with moderate width (e.g., dt-classifier, ludomp), and wide distributions (e.g., 303, 376, mrigridding), not to mention skewed distributions with long tails (e.g., streamcluster). The existence of multiple modes is

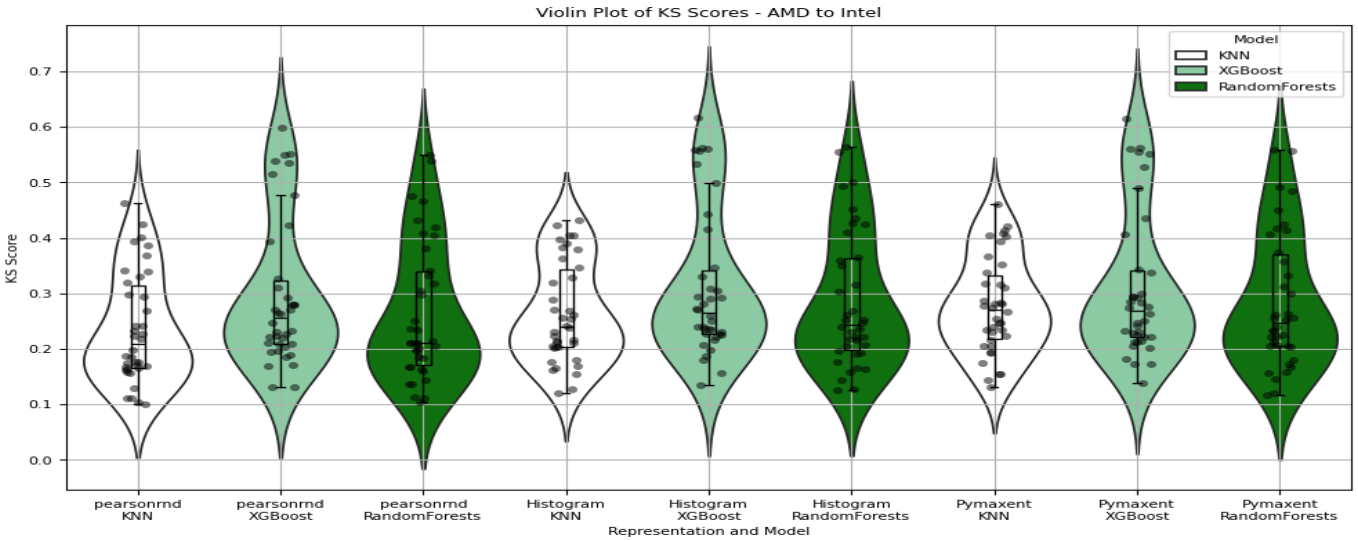

Fig. 7. KS score of the predicted distribution when collecting data on the AMD system and making predictions for the Intel system

also predicted with reasonable success (e.g., 303, 304, 376, bt, mrigridding), particularly the relative positions of the modes and their relative sizes. These results show that although a performance distribution is not predicted very precisely, a sufficient amount of correct information about the shape of the distribution can be predicted from just a few samples.

C. Evaluating Use Case #2: Predicting Distributions for New Systems

Figure 7 shows how the KS score of the predicted distributions varies with the representation of the predicted distribution and the choice of model for the second use case when collecting data on the AMD system and making predictions for the Intel system. For each combination of representation and model, the violin plot shows how the KS score is distributed across the benchmarks.

1) Distribution representation: We observe from Figure 7 that, similar to the first use case, the representation of the distribution that results in the best mean KS score is the PearsonRnd representation. The mean KS score of the PearsonRnd representation for the best choice of model is 0.236, in contrast with 0.264 and 0.277 for the Histogram and PyMaxEnt representations, respectively.

2) Choice of model: We observe from Figure 7 that, similar to the first use case, the choice of model that results in the best mean KS score is the kNN model. The mean KS score of the kNN model for the best choice of distribution representation is 0.236, in contrast with 0.291 and 0.263 for the XGBoost and Random Forest models, respectively.

3) Direction of prediction: Figure 8 shows the KS score of the predicted distribution for different benchmarks when measuring on the AMD system and predicting on the Intel system, and vice versa. For each case, the violin plot shows how the KS score is distributed across the benchmarks. It is interesting that predicting from the AMD system to the Intel system seems easier overall than the other way around, but only slightly.

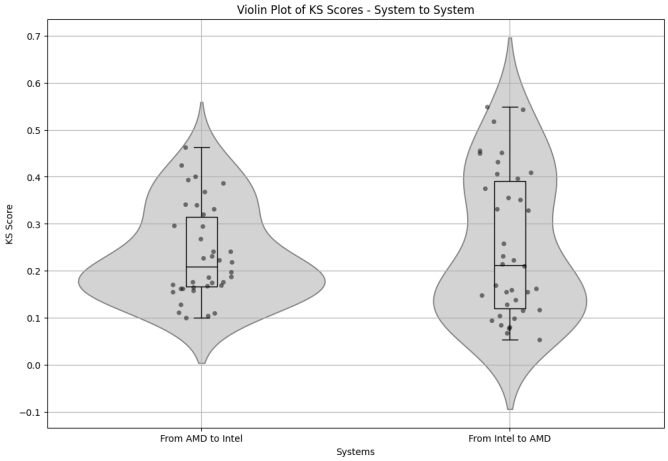

Fig. 8. KS score of the predicted distribution for different system-to-system predictions

4) Examples: As shown in Figures 7 and 8, the KS score varies significantly across benchmarks for a fixed representation, model, and direction. To get more intuition about the accuracy of the predicted distributions, Figure 9 shows an overlay of the predicted and actual distributions for selected benchmarks across the KS score spectrum when using the PearsonRnd representation, the kNN model, and predicting from the AMD system to the Intel system. We observe that predicting the overall width of the distribution is done fairly well, including very narrow distributions (e.g., is, heartwall, spmv), distributions with moderate width (e.g., bfs, gbtclassifier, sgemm), and wide distributions (e.g., bodytrack, canneal, correlation, histo). The existence of multiple modes is also predicted fairly well (e.g., bfs, canneal, correlation, histo,

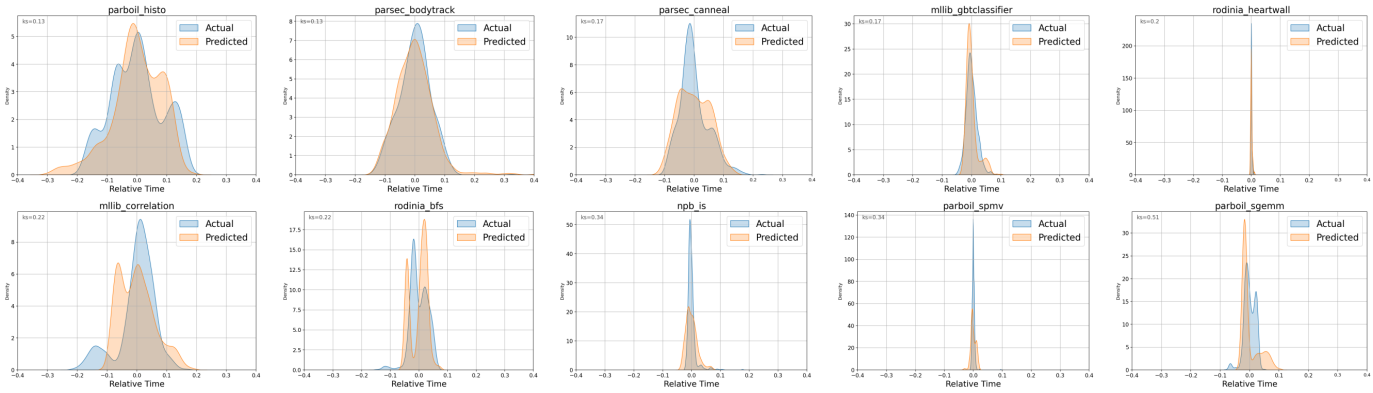

Fig. 9. Overlay of the predicted and actual distributions for selected benchmarks across the KS score spectrum when predicting from the AMD system to the Intel system

sgemm), but with mixed success in correctly predicting the relative positions of the modes (e.g., bfs, canneal, histo, sgemm) and their relative sizes (e.g., histo, sgemm). These results show that although a precise performance distribution is difficult to predict, a sufficient amount of correct information about the shape of the distribution can be predicted to be useful in a preliminary performance analysis for a new system.

VI. Conclusion and Future Work

The aim of this work is to find out how successfully the performance distribution of an application on a system can be predicted by learning from other representative applications. We considered two use cases for our predictors: predicting the performance distribution of an application on a system from a few runs of the application on that system, and predicting the performance distribution of an application on a system from a measured distribution on a different system. We considered different techniques for formulating the prediction problem as well as different types of prediction models. Our evaluation shows that, for both cases, the best prediction results can be achieved when using the k-nearest neighbors approach to predict the first four moments of the distribution, then reconstructing the distribution by sampling the Pearson distribution with the predicted moments.

There are a couple of factors that could limit the generality of our evaluation and can be improved in our future work. The first factor is that a prediction model's accuracy is generally improved with more training data. Although the number of benchmarks we have used in our evaluation is not small, it is still modest and mostly focused on scientific computing and data analytics applications. We expect that increasing the number and diversity of benchmarks that we train on could further improve the accuracy of the predicted distributions. The second limitation is that our evaluation only uses two CPU systems. We expect that our results can be further generalized by being demonstrated on a larger number of systems, including systems with accelerators such as GPUs and benchmarks that use those accelerators.

References

[1] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci, "Taming performance variability," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, Oct. 2018, pp. 409–425. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/maricq

[2] M. S. Müller, J. Baron, W. C. Brantley, H. Feng, D. Hackenberg, R. Henschel, G. Jost, D. Molka, C. Parrott, J. Robichaux *et al.*, "SPEC OMP2012 — an application benchmark suite for parallel systems using OpenMP," in *International Workshop on OpenMP*. Springer, 2012, pp. 223–236.

[3] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results," in *Proceedings of the international conference for high performance computing, networking, storage and analysis (SC)*. IEEE/ACM, Nov. 2015, pp. 1–12.

[4] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "The popper convention: Making reproducible systems evaluation practical," in *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, Jul. 2017, pp. 1561–1570.

[5] P. Bruel, V. Mittal, D. M. M. Faloutsos, and E. Frachtenberg, "Revisiting performance evaluation in an age of uncertainty," in *Proceedings of the 5th Workshop on Education for High Performance Computing (EduHiPC'23)*. Goa, India: IEEE, Dec. 2023.

[6] S. Hunold and A. Carpen-Amarie, "Reproducible mpi benchmarking is still not as easy as you think," *Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3617–3630, Mar. 2016.

[7] V. Mittal, P. Bruel, D. Milojicic, and E. Frachtenberg, "Adaptive stopping rule for performance measurements," in *14th IEEE International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Denver, CO: IEEE, Nov. 2023.

[8] G. Kiczales, "Towards a new model of abstraction in software engineering," in *Proceedings of the International Workshop on Object Orientation in Operating Systems*. IEEE, Oct. 1991, pp. 127–128.

[9] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," *SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, Apr. 2014.

[10] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suite," in *Proceedings of the 3rd workshop on general-purpose computation on graphics processing units (GPGPU*. ACM, Mar. 2010, pp. 63–74.

[11] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," *SIGPLAN Notices*, vol. 48, no. 4, pp. 77–88, Apr. 2013.

[12] A. Nassereldine, S. Diab, M. Baydoun, K. Leach, M. Alt, D. Milojicic, and I. El Hajj, "Predicting the performance-cost trade-off of applications across multiple systems," in *IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, May 2023, pp. 216–228.

[13] L. Bulej, V. Horkỳ, P. Tuma, F. Farquet, and A. Prokopec, "Duet benchmarking: Improving measurement accuracy in the cloud," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, Apr. 2020, pp. 100–107.

[14] S. Eismann, C.-P. Bezemer, W. Shang, D. Okanović, and A. van Hoorn, "Microservices: A performance tester's dream or nightmare?" in *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, Apr. 2020, pp. 138–149.

[15] J. Scheuner, "Performance evaluation of serverless applications and infrastructures," Ph.D. dissertation, 2022.

[16] C. Collberg and T. A. Proebsting, "Repeatability in computer systems research," *Communications of the ACM*, vol. 59, no. 3, pp. 62–69, 2016.

[17] J. Vitek and T. Kalibera, "Repeatability, reproducibility, and rigor in systems research," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, ser. EMSOFT'11. New York, NY, USA: Association for Computing Machinery, 2011, p. 33–38. [Online]. Available: https://doi.org/10.1145/2038642.2038650

[18] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-Eldin, C. L. Abad, J. N. Amaral, P. Tůma, and A. Iosup, "Methodological principles for reproducible performance evaluation in cloud computing," *Transactions on Software Engineering*, vol. 47, no. 8, pp. 1528–1543, Jul. 2019.

[19] D. Beyer, S. Löwe, and P. Wendler, "Reliable benchmarking: requirements and solutions," *International Journal on Software Tools for Technology Transfer*, vol. 21, pp. 1–29, Feb. 2019.

[20] A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J. Rellermeyer, C. Maltzahn, R. Ricci, and A. Iosup, "Is big data performance reproducible in modern cloud networks?" in *symposium on networked systems design and implementation (NSDI)*. ACM, Feb. 2020, pp. 513–527. [Online]. Available: https://www.usenix.org/system/files/nsdi20-paper-uta.pdf

[21] A. B. De Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Why you should care about quantile regression," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 207–218, 2013. [Online]. Available: https://uwaterloo.ca/embedded-software-group/sites/ca.embedded-software-group/files/uploads/files/asplos13-quantreg.pdf

[22] M. Hocko and T. Kalibera, "Reducing performance non-determinism via cache-aware page allocation strategies," in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*. ACM, 1 2010, pp. 223–234.

[23] T. Patki, J. J. Thiagarajan, A. Ayala, and T. Z. Islam, "Performance optimality or reproducibility: That is the question," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. ACM/IEEE, Nov. 2019, pp. 1–30.

[24] D. Nichols, A. Marathe, K. Shoga, T. Gamblin, and A. Bhatele, "Resource utilization aware job scheduling to mitigate performance variability," in *International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, Jun. 2022, pp. 335–345.

[25] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Diagnosing performance variations in hpc applications using machine learning," in *High Performance Computing: 32nd International Conference, ISC High Performance*. Springer, Jun. 2017, pp. 355–373. [Online]. Available: https://www.bu.edu/peaclab/files/2020/01/isc.pdf

[26] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting cloud performance for hpc applications: A user-oriented approach," in *17th International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, May 2017, pp. 524–533.

[27] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, Mar. 2017, pp. 469–482. [Online]. Available: https://www.usenix.org/system/files/conference/nsdi17/nsdi17-alipourfard.pdf

[28] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best vm across multiple public clouds: A data-driven performance modeling approach," in *Proceedings of the Symposium on Cloud Computing (SOCC)*. ACM, Sep. 2017, pp. 452–465.

[29] Y. Zhao, D. Duplyakin, R. Ricci, and A. Uta, "Cloud performance variability prediction," in *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, Apr. 2021, pp. 35–40.

[30] T. Saad and G. Ruai, "Pymaxent: A python software for maximum entropy moment reconstruction," *SoftwareX*, vol. 10, p. 100353, 2019.

[31] T. M. Inc., "Matlab (r2022b)," Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com

[32] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[33] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[34] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[35] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and Others, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[37] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[38] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The NAS parallel benchmarks summary and preliminary results," in *Supercomputing'91: Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*. IEEE, 1991, pp. 158–165.

[39] X. Zhan, Y. Bao, C. Bienia, and K. Li, "PARSEC3.0: A multicore benchmark suite with network stacks and SPLASH-2X," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 5, pp. 1–16, 2017.

[40] G. Juckeland, W. Brantley, S. Chandrasekaran, B. Chapman, S. Che, M. Colgrove, H. Feng, A. Grund, R. Henschel, W.-M. W. Hwu *et al.*, "SPEC ACCEL: A standard application suite for measuring hardware accelerator performance," in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 2014, pp. 46–67.

[41] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, p. 27, 2012.

[42] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2009, pp. 44–54.

[43] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "MLlib: Machine learning in Apache Spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[44] A. C. De Melo, "The new linux'perf'tools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[45] T. B. Arnold and J. W. Emerson, "The r journal: Nonparametric goodness-of-fit tests for discrete null distributions," *The R Journal*, vol. 3, pp. 34–39, 2011, https://doi.org/10.32614/RJ-2011-016.