On the Feasibility and Benefits of Extensive Evaluation

YUJIE HUI*, The Ohio State University, USA
MIAO YU*, The Ohio State University, USA
HAO QI, University of California, Merced, USA
YIFAN GAN, The Ohio State University, USA
TIANXI LI, The Ohio State University, USA
YUKE LI, University of California, Merced, USA
XUEYUAN REN, The Ohio State University, USA
SIXIANG MA, The Ohio State University, USA
XIAOYI LU, University of California, Merced, USA
YANG WANG, The Ohio State University, USA

Benchmark and system parameters often have a significant impact on performance evaluation, which raises a long-lasting question about which settings we should use.

This paper studies the feasibility and benefits of extensive evaluation. A full extensive evaluation, which tests all possible settings, is usually too expensive. This work investigates whether it is possible to sample a subset of the settings and, upon them, generate observations that match those from a full extensive evaluation. Towards this goal, we have explored the incremental sampling approach, which starts by measuring a small subset of random settings, builds a prediction model on these samples using the popular ANOVA approach, adds more samples if the model is not accurate enough, and terminates otherwise.

To summarize our findings: 1) Enhancing a research prototype to support extensive evaluation mostly involves changing hard-coded configurations, which does not take much effort. 2) Some systems are highly predictable, which means that they can achieve accurate predictions with a low sampling rate, but some systems are less predictable. 3) We have not found a method that can consistently outperform random sampling + ANOVA. Based on these findings, we provide recommendations to improve artifact predictability and strategies for selecting parameter values during evaluation.

CCS Concepts: • **Information systems → Database performance evaluation**.

Additional Key Words and Phrases: Database Performance Evaluation, Benchmarking

*Yujie Hui and Miao Yu contributed equally to this paper.

Authors' Contact Information: Yujie Hui, The Ohio State University, Columbus, Ohio, USA, hui.82@osu.edu; Miao Yu, The Ohio State University, USA, Columbus, Ohio, yu.3053@osu.edu; Hao Qi, University of California, Merced, USA, Merced, California, hqi6@ucmerced.edu; Yifan Gan, The Ohio State University, USA, Columbus, Ohio, gan.101@osu.edu; Tianxi Li, The Ohio State University, USA, Columbus, Ohio, li.9443@osu.edu; Yuke Li, University of California, Merced, USA, Merced, California, yli304@ucmerced.edu; Xueyuan Ren, The Ohio State University, USA, Columbus, Ohio, ren.450@osu.edu; Sixiang Ma, The Ohio State University, USA, Columbus, Ohio, ma.1189@osu.edu; Xiaoyi Lu, University of California, Merced, USA, Merced, California, xiaoyi.lu@ucmerced.edu; Yang Wang, The Ohio State University, USA, Columbus, Ohio, wang.7564@osu.edu.

ACM 2836-6573/2024/9-ART201
https://doi.org/10.1145/3677137

ACM Reference Format:
Yujie Hui, Miao Yu, Hao Qi, Yifan Gan, Tianxi Li, Yuke Li, Xueyuan Ren, Sixiang Ma, Xiaoyi Lu, and Yang Wang. 2024. On the Feasibility and Benefits of Extensive Evaluation. *Proc. ACM Manag. Data* 2, N4 (SIGMOD), Article 201 (September 2024), 24 pages. https://doi.org/10.1145/3677137

1 Introduction

Performance evaluation and comparison are routine tasks in both academia and industry. However, as shown in multiple works [34, 45, 46, 75, 78], evaluation results can be highly sensitive to the settings (i.e., hardware and software configurations) of the experiments — it often happens that one system can outperform another under one setting but fails to do so under a different setting. Even worse, the choice of experimental settings may introduce bias into the results [35].

There are at least two philosophies to address this problem, each with its limitations. First, we can restrict the possible settings, probably based on studies of realistic settings in production [10, 14, 55, 79]. This philosophy has been adopted by many industrial benchmarks [25, 58, 66, 72], but we observe that, on the one hand, too many restrictions will limit its applicability and drive researchers away. For example, the standard TPC-C benchmark [71] requires a long "think time" before each transaction, but since think time limits the contention level of TPC-C, most research prototypes simply remove it [75]. On the other hand, less restriction will give researchers enough flexibility, but this again raises the question of which settings to use. For example, YCSB [20] includes five types of workloads, with tunable key-value sizes and skewness, but it is not uncommon for a work to only test a subset of the possible settings.

Second, we can conduct an extensive evaluation to test all the possible combinations of parameter values. Extensive evaluation could provide a comprehensive picture of system performance, but is often too expensive. To reduce the cost, the standard practice is to start from a popular setting and then tune one parameter at a time. However, due to the intricate effects of multiple parameters [11, 12, 68, 75], it is unclear whether tuning one parameter at a time will miss any important observations.

While it probably requires a community debate to discuss the appropriate philosophy, this work explores whether we can improve the state of the art in extensive evaluation. In particular, the key question we want to answer is, assuming we only have resources to test a limited number of settings, how to choose the settings to test and whether testing these settings can generate observations that match those from the full extensive evaluation. In this paper, we report our attempt, share our experiences, and discuss possible future directions.

We conducted our study on 10 research prototypes and one mature open-source software with the following methodology: We first performed a comprehensive, best-effort evaluation on these systems, which is costly, but provides us with a "ground truth". Subsequently, we investigated whether we can measure a subset of the experiment settings and still generate observations that match those from the "ground truth". To be specific, we used a classic solution in computer systems performance analysis [39], which starts by measuring the performance of a small subset of random settings, builds a prediction model on these samples using the ANOVA method [64], adds more samples if the model is not accurate enough, and terminates otherwise. We further tried other sampling and analysis methods. In this paper, we refer to these approaches "incremental sampling".

We make the following observations from our study:

- **Required effort**: Since a research prototype is usually not as stable as a mature product, there may be concerns about whether a prototype can even run properly under different settings, especially under a new setting that has not been used in the corresponding publication. Our overall experience is quite positive: Although many prototypes initially indeed cannot run properly under a new setting, we find that most of the problems are due to hard-coded configuration values, which can be fixed with a small amount of effort. There exist new settings that a prototype

is more likely to encounter errors, but failing a few settings usually does not prevent us from getting an overall picture.

- **Cost**: The effectiveness of incremental sampling depends on the target system: For some systems, incremental sampling can terminate with a low sampling rate (as low as 10% in our experiments) while still providing an analysis result that can match that from the ground truth. We call these systems highly predictable in our paper. Some other systems are less predictable and thus incremental sampling needs a high sampling rate to terminate. As one can imagine, incremental sampling can significantly reduce the evaluation cost of highly predictable systems, but is less effective for less predictable systems. By comparing different implementations, we have verified that it is possible to improve the implementation of some of the less predictable systems to provide more predictable performance.
- **Benefits**: Extensive evaluation, even with incremental sampling, can reveal design issues that cause performance degradation in previously untested settings. For example, our evaluation of MySQL and DrTM [77] shows that they experience performance degradation when loaded with a large amount of data, since their tree indexes grow larger. This problem could be addressed by partitioning data [16, 21, 41, 43]. Such issues have confirmed the necessity of extensive evaluation.
- **Future**: Despite our efforts, we have not found a sampling and/or prediction method that can consistently outperform random sampling + ANOVA. However, we find that these highly predictable systems show consistent predictability under different sampling and modeling methods. This suggests that, in addition to reproducibility, we should further encourage the predictability of system performance.

2 Methodologies

We carry out our research in three steps.

Step 1. Reproduction. In the first step, we selected 10 research prototypes and tried to reproduce their results reported in the corresponding publications (i.e., Calvin [2, 60, 70], Silo [65, 73], DrTM [26, 77], Janus [40, 53], Cicada [17, 45], GAM [15, 30], Star [47, 67], Aria [9, 46], HERD [36, 41], and TAPIR [7, 81]). These publications use TPC-C [71] and/or YCSB [20] benchmarks for evaluation. We run each experiment three times, and use the same setup (e.g., warm up length, total length, etc.) as mentioned in the original papers. This step confirms that we have properly set up and run these prototypes.

For a very brief introduction, Silo and Cicada are single-node in-memory transaction processing engines; Calvin, DrTM, GAM, Star, and Aria support distributed transactions; TAPIR and Janus further support geo-distributed transactions; HERD is a distributed key-value store. DrTM, GAM, and HERD use RDMA techniques to improve performance.

To evaluate the impact of configuration parameters, the state of the art is to tune one parameter at a time while using fixed values for other parameters. However, from the publications we have reproduced, we observe that there is no consensus about which parameter to tune, the value range to tune the parameter, and the values of the remaining fixed parameters. To give a few examples,

- Number of warehouses in TPC-C: Calvin uses fixed values of 10 and 20 warehouses per server and tunes the number of servers from 10 to 100. Silo tunes it from 1 to 32. DrTM tunes it from 1 to 16. Janus uses a fixed value of 6. Cicada tunes it from 1 to 28. GAM uses a fixed value of 32. Star uses a fixed value of 12 per server and tunes the number of servers from 2 to 16. Aria tunes it from 1 to 180.
- Number of key-value pairs in YCSB: Silo uses a fixed value of 160M. Cicada uses a fixed value of 10M. Star uses a fixed value of 2.4M. Aria uses a fixed value of 480K. HERD uses a fixed value of 80M. Tapir uses a fixed value of 1M.

- Skewness in YCSB: Silo, Star, and Tapir use a uniform distribution. Cicada tunes Zipf factor from 0 to 0.99. Aria tunes Zipf factor from 0 to 0.999. HERD uses fixed Zipf values of 0 and 0.99.

On the one hand, such arbitrary selection of parameter values makes it hard to compare different works, and creates questions like whether a chosen setting may be in favor of one work over another. On the other hand, it is probably unrealistic to ask every publication to follow exactly the same configurations, since different works may target different scenarios. This dilemma serves as the motivation for our work.

We ran our experiments on CloudLab [18] and a local cluster. For each work, we try to use hardware that is close to the one used by the original paper. On CloudLab, we use m510 machines to run Aria experiments, c6320 machines to run Silo and Cicada experiments, rs630 to run Janus experiments, c220g1 machines to run TAPIR experiments, d430 machines to run Calvin experiments, c6220 machines to run Star and GAM experiments, and r320 machines to run HERD experiments. We test DrTM on a local cluster, in which each node is equipped with a 28-core Intel Xeon E5-2680 CPU and 128 GB of RAM. The cluster is interconnected by a 100 Gbps Mellanox CX-4 adapter. We use OpenFabrics Enterprise Distribution (OFED) version 4.9. Detailed hardware configuration information is recorded in Table 2 in the appendix.

Step 2. Best-effort extensive evaluation. In the second step, we try to carry out a best-effort extensive evaluation, which includes settings that the original papers do not cover. We measured their throughput in this work and leave other metrics, like latency, for the future. The result of this step forms a "ground truth" for our next step. In this step, we added MySQL, a mature open-source database, for comparison. Specifically, we tuned the following parameters:

- TPC-C parameters. We 1) tune the number of warehouses from 1 to the maximum allowed by DRAM (all experiments assume that all data can fit into DRAM); 2) tune the number of workers from 1 to up to 32; 3) tune the cross-warehouse rate from 0% to 100%; 4) tune the workload to include all five types of transactions, two types of transactions (NewOrder and Payment), and one type of transactions (NewOrder).
- YCSB parameters. We 1) tune the key-value size from 8B to 1KB; 2) tune the number of key-value pairs from 128K to the maximal allowed by DRAM; 3) tune read/write ratio from 100:0 to 0:100; and 4) tune the α value of the Zipfian distribution from 0 to 0.99 ($\alpha = 0$ is equivalent to uniform distribution).
- System parameters. For distributed systems, we tune the number of servers from 1 to 32. For systems that support replication, we tune the number of replicas from 1 to 7. We further tune the performance-critical parameters of some systems, such as the memory buffer and page size in MySQL and the number of locks in Aria.

Table 1 summarizes the settings we finished running (some systems do not support the tuning of a certain parameter) and the total machine hours it took to run these experiments. Such experiences have confirmed the high cost of extensive evaluation: Despite the fact that we only target a few parameters and are selective about parameter values, some of these systems require thousands of machine hours to test. Such cost may further grow significantly considering the possibility of having more parameters to tune and the need to run the experiments for multiple iterations due to debugging and optimization. As a result, we don't have the resources to test all the possible values in our study and thus have to pre-select values as shown in Table 1.

During the procedure, we found that many prototypes are not fully ready for extensive evaluation, mostly due to hard-coded configuration parameters. To be concrete, Silo, Cicada, DrTM, and GAM allocate a fixed amount of memory and we made it configurable; DrTM assumes the number of worker threads is equal to the number of warehouses and we made the ratio configurable; Janus could cause "too many open files" when using many servers, and we increased this threshold;

System	Settings	Machine hours	#Competitors
Calvin-TPCC	#WH={1, 2, 4…128}, #Node={1, 2, 4…32}, Cross-WH(%)={0, 10…100}	832 × 3	1
Silo-TPCC	#WH={1, 2, 4 … 512}, #Workers={1, 2, 4, 8, 16, 28}, #Txn-Types={5, 2, 1}, Cross-WH-NewOrder(%)={0, 10 … 100}	130 × 3	1
DrTM-TPCC	#Node={2, 4, 8}, #Workers={2, 4, 8, 16, 32}, #WHperNode={1, 2, 4 … 256}, Cross-WH(%)={0, 20 … 100}	192 × 3	1
Janus-TPCC	#Clients={1, 10, 20, 40, 80, 100, 200, 500, 1000, 2000, 4000} #Replica={1, 3, 5, 7}, #CPUperNode={1, 2, 4, 8, 16}, #WH={1, 2, 4· · · | #WH×#Replica/#CPUperNode ≤ 25}	1886 × 3	6
Cicada-TPCC	#WH={1, 2, 4…256}, #Workers={1, 2, 4, 8, 16, 28}, #Txn-Types={5, 2}	9 × 3	4
GAM-TPCC	#WH={2, 4, 8, 16, 32} / #Node, #Node={2, 4, 8}, Cross-WH(%)={0, 20, 40…100}, #Workers={1, 2, 4, 8, 16}, #Txn-Types={2}	721 × 3	3
Star-TPCC	#WH={16, 32, 128, 256}, #Workers={1, 2, 4, 8, 16}, Cross-WH(%)={0, 20, 40…100}, #Node={2, 4, 8}, #Txn-Types={2}	210 × 3	1
Aria-TPCC	#WH={1, 2, 4…1024}, Cross-WH(%)={0, 10…100}, #Nodes={1, 2, 4, 8} #Workers={1, 2, 4, 8, 12}, #Locks={1, 2, 4, 6|#Locks<#Workers}	360 × 3	3
Star-YCSB	#KV Pairs={3.2M, 6.4M, 25.6M, 51.2M }, #Workers={2, 8, 16}, Cross-partition(%)={0, 20, 40… 100}, Zipfθ={0, 0.4, 0.8, 0.99}, R/W Ratio={0/100, 20/80, 40/60, 100/0}, #Node={2, 4, 8}	384 × 3	3
Silo-YCSB	#KV Pairs={128K, 512K, 2M, 8M, 32M, 128M} #Workers={1, 2, 4, 8, 16, 28}, R/W Ratio={0/100, 20/80, 40/60, 100/0}	5 × 3	1
Cicada-YCSB	#KV Pairs={128K, 512K, 2M, 8M, 32M}, KV size={8B, 32B, 128B, 1KB}, #Workers={1, 4, 16, 28}, Zipfθ={0, 0.4, 0.8, 0.99}, R/W Ratio={0/100, 20/80, 40/60, 100/0}, batch size={1, 10, 100}	300 × 3	4
HERD-YCSB	#KV Pairs={6M, 24.5M, 96M, 384M, 1536M}, KV size={8B, 32B, 128B, 512B, 1KB}, Zipfθ={0, 0.4, 0.8, 0.99}, R/W Ratio={0/100, 20/80, 40/60, 100/0}	325 × 3	2
TAPIR-YCSB	#Clients={5, 10, 15, 20, 25}, #KV Pairs={128K, 512K, 2M, 8M, 32M}, R/W Ratio={0/100, 20/80, 40/60, 100/0}, (#shard, #replica)={(1, 3), (1, 5), (1, 7), (2, 3), (2, 5), (3, 3)}	1218 × 3	2
MySQL-TPCC	#WH={1, 2, 4 … 256}, #Terminals={1, 2, 4 … 512}, innodb_buffer_pool_chunk_size={1GB, 2GB, 4GB, 8GB}, innodb_buffer_pool_instances={1, 2, 4, 8}	548 × 3	1

Table 1. The settings and machine hours we use for each system, including its competitors. Some systems do not support the tuning of a certain parameter. We set the length of each experiment to be the same as that used in the original paper. We test each setting three times.

many systems use a timeout for different purposes and we sometimes need to tune the timeout interval for different settings. We were able to diagnose and fix most of these problems ourselves with a reasonable amount of effort. The only exception is DrTM, for which we requested help from its authors. Therefore, we expect that making a prototype ready for extensive evaluation usually should not require much additional effort from its authors.

We did encounter cases in which some systems become less reliable under certain new settings. We tried to diagnose their root causes, but due to their nondeterministic nature, we failed on most of them after a reasonable amount of effort. However, since the number of these problematic settings is limited, they did not prevent us from continuing our study.

Step 3. Incremental sampling. In the third step, we studied whether we can measure a subset of the settings we ran in the second step and still do a meaningful analysis on the results of these sampled settings. We compared our analysis result with the "ground truth" from the second step to determine whether our analysis is accurate. We present the details of this step in the next section.

3 Incremental Sampling to Reduce Experiment Time

In this section, we explore whether we can test a subset of settings and still get an overall picture of system performance characteristics under all settings. To achieve this, we measured the performance of the target system under a subset of settings, and then split the results into a training set and a validation set; if we can build a model on the training set to accurately predict the performance results (i.e., throughput) in the validation set, then we consider our model accurate enough to capture the overall system performance characteristics. Next, we discuss how to build the model, how to choose samples, and how to analyze the results in detail.

3.1 ANOVA Analysis

We used analysis of variance (ANOVA) [64] to build a model on the training set, because ANOVA is widely used, can show the impact of each parameter, can analyze the impact of the interaction of multiple parameters, and does not require any prior knowledge of data distribution, except for the assumption that the results of the same setting follow a normal distribution.

At a high level, ANOVA works similarly to regression algorithms, except that ANOVA focuses primarily on categorical factors. We treat all parameters as categorical factors in this work for two reasons: First, for most parameters, even if their values are numerical, their possible values are naturally non-continuous (e.g., we cannot have 1.5 warehouses in TPC-C). Second, treating parameters as categorical factors often makes it easier to identify non-linear relationships, as explained in the following example.

Assuming that each experiment has n parameters and a result (i.e., system throughput in this work) and we run m experiments in total, the input to ANOVA is 1) a $n \times m$ matrix, in which each row records the parameter values of each experiment, and 2) a vector with m experiment results. ANOVA will determine whether there are enough samples to reach a conclusion. If so, it will output the impacts of each parameter. ANOVA assumes that there are two sources of variability in an experiment: effects from parameters and effects from random error. With the results obtained from the experiments, it calculates the variances for different parameters, as well as the variance for the experiment results. Based on the ratio of these variances, it further determines the statistical significances of each parameter, i.e., if it has a significant influence on the experiment result.

To be concrete, ANOVA categorizes independent variables (experiment parameters in our case) into various groups and evaluates their effects on the response variable (experiment results in our case). The model does not presuppose a particular shape of the relationship between variables; but, it examines whether the means across different settings (groups) differ significantly. To fit the model, ANOVA selects subsets of independent variables, and calculates the mean differences within and between these groups. It then uses the F-statistic, derived from the ratio of mean squares between groups to that within groups, to test the hypothesis that no significant difference exists among group means. If the p-value obtained from this test is below the chosen threshold (0.05), it suggests that the current model settings significantly affect the response variable.

Here, we provide an example of the output of ANOVA, which is used in our following discussion. Suppose that our experiment has two parameters T and P, where T can be equal to 0, 10, and 20, and P can be 5 and 15. One possible model that is tested will look as follows, where μ is the intercept of Y, ε is the random error, $\beta_T T$ is the effect of the parameter T, $\beta_P P$ is the effect of the parameter P, and $\beta_{TP} TP$ is the cross (or interaction) effect of the parameters T and P.

$$Y = \mu + \beta_T T + \beta_P P + \beta_{TP} TP + \varepsilon \tag{1}$$

The output of ANOVA may look like the following, together with the significance of T, P, and TP.

$$Y = \mu + \begin{cases} \beta_{T_1}, (T = 0) \\ \beta_{T_2}, (T = 10) \\ \beta_{T_3}, (T = 20) \end{cases} + \begin{cases} \beta_{P_1}, (P = 5) \\ \beta_{P_2}, (P = 15) \end{cases} + \begin{cases} \beta_{T_1P_1}, (T = 0, P = 5) \\ \beta_{T_1P_2}, (T = 0, P = 15) \\ \quad\ldots\ldots \\ \beta_{T_3P_2}, (T = 20, P = 15) \end{cases} \tag{2}$$

We emphasize three points about ANOVA's output. First, if we treat T or P as a numerical factor, then ANOVA requires the result to have a linear relationship with the values of these factors, which may not be true. To identify non-linear relationships, the classic solution is to add non-linear terms like T^2, P^2, etc., which requires us to "guess" the shape of the formula. If we treat each as a categorical factor, as shown in this example, then ANOVA requires the result to have a linear relationship with a set of boolean values, each indicating whether the value falls into a category. This approach can potentially capture non-linear relationships when the parameter was treated as numerical (e.g., $\beta_{T_1} = 1, \beta_{T_2} = 100, \beta_{T_3} = 10$). This is one reason we treat all parameters as categorical in this work. However, this approach may incur several problems: First, when treating a factor as categorical, it is impossible to analyze what would happen between categories (e.g., what happens if $T = 5$), but considering most of our parameters have non-continuous values, this is acceptable. Second, it may lead to overfitting, which is discussed later.

Second, the coefficient β identifies the impact of each parameter or each interaction of parameters, and thus we can analyze the trend of the coefficient to understand how the impacts of parameters change (Section 3.3).

Third, one can control the accuracy of the model (i.e., ε) by choosing what parameters and interactions to test. If we choose all possible interactions, then we can reach 100% accuracy, but may cause overfitting. To avoid overfitting, we try 1) using only individual parameters and 2) using individual parameters and interactions of two parameters in ANOVA. We use 1) if 1) is accurate enough. We further remove insignificant parameters whose p-value is greater than 0.05 based on ANOVA. In the above example, we may find TP is insignificant and then rerun ANOVA with only T and P.

In our experiments, we find that including individual parameters and interactions of two parameters is generally good enough; adding interactions of three parameters often results in lower prediction accuracy, likely due to overfitting. However, this may be because our experiments test at most six parameters, and systems with more parameters might necessitate the inclusion of interactions of more than two parameters.

3.2 Choosing Samples

We adopt the incremental sampling approach (Algorithm 1): In each iteration, our algorithm splits the existing samples into a training set and a validation set (line 6), applies ANOVA to build a model on the training set (line 7), validates the model on the validation set (line 8), terminates when the R^2 value is good enough (lines 11-12), and adds more samples otherwise (lines 14-16). The basic idea behind this approach is that, if we can develop a model that can accurately predict system

Algorithm 1: Incremental Sampling Algorithm

Input: s: the starting sampling rate
δ: the number of settings to add in each step
Result: A representative sample dataset

```
1  current_settings ← choose_settings(∅, s);
2  D_exp ← run_experiemnts(current_settings);
3  do
4      results ← empty list;
5      repeat
6          training, validation ← split(D_exp);
7          model ← ANOVA_Analysis(training);
8          R^2 ← evaluate(model, validation);
9          Add R^2 to results;
10     until 10 times;
11     if All values in results ≥ threshold then
12         terminate;
13     else
14         new_settings ← choose_settings(current_settings, δ);
15         D_exp ← D_exp ∪ run_experiments(new_settings);
16         current_settings ← current_settings ∪ new_settings
17     end
18 while current_settings < all_settings;
19 return D_exp;
```

performance, we can then analyze the predicted performance to understand system characteristics and make comparisons, rather than running all experiments. In this section, we use random sampling when adding more samples and discuss other sampling methods in Section 4.1, assuming that we have no prior knowledge about how performance may change across different settings. If the experimenter has certain knowledge, she may prioritize certain settings, but this may increase the risk of a biased evaluation. For stability, in each iteration, this algorithm repeats the training and prediction procedure 10 times on different training sets and only terminates when all of them provide accurate estimates (lines 10 and 11).

In reality, we should run experiments to measure the system performance of newly chosen settings. In this work, since we have already run all settings (Section 2), we simply reuse results from the prior experiments.

3.3 Analyzing Parameter Impact

This section presents the analysis module we developed based on the result of ANOVA.

The coefficients from ANOVA analysis represent the impact of a parameter on the final results. If the ANOVA coefficients of a parameter monotonically increase or decrease with the values of the parameter, we try to find *knee points*, which means points with maximum curvature. In computer systems, such points represent operating points where the cost of changing a setting is no longer worth the performance benefit. Otherwise, we try to find *turning points*, where the ANOVA coefficients first increase with the parameter values but decrease after the turning points

Algorithm 2: Trend Analysis on ANOVA output

Data: A set of results from n significant terms $T = \{t_1, t_2, \ldots, t_n\}$, where

- $t.parm\_name$ is the name(s) of parameter p or parameter pair (p, q) associated with t. It has i distinct values.
- $t.parm\_val$ is the value corresponds to $t.parm\_name$.
  - $t.parm\_val =< p_1, p_2, \ldots, p_i >$ if t is one parameter, $< (p_1, q_1), (p_2, q_2), \ldots, (p_i, q_i) >$ if t is a parameter pair
- $t.coef =< c_1, c_2, \ldots, c_i >$ is ANOVA coefficients
- $t.per\_var\_explained$ is the percentage of variance explained by this term

Result: A set of important points Y

1 **foreach** $t_i \in T \wedge t_i.per\_var\_explained \geq 10\%$ **do**
2 **if** *t is a parameter pair (p, q)* **then**
3 $P \leftarrow$ distinct values of p in $t.parm\_val$;
4 $Q \leftarrow$ distinct values of q in $t.parm\_val$;
5 $Y_i \leftarrow \emptyset$;
6 **foreach** $p_j \in P$ **do**
7 $parms \leftarrow \{q_i | (p = p_j, q_i) \in t.parm\_val\}$;
8 $coef \leftarrow$corresponding subset of $t.coef$;
9 $Y_i \leftarrow Y_i \cup$ `FindImportantPoints`$(t.parm\_name, parms, coef)$
10 **end**
11 **foreach** $q_j \in Q$ **do**
12 $parms \leftarrow \{p_i | (p_i, q = q_j) \in t.parm\_val\}$;
13 $coef \leftarrow$corresponding subset of $t.coef$;
14 $Y_i \leftarrow Y_i \cup$ `FindImportantPoints`$(t.parm\_name, parms, coef)$
15 **end**
16 **else**
17 $Y_i \leftarrow$`FindImportantPoints`$(t.parm\_name, t.parm, t.coef)$
18 **end**
19 $Y \leftarrow Y \cup Y_i$
20 **end**
21 Rank important point results (if any) based on $turn\_ratio \times t.per\_var\_explained$

or vice versa. We are interested in these two types of points since they usually indicate a change in the behavior of the system. We refer to them as "important points" in the rest of this paper.

Algorithm 2 presents the overall algorithm. At a high level, for each parameter (note that we already filtered out insignificant parameters), we try to find the important points (Algorithm 3). We then rank the important points based on the product of the percentage of variance explained by this parameter and the turn ratio of the important points defined in line 15 of Algorithm 3.

For each significant parameter or interaction that explains at least 10% of the variance, we attempt to locate the important points within them. For individual parameters, we directly identify the important points based on their parameter values and ANOVA coefficients. For an interaction of parameters, we fix one parameter at a time while varying the other, which reduces the problem to finding important points on individual parameters.

Algorithm 3: FindImportantPoints

Input: $parm\_name$ is the name of this parameter,
$P =< p_1, p_2, \ldots, p_n >$ is the sorted parameter values,
$C =< c_1, c_2, \ldots, c_n >$ is the corresponding ANOVA coefficients

Output: A set of important points $Y = \{y_1, y_2, \ldots, y_m\}$, where

- $y.parm\_name = parm\_name$
- $y.type \in \{$"$turning$", "$knee$"$\}$ is the type of this point.
- $y.val$ is the parameter value of this point.
- $y.turn\_ratio$ measures the change of slope before and after this turning point

1 **if** `SpearmanRankCorrelation` $(P, C) = \pm 1$ **then** `// monotonically increasing or decreasing`
2 $y.val \leftarrow$ `FindKneePoints` (P, C) ; `// external library`
3 $y.type \leftarrow$"knee";
4 **else**
5 **for** $i \leftarrow 2$ ***to*** $n - 1$ **do**
6 $P_1, P_2 \leftarrow < p_1, \ldots, p_{i-1} >, < p_{i+1}, \ldots, p_n >$;
7 $C_1, C_2 \leftarrow < c_1, \ldots, c_{i-1} >, < c_{i+1}, \ldots, c_n >$;
8 $\Delta Spear_i \leftarrow$ `SpearmanRankCorrelation`(P_1, C_1) − `SpearmanRankCorrelation`(P_2, C_2) ;
9 **end**
10 $y.val \leftarrow P_j$, where j maximize $|\Delta Spear|$ and $|\Delta Spear| \geq 1$;
11 $y.type \leftarrow$"turning";
12 **end**
13 $P_a, P_b, C_a, C_b \leftarrow P$ and C divided by $y.val$;
14 $k_a, k_b \leftarrow$ the linear regression slope fitted by (P_a, C_a), (P_b, C_b) ;
15 $y.turn\_ratio \leftarrow |\frac{k_b - k_a}{k_a}|$;

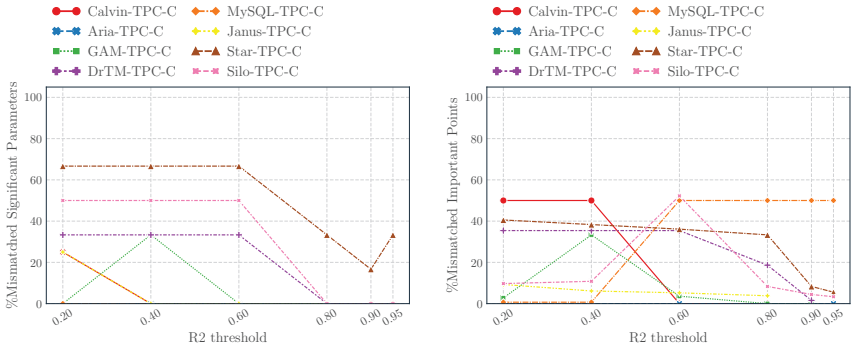
Identifying knee points (lines 1-3 in Algorithm 3). In a continuous function $f(x)$, knee point is defined as the point with maximum curvature, where curvature $K_f(x)$ is defined by f's first and second derivative [62].

$$K_f(x) = \frac{f''(x)}{(1 + f'(x)^2)^{1.5}} \tag{3}$$

For knee points detection in discrete data, we apply the Kneedle algorithm [63], in which a smoothing spline is used to preserve the shape of the original data set and knee points are approximated by the set of points in that spline curve that are local maxima if the curve is rotated by θ degrees clockwise about (x_{min}, y_{min}) through the line formed by (x_{min}, y_{min}) and (x_{max}, y_{max}).

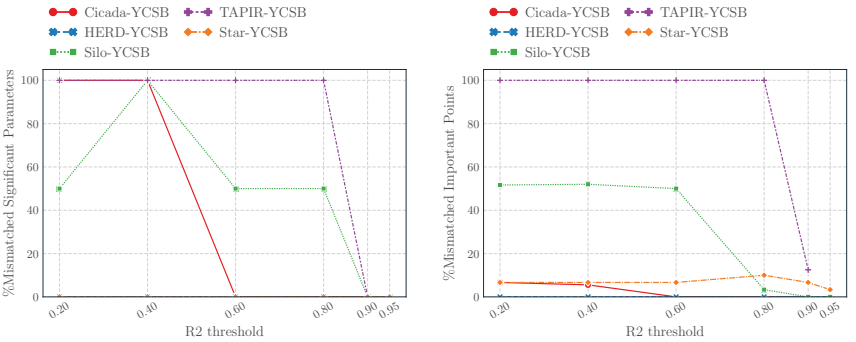
Identify turning points (lines 4-12 in Algorithm 3). *Spearman* correlation measures the monotonicity of a relationship between two variables [54], which ranges from -1 to +1. To detect turning points, we try to find a dividing point where the absolute difference in *Spearman* correlations before and after the dividing point is the largest and above 1.

The aforementioned algorithms assume that the values of the target parameters are ordered, which is the common case in our experiments. For other cases (e.g., a parameter to determine whether a system works under memory-only mode or persistent mode), one may design additional analysis components depending on the questions she wants to answer. For example, one can analyze



(a) %mismatched significant parameters, TPC-C.

(b) %mismatched important points, TPC-C.



(c) %mismatched significant parameters, YCSB.

(d) %mismatched important points, YCSB.

Fig. 1. Comparing analysis results on sampled data set and those on full data set under different R^2 thresholds. Their difference is quantified as the number of mismatched significant parameters and important points between sampled and full results (normalized by the total number of significant parameters and important points from the full results). Lower mismatch rates indicate the sampled data set can better approximate the full data set. Significant parameters are identified in ANOVA analysis. Important points are identified by Algorithm 3. Each number is the median of 100 trials.

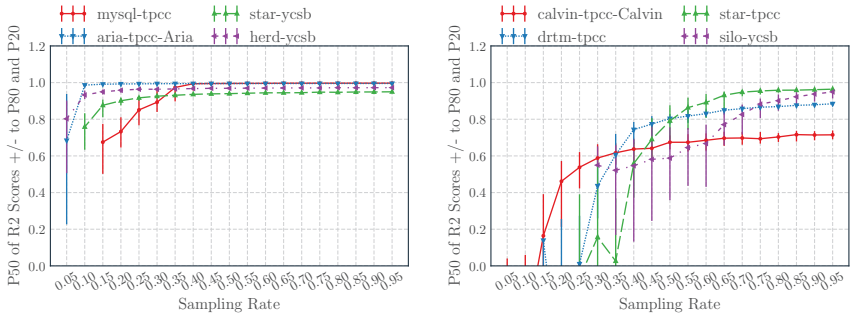
the difference between memory-only and persistent mode under different settings and further answer under which settings this mode makes no difference.

3.4 Results

We have tested incremental sampling on all systems to answer the following questions:

- What should the R^2 threshold be set at so that the samples can adequately approximate the full data set?
- How many samples are needed to achieve the desired R^2 threshold?
- For systems that require a large number of samples, is it possible to improve their predictability through better implementations?

R^2 threshold. To find the right R^2 threshold, we tested incremental sampling with different R^2 thresholds and compared the set of significant parameters and important points found on



(a) Highly predictable systems. (b) Less predictable systems.

Fig. 2. Model accuracy vs sampling rate. We categorize all systems into two types: A highly predictable system can be predicted (median R^2 score can reach at least 0.9) with less than 30% sample as shown in (a). A less predictable system (b) needs a higher sampling rate. We conducted 100 trials for each sampling rate, and for each trial, we draw the median, p20, and p80 accuracy (R^2). Some lines miss data at a low sampling rate due to ANOVA reporting insufficient samples.

the full data set ($Y_{full-para}$ and $Y_{full-pts}$), and those found in the sampled data set ($Y_{sample-para}$ and $Y_{sample-pts}$). We then calculated mismatched significant parameters or important points as $(Y_{full} - Y_{sample}) \cup (Y_{sample} - Y_{full})$. Since important points are identified after identifying significant parameters, mismatch of important points and significant parameters needs to be interpreted together. For instance, the rise of important points mismatch in Figure 1b at R^2 $threshold = 0.6$ for MySQL and Silo is due to the change of significant parameters as shown in Figure 1a.

As shown in Figure 1, a termination condition of $R^2 > 0.9$ is sufficient for most systems to reach a low mismatch rate. There are two exceptions: For MySQL in Figure 1b, our investigation reveals that the high mismatch rate is due to a knee point in the full dataset being identified as a turning point in some sampled datasets, with the classification being close to the threshold. For Star in Figure 1a, our investigation shows that, similarly, several significant parameters have a significance level close to the threshold. In short, these two exceptions are caused by values close to the threshold, which may be addressed by better threshold selection algorithms.

Desired number of samples. For each system, we start from 5% random samples and add 5% more random samples in each step, until we reach a 95% sampling rate. For each sampled set, we record its R^2 as the lowest one across ten splits to match Algorithm 1. For each system, we repeat the above procedure 100 times.

Figure 2 shows the results (we ignore ones that report not enough samples). For each sampling rate, we show the median R^2, the 20th percentile R^2, (p20), and the 80th percentile R^2 (p80) among 100 trials. From the predictability perspective only, we categorize all systems into two kinds: highly predictable ones that can reach $R^2 > 0.9$ usually with less than 30% sampling rate, and less predictable ones that either require more samples (usually more than 50% sampling rate) to reach $R^2 > 0.9$ or never reach $R^2 > 0.9$. For readability, we only display four systems per kind.

Combined with the results from Figure 1, we can conclude that for highly predictable systems, it is possible to perform a reasonably accurate analysis with a low sampling rate.

What to do if the system is not very predictable? Unpredictability can be caused by implementation issues and/or inappropriate prediction models. It is challenging to give a decisive conclusion, given the impossibility of exhausting all prediction models and the considerable effort required to

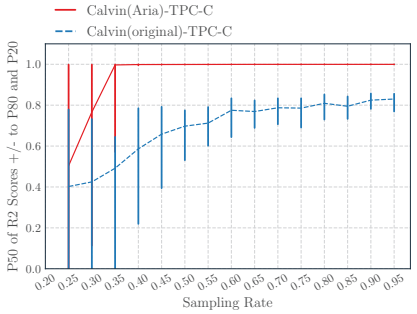


Fig. 3. Comparison between original Calvin and a new Calvin implementation from Aria (ANOVA + random sampling). Note that the Calvin (original) line in this figure is slightly different from that in Figure 2b, since in this figure, we can only compare settings that are supported by both versions.

improve and understand the implementation. Fortunately, we find that, for comparison purposes, some systems re-implemented prior works for various reasons. This allows us to compare the re-implemented versions with the original versions. For example, as shown in Figure 3, the Calvin version implemented by the Aria authors seems to provide better predictability than the original version. Therefore, our recommendation is that, if an experimenter finds her system to be not very predictable during incremental sampling (i.e., R^2 is low despite using a high sampling rate), she should investigate the possibility of implementation issues causing unpredictability before blindly adding more samples. In particular, she should verify whether the results from the finished experiments match with her expectations and whether there are explanations for performance changes at certain settings.

4 Exploring Other Methods

In this section, we explore other sampling and prediction methods.

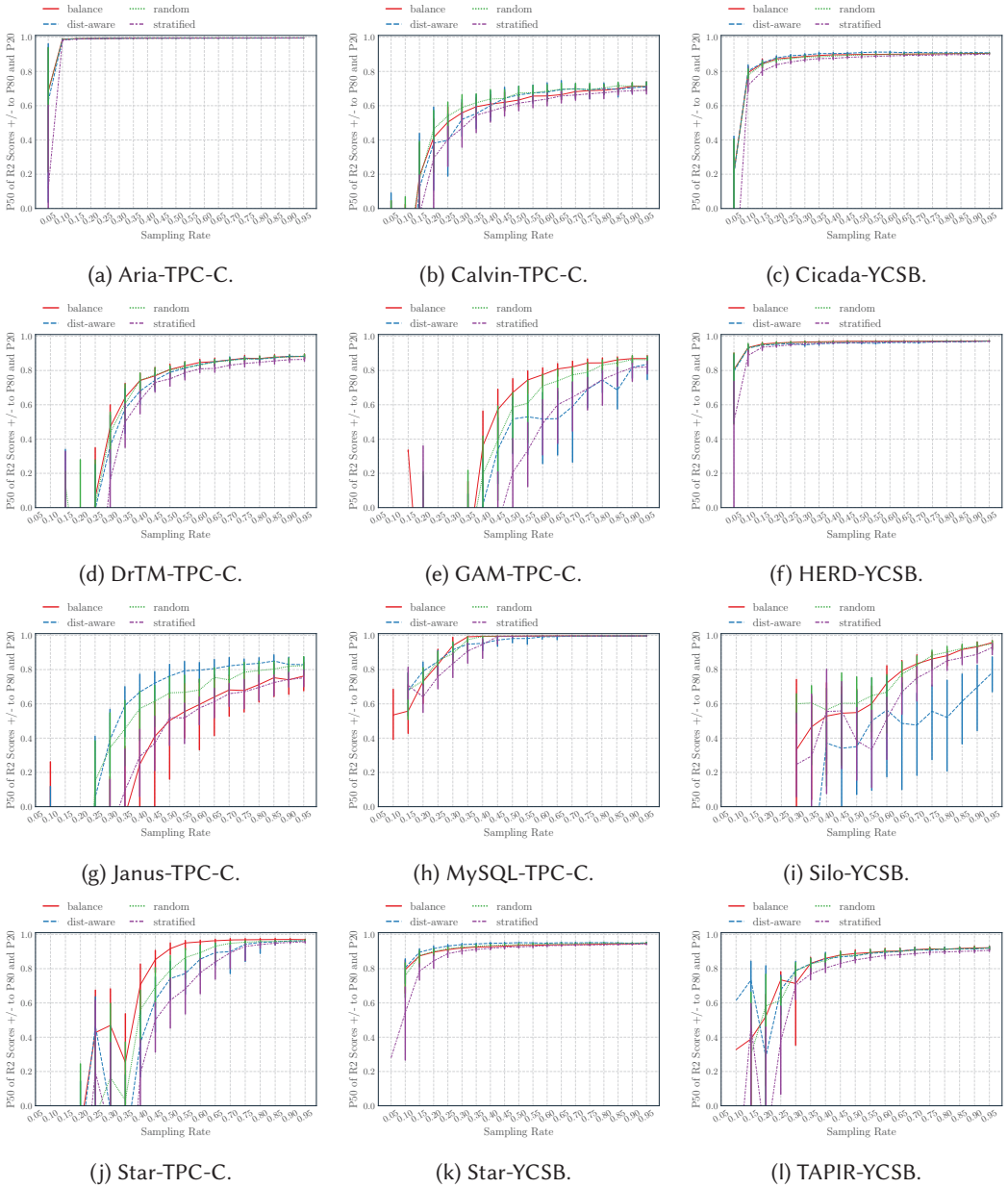
4.1 Sampling Methods

In addition to random sampling, we explore three different sampling methods. We want to investigate whether we can further reduce the evaluation cost of incremental sampling.

Balanced sampling. This approach aims to ensure that each value of a parameter is sampled with a similar frequency. Initially, it picks 5% of the whole dataset randomly as the sampling set. Then, it analyzes the frequency of values of each parameter in the sampling set. When expanding the sampling set, it first selects the setting with the least frequently appearing value of each parameter, updates the frequency, and then repeats the procedure to select the least frequently appearing values.

Stratified sampling. This method aims to ensure that each value of a parameter has the same appearance rate in both the sampling set and the testing set [56]. More formally, given a parameter P with k possible values, stratified sampling method tries to ensure that $\frac{N_{k_i\_sampling}}{N_{sampling}} = \frac{N_{k_i\_testing}}{N_{testing}}$, where $N_{k_i\_sampling}$ and $N_{k_i\_testing}$ represent the number of times k_i appears in the sampling set and the test set, respectively, and $N_{sampling}$ and $N_{testing}$ represent the size of the sampling set and the test set, respectively.

Distribution–aware sampling. This method aims to sample more heavily in areas where values change more abruptly. It starts by randomly selecting a small subset of settings as the first sampling set. Then when adding more samples, for each setting in the unsampled set, it 1) identifies 2^k closest



(a) Aria-TPC-C. (b) Calvin-TPC-C. (c) Cicada-YCSB.

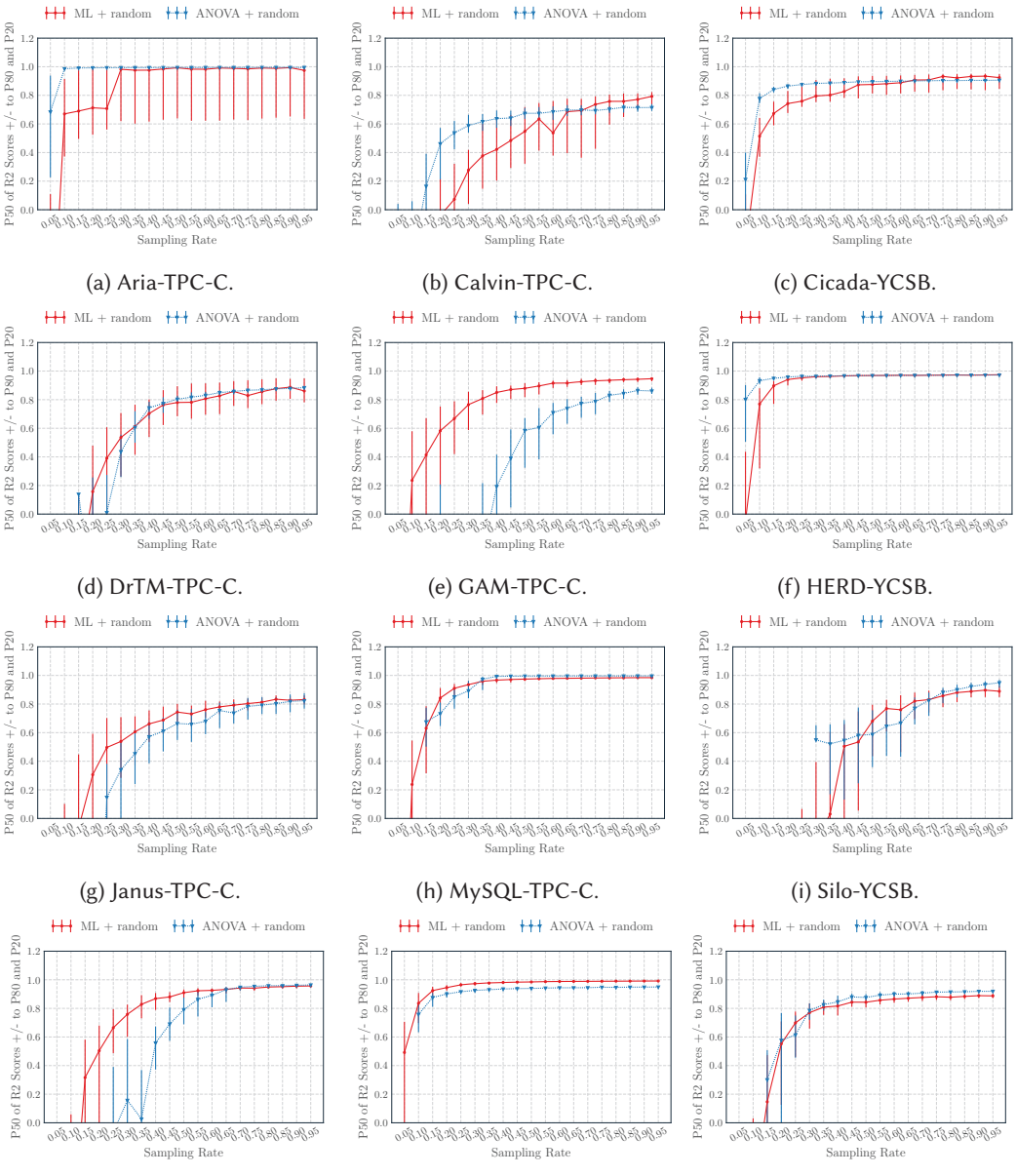
(d) DrTM-TPC-C. (e) GAM-TPC-C. (f) HERD-YCSB.

(g) Janus-TPC-C. (h) MySQL-TPC-C. (i) Silo-YCSB.

(j) Star-TPC-C. (k) Star-YCSB. (l) TAPIR-YCSB.

Fig. 4. The effects of different sampling methods. The best sampling method varies across different systems, and no sampling method consistently outperforms random sampling.

settings in the sampled set as a neighbor set $\mathcal{N}$, where k is the number of parameters in that system; 2) calculates the neighborhood variability score for that neighbor set, which is computed as $avg(\frac{|y_m - y_n|}{dist(x_m, x_n)}), \forall (m, n) \in \mathcal{N}$; 3) uses the neighborhood variability score as weights, and performs weighted random selection in unsampled settings.



(a) Aria-TPC-C. (b) Calvin-TPC-C. (c) Cicada-YCSB.

(d) DrTM-TPC-C. (e) GAM-TPC-C. (f) HERD-YCSB.

(g) Janus-TPC-C. (h) MySQL-TPC-C. (i) Silo-YCSB.

(j) Star-TPC-C. (k) Star-YCSB. (l) TAPIR-YCSB.

Fig. 5. ML vs ANOVA (random sampling). No method consistently outperforms the other.

Figure 4 presents a subset of the results of running ANOVA with different sampling methods. Some sampling methods are significantly affected by the random seed, so we conducted 100 runs for each sampling method using different random seeds. For a combination of system and workload, we define its best sampling method as the one that can reach $R^2 > 0.9$ with the lowest sampling rate (note that a tie is possible). Overall, the best sampling method varies between different systems,

and we have not found a sampling method that can consistently outperform random sampling. Specifically, random sampling is the best on Aria-TPC-C, HERD-YCSB, Silo-YCSB, Silo-TPC-C, Star-YCSB and TAPIR-YCSB; balanced sampling is the best on Aria-TPC-C, HERD-YCSB, MySQL-TPC-C, Star-TPC-C and TAPIR-YCSB; distribution-aware sampling is the best on Aria-TPC-C, Cicada-YCSB, HERD-YCSB, MySQL-TPC-C and Star-YCSB; stratified sampling is the best on Aria-TPC-C; in other combinations of systems and workloads, no sampling method can reach $R^2 > 0.9$. Note that some system-workload combination might have multiple best sampling methods.

4.2 Prediction Methods

In addition to ANOVA, we have also explored 1) machine learning methods, which will be presented in detail in this section; 2) linear regression treating all parameters as numerical data (note that we configure ANOVA to treat all parameters as categorical data); and 3) Lorenzo Predictor, which is typically applied in compression algorithms [38, 69]. Since linear regression and Lorenzo Predicator do not perform well in our experiments, we do not present them in detail in this paper.

Machine learning. We chose neural networks as the prediction method due to its popularity and capability. The neural network we used, Multilayer Perceptron (MLP), is the standard model for performance prediction. We utilized scikit-learn [57] to prepare the training data and train the model. Our model contains three hidden layers with 10, 10, and 5 neurons, respectively. The model is trained with the lbfgs optimizer, which performs better for small datasets [57].

Figure 5 shows a subset of results of running different prediction methods. To summarize, ML is the best on Cicada-YCSB, GAM-TPC-C, MySQL-TPC-C, Star-TPC-C, Star-YCSB; ANOVA is the best on Aria-TPC-C, Cicada-YCSB, Herd-YCSB, Silo-TPC-C, Silo-YCSB, and Tapir-YCSB; and in other combinations of systems and workloads, no prediction method can reach $R^2 > 0.9$. Again, there is no consistent trend that one can outperform the other. We further compared these two methods with different sampling methods and found that no combination consistently outperforms others. This conclusion is consistent with a prior study [42], which examines different sampling and ML methods for performance prediction and finds no single best combination. Since ANOVA is more efficient than MLP, and more importantly, can further explain the impact of each parameter, we opt for ANOVA in the rest of this paper.

4.3 Discussion

Due to the large number of sampling and prediction methods, it is challenging, if not impossible, to exhaustively try every sampling and prediction method. Therefore, our effort should be viewed as an attempt to explore this direction, rather than as a decisive conclusion. In particular, for many systems, there is a great gap between p80 R^2 and p20 R^2 of the same sampling rate, indicating possible improvement.

On the other hand, we observed that those highly predictable systems exhibit good predictability no matter which sampling method or prediction method we use. Therefore, we expect that the implementation of the system will still play an important role in the predictability of the system.

5 Comparison With Ground Truth

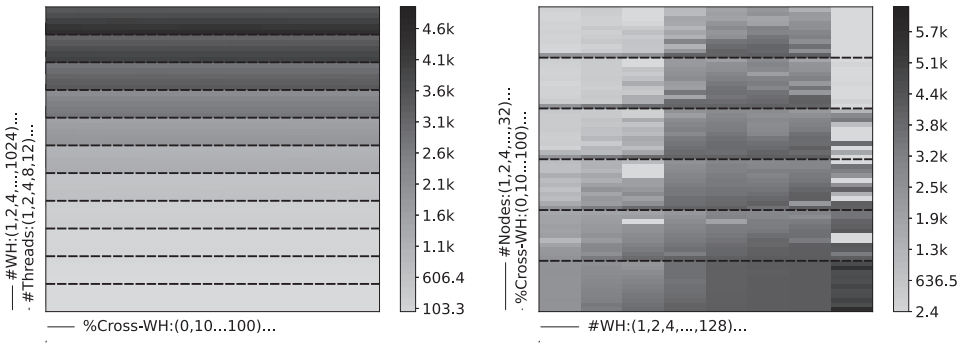
In this section, we compare the results of incremental sampling to the "ground truth", using the full results. We perform the comparison to validate that the trends or important points found by incremental sampling are indeed observable in the ground truth.

Since these systems often have multiple parameters, to present the full results, we visualize high-dimensional data in the following way (e.g., Figure 6 and Figure 7): Assuming the system has n parameters from p_1 to p_n, in the first iteration, our tool picks two parameters (e.g., p_1 and p_2) and

groups all data with the same p_3 to p_n values; then it is able to draw each group on a 2D figure called a "tile" since each group only contains two parameters p_1 and p_2. In the second iteration, our tool picks two additional parameters (e.g., p_3 and p_4) and groups tiles with the same p_5 to p_n values into a super-tile. This procedure repeats until it covers all parameters (the last iteration may just pick one parameter).

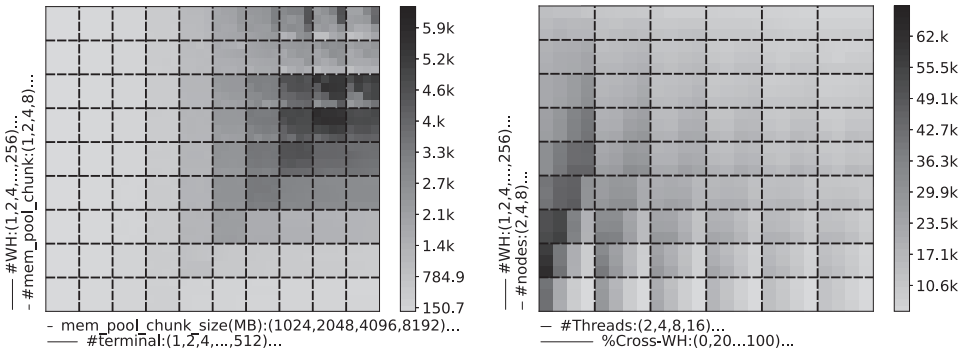
By comparing the visualized data with the output of incremental sampling. We try to answer the following questions:

- Does the visualization align with our previous observation that some systems are more predictable than others?
- Does the output of incremental sampling align with the ground truth, and does it provide valuable information?



(a) Per core throughput of Aria TPC-C. (b) Per core throughput of Calvin TPC-C.

Fig. 6. Predictability of different systems. Per core throughput is reported in txns/sec.



(a) Per core throughput of MySQL TPC-C. (b) Per core throughput of DrTM TPC-C.

Fig. 7. Systems with complex performance patterns. Per core throughput is reported in txns/sec.

5.1 Predictability of Different Systems

As discussed in Section 3.4, we find that some systems are less predictable than others. This observation can be validated with the visualized data. For example, as shown in Figure 6a, Aria, which exhibits good predictability as shown in Section 3.4, has a regular pattern in its visualized

data. Calvin, on the other hand, has irregular patterns in its visualized data, which is consistent with its less ideal predictability as shown in Section 3.4.

Again, we emphasize that the predictability of a system is dependent on both the sampling method and the prediction model. Although this work has explored several methods, it is possible that more effective methods exist. Therefore, the discussion here should be considered as a best-effort validation.

5.2 Output of Incremental Sampling

To verify the accuracy of incremental sampling, we validate whether the significant terms and important points reported by incremental sampling match the visualized data. We report results on MySQL and DrTM here, as these two exhibit complex patterns in their visualized data.

For MySQL, our analysis shows that the most significant parameter is the number of terminals (i.e., clients or worker threads since MySQL creates one worker thread for each client), which shows an increasing trend with a knee point. This is consistent with the pattern in Figure 7a, where the right side shows higher throughput but the increasing trend stops at a certain point. Our analysis shows that the second most significant parameter is the combination of (number of terminals, number of warehouses), which includes turning points. This is consistent with the figure's pattern that the highest throughput appears in the middle of the y-axis.

Both findings are consistent with common knowledge or manual investigation as well: An experiment usually requires a certain number of terminals to saturate the system, but beyond that point, adding more terminals won't further increase system throughput. That's why the number of terminals shows an increasing trend with a knee point. More warehouses will decrease contention, thereby increasing throughput, which is why we can observe that throughput increases with more warehouses until a certain point in Figure 7a. With more warehouses, we need more terminals to saturate the system, which is why we observe a combined effect of (number of terminals, number of warehouses). To understand why throughput decreases with more warehouses after a certain point, we profile MySQL and find that the lookup overhead in a tree-like index will grow with more rows in the database. Therefore, beyond a certain point, the lookup overhead will become the dominant factor, explaining the observed turning point.

For DrTM TPC-C, our analysis shows that the most significant parameter is the number of worker threads, with a decreasing trend, which is consistent with Figure 7b. This is because DrTM is a distributed database using transactional memory and RDMA techniques, and more worker threads on the same node will contend on such shared resources. The second parameter is the cross-warehouse ratio, with a decreasing trend and a knee point. More cross-warehouse transactions are well known to introduce more contention and more network traffic, which explains why it decreases throughput. The third parameter is the combination of (the number of warehouses and the number of worker threads) with turning points. This is consistent with the step-like patterns in Figure 7b. The explanation is the same as for MySQL.

For other systems, we observed a similar match between observations on the sampled settings and those on all settings, although, once again, the required sampling rate to reach the match varies by different systems.

5.3 Discussion and Recommendation

The effect of the number of warehouses in TPC-C is a typical example of how experiment results can be affected by parameter selection: When this number is small, contention is likely to be the main bottleneck, and when this number becomes large, index searching is likely to become the main bottleneck. Therefore, the optimization of concurrency control may show superior improvement when this number is small, but the improvement will diminish when this number becomes large.

Furthermore, whether to partition data (i.e., whether to store all data in a single table or in multiple ones) further complicates the comparison, since partitioning naturally limits the size of an index.

For an unbiased evaluation, our general recommendation is to measure and report results on both sides of an important point. When comparing two systems, such measurements and comparisons should cover the two sides of a turning/kneeing point in both systems. More formally, for a parameter P, if system A has a turning point P_A and system B has a turning point P_B (assuming $P_A < P_B$), then the comparison should cover (min, P_A), (P_A, P_B), and (P_B, max).

6 Suggestions, Limitations, and Future Work

In this section, we summarize our recommendations about artifact implementation and evaluation. We further discuss the limitations of our approach and future work.

- A researcher should prepare her artifact for extensive evaluation. This often includes eliminating hard-coded configuration parameters. She should also clarify the value range of each configuration parameter the artifact can support.
- An experimenter should design evaluations to cover the entire parameter space, including the possible range of each parameter. If the space is too large for an extensive evaluation, the experiments can use the incremental sampling approach discussed in this work, starting from a small number of samples and adding more samples if existing samples do not provide a good estimation. We recommend random sampling and ANOVA analysis based on our experience.
- If at some point, those samples could provide a good estimation (we recommend using an R^2 threshold of 0.9), then the experimenter can further analyze the output of ANOVA to identify the important points.
- If the experimenter cannot obtain a good estimation with many samples (the exact threshold depends on how many experiments the experimenter can afford to run in total), she should investigate whether this is due to implementation issues. For this purpose, she can check whether the results of existing samples meet her expectations. If not, she should address the implementation issues and retry.
- For an unbiased evaluation, the experimenter should report results on both sides of an important point. When comparing two systems, the experimenter can apply the above approach to two systems independently, possibly with different sampling rates for different systems, and then compare their predicted performance. The comparison should cover both sides of important points in both systems.

The effectiveness of this approach depends on the predictability of the target system. Incremental sampling can significantly reduce the experiment cost for highly predictable systems. Less predictable systems, however, will bring at least two problems, due to the inherent limitations of the sampling approach: First, incremental sampling may observe the unpredictability and require more samples, eventually running out of the experimenter's resource. Second, the sampling approach may miss important points if there are too few samples and the performance of the target system can change abruptly. For example, if we test two configuration values $v1$ and $v2$ ($v1 < v2$), but the system performance drops at $v3$ and goes back to normal at $v4$ ($v1 < v3 < v4 < v2$), then the sampling approach will miss this drop. On the other hand, a system whose performance can change abruptly is perhaps undesirable in the first place.

In the future, we plan to investigate the applicability of our approach on other metrics, such as latency, which may have more variance than average throughput. For highly predictable systems, we will investigate ways to lower the cost of each sample's experiments, such as early stopping when early experimental results match expectations.

7 Related Work

Benchmarks. Many areas have well-established benchmarks for performance evaluation and comparison, such as the TPC series [72], SmallBank [8], YCSB [20], fio [27], the SPEC series [66], HPL [58], HPCG [25], Graph500 [31], MLPerf [51, 59], DAWNBench [19], and many others. Most of these benchmarks have restrictions on which parameters can be tuned and what values can be used for these parameters.

Reproducibility. To improve code quality and facilitate future research, our community has encouraged research papers to verify their reproducibility through artifact evaluation [1, 3–6]. In the meantime, many works study how to improve the reproducibility of research prototypes. For example, CloudLab studies the performance variability of its experiments and proposes guidelines to reduce such variability [50]; a recent work studies how cloud-based big data workload can be affected by network variability [74]; Fursin discussed the challenges and solutions from his experience of reproducing 150+ research papers [29]; Ursprung [61] studies how to improve the reproducibility of data science workloads. The key idea of Ursprung is to transparently capture provenance and build lineage to automatically track static and runtime configuration parameters of data science pipelines. With Ursprung, the authors hope to improve the reproducibility of data science workloads.

However, reproducibility does not answer the question of which settings we should use in experiments. Compared to these efforts, our work further argues that, for the purpose of improving code quality and facilitating future research, we should encourage extensive evaluations to cover a wider range of settings.

Regression/Machine learning for systems. Regression and machine learning methods have been applied in systems designs and optimizations. For example, many studies [13, 76, 80] propose using regression-based algorithms to tune systems configurations to achieve better performance. Recent systems research successfully exploits machine learning methods for performance prediction [28, 32, 42, 44], resource management [23, 37, 48, 52], scheduling [22, 49], I/O optimizations [24, 33], and so on. Our work explores whether we can apply a similar idea to reduce the cost of the extensive evaluation. Our work will benefit from improvement in this field, such as better prediction models or designs that can improve performance predictability.

8 Conclusion

This paper explores whether we can leverage sampling and prediction to reduce the cost of extensive evaluation. Our exploration shows that it is possible to gain the benefits of extensive evaluation with a low cost, but that may require a good engineering effort to improve the predictability of the artifacts. Based on our exploration, we further discuss possible future directions and make two recommendations: 1) We may encourage artifact predictability in addition to reproducibility, and 2) Measurement and comparison should cover both sides of a knee or turning point.

Acknowledgments

We thank the anonymous reviewers for valuable feedback. We thank Dr. Xingda Wei for his help on DrTM. This material is based in part upon work supported by the National Science Foundation under Grant Numbers CCF-2118745 and OAC-2333324.

References

[1] ACM SIGMOD Reproducibility. https://reproducibility.sigmod.org/.
[2] Calvin Source Code. https://github.com/yaledb/calvin.
[3] EuroSys Call for Artifacts. https://sysartifacts.github.io/eurosys2021/.

[4] OSDI Call for Artifacts. https://www.usenix.org/conference/osdi21/call-for-artifacts.
[5] PVLDB Reproducibility. http://vldb.org/pvldb/reproducibility/.
[6] SOSP Call for Artifacts. https://sysartifacts.github.io/sosp2021/call.html.
[7] Tapir Source Code. https://github.com/UWSysLab/tapir.
[8] Mohammad Alomari, Michael J. Cahill, Alan D. Fekete, and Uwe Röhm. The Cost of Serializability on Platforms That Use Snapshot Isolation. In *ICDE*, pages 576–585. IEEE Computer Society, 2008.
[9] Aria Source Code. https://github.com/luyi0619/aria.
[10] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload Analysis of a Large-scale Key-value Store. In *SIGMETRICS*, pages 53–64. ACM, 2012.
[11] Tiemo Bang, Norman May, Ilia Petrov, and Carsten Binnig. The Tale of 1000 Cores: An Evaluation of Concurrency Control on Real(ly) Large Multi-socket Hardware. In *DaMoN*, pages 3:1–3:9. ACM, 2020.
[12] Tiemo Bang, Norman May, Ilia Petrov, and Carsten Binnig. The Full Story of 1000 Cores: An Examination of Concurrency Control on Real(ly) Large Multi-socket Hardware. *The VLDB Journal*, 31(6):1185–1213, apr 2022.
[13] Zhendong Bei, Zhibin Yu, Huiling Zhang, Wen Xiong, Cheng-Zhong Xu, Lieven Eeckhout, and Shengzhong Feng. RFHOC: A Random-Forest Approach to Auto-Tuning Hadoop's Configuration. *IEEE Trans. Parallel Distributed Syst.*, 27(5):1470–1483, 2016.
[14] Benjamin Berg, Daniel S. Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, and Gregory R. Ganger. The CacheLib Caching Engine: Design and Experiences at Scale. In *OSDI*, pages 753–768. USENIX Association, 2020.
[15] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. Efficient Distributed Memory Management with RDMA and Caching. *Proc. VLDB Endow.*, 11(11):1604–1617, 2018.
[16] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A Distributed Storage System for Structured Data. In *OSDI*, pages 205–218. USENIX Association, 2006.
[17] Cicada Source Code. https://github.com/efficient/cicada-exp-sigmod2017.
[18] Cloudlab. https://www.cloudlab.us/.
[19] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Re, and Matei Zaharia. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. https://cs.stanford.edu/~deepakn/assets/papers/dawnbench-sosp17.pdf, 2017.
[20] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *SoCC*, pages 143–154. ACM, 2010.
[21] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *SOSP*, pages 205–220. ACM, 2007.
[22] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *ASPLOS*, pages 77–88. ACM, 2013.
[23] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. In *ASPLOS*, pages 127–144. ACM, 2014.
[24] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-Learning Congestion Control. In *NSDI*, pages 343–356. USENIX Association, 2018.
[25] Jack J. Dongarra, Michael A. Heroux, and Piotr Luszczek. High-performance Conjugate-gradient Benchmark: A New Metric for Ranking High-performance Computing Systems. *Int. J. High Perform. Comput. Appl.*, 30(1):3–10, 2016.
[26] DrTM Source Code. https://github.com/SJTU-IPADS/drtm.
[27] fio - Flexible I/O Tester. https://github.com/axboe/fio.
[28] Silvery Fu, Saurabh Gupta, Radhika Mittal, and Sylvia Ratnasamy. On the Use of ML for Blackbox System Performance Prediction. In *NSDI*, pages 763–784. USENIX Association, 2021.
[29] Fursin, Grigori. Reproducing 150 Research Papers and Testing Them in the Real World: Challenges and Solutions. https://learning.acm.org/binaries/content/assets/leaning-center/webinar-slides/2021/grigorifursin_techtalk_slides.pdf, 2021.
[30] GAM Source Code. https://github.com/ooibc88/gam.
[31] Graph500 Committee. Graph500 benchmark. http://graph500.org.
[32] Alexander Grebhahn, Norbert Siegmund, and Sven Apel. Predicting Performance of Software Configurations: There is no Silver Bullet. *CoRR*, abs/1911.12643, 2019.
[33] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *OSDI*, pages 173–190. USENIX Association, 2020.

[34] Rachael Harding, Dana Van Aken, Andrew Pavlo, and Michael Stonebraker. An Evaluation of Distributed Concurrency Control. *Proc. VLDB Endow.*, 10(5):553–564, 2017.
[35] Gernot Heiser. Systems Benchmarking Crimes. https://gernot-heiser.org/benchmarking-crimes.html.
[36] HERD Source Code. https://github.com/efficient/HERD.
[37] Henry Hoffmann. Jouleguard: energy guarantees for approximate applications. In *SOSP*, pages 198–214. ACM, 2015.
[38] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. Out-of-core Compression and Decompression of Large n-dimensional Scalar Fields. *Comput. Graph. Forum*, 22(3):343–348, 2003.
[39] Raj Jain. *The Art of Computer Systems Performance Analysis - Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley professional computing. Wiley, 1991.
[40] Janus Source Code. https://github.com/NYU-NEWS/janus.
[41] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA Efficiently for Key-value Services. In *SIGCOMM*, pages 295–306. ACM, 2014.
[42] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, and Sven Apel. The Interplay of Sampling and Machine Learning for Software Performance Prediction. *IEEE Softw.*, 37(4):58–66, 2020.
[43] Sangmin Lee, Zhenhua Guo, Omer Sunercan, Jun Ying, Thawan Kooburat, Suryadeep Biswal, Jun Chen, Kun Huang, Yatpang Cheung, Yiding Zhou, Kaushik Veeraraghavan, Biren Damani, Pol Mauri Ruiz, Vikas Mehta, and Chunqiang Tang. Shard Manager: A Generic Shard Management Framework for Geo-distributed Applications. In *SOSP*, pages 553–569. ACM, 2021.
[44] Chieh-Jan Mike Liang, Zilin Fang, Yuqing Xie, Fan Yang, Zhao Lucis Li, Li Lyna Zhang, Mao Yang, and Lidong Zhou. On Modular Learning of Distributed Systems for Predicting End-to-End Latency. In *NSDI*, pages 1081–1095. USENIX Association, 2023.
[45] Hyeontaek Lim, Michael Kaminsky, and David G. Andersen. Cicada: Dependably Fast Multi-Core In-Memory Transactions. In *SIGMOD Conference*, pages 21–35. ACM, 2017.
[46] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. Aria: A Fast and Practical Deterministic OLTP Database. *Proc. VLDB Endow.*, 13(11):2047–2060, 2020.
[47] Yi Lu, Xiangyao Yu, and Samuel Madden. STAR: Scaling Transactions through Asymmetric Replication. *Proc. VLDB Endow.*, 12(11):1316–1329, 2019.
[48] Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, and Colin Raffel. Learning-based Memory Allocation for C++ Server Workloads. In *ASPLOS*, pages 541–556. ACM, 2020.
[49] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *SIGCOMM*, pages 270–288. ACM, 2019.
[50] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, Robert Ricci, and Ana Klimovic. Taming Performance Variability. In *OSDI*, pages 409–425. USENIX Association, 2018.
[51] Peter Mattson, Christine Cheng, Gregory F. Diamos, Cody Coleman, Paulius Micikevicius, David A. Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim M. Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. MLPerf Training Benchmark. In *MLSys*. mlsys.org, 2020.
[52] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. CALOREE: Learning Control for Predictable Latency and Low Energy. In *ASPLOS*, pages 184–198. ACM, 2018.
[53] Shuai Mu, Lamont Nelson, Wyatt Lloyd, and Jinyang Li. Consolidating Concurrency Control and Consensus for Commits under Conflicts. In *OSDI*, pages 517–532. USENIX Association, 2016.
[54] Leann Myers and Maria J Sirois. Spearman Correlation Coefficients, Differences between. *Encyclopedia of Statistical Sciences*, 12, 2004.
[55] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling Memcache at Facebook. In *NSDI*, pages 385–398. USENIX Association, 2013.
[56] Van L Parsons. Stratified Sampling. *Wiley StatsRef: Statistics Reference Online*, pages 1–11, 2014.
[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
[58] Petitet, A. and Whaley, R. C. and Dongarra, Jack and Cleary, A. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. https://www.netlib.org/benchmark/hpl/.
[59] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius

Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. MLPerf Inference Benchmark. In *ISCA*, pages 446–459. IEEE, 2020.

[60] Kun Ren, Alexander Thomson, and Daniel J. Abadi. An Evaluation of the Advantages and Disadvantages of Deterministic Database Systems. *Proc. VLDB Endow.*, 7(10):821–832, 2014.

[61] Lukas Rupprecht, James C. Davis, Constantine Arnold, Yaniv Gur, and Deepavali Bhagwat. Improving Reproducibility of Data Science Pipelines through Transparent Provenance Capture. *Proc. VLDB Endow.*, 13(12):3354–3368, 2020.

[62] Stan Salvador and Philip Chan. Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms. In *ICTAI*, pages 576–584. IEEE Computer Society, 2004.

[63] Ville Satopaa, Jeannie R. Albrecht, David E. Irwin, and Barath Raghavan. Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior. In *ICDCS Workshops*, pages 166–171. IEEE Computer Society, 2011.

[64] Henry Scheffe. *The Analysis of Variance*, volume 72. John Wiley & Sons, 1999.

[65] Silo Source Code. https://github.com/stephentu/silo.

[66] Standard Performance Evaluation Corporation. https://www.spec.org/.

[67] Star Source Code. https://github.com/luyi0619/star.

[68] Takayuki Tanabe, Takashi Hoshino, Hideyuki Kawashima, and Osamu Tatebe. An Analysis of Concurrency Control Protocols for In-Memory Databases with CCBench. *CoRR*, abs/2009.11558, 2020.

[69] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. In *IPDPS*, pages 1129–1139. IEEE Computer Society, 2017.

[70] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. Calvin: Fast Distributed Transactions for Partitioned Database Systems. In *SIGMOD Conference*, pages 1–12. ACM, 2012.

[71] Transaction Processing Performance Council. The TPC-C home page. http://www.tpc.org/tpcc/.

[72] Transaction Processing Performance Council. The TPC home page. http://www.tpc.org.

[73] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy Transactions in Multicore In-Memory Databases. In *SOSP*, pages 18–32. ACM, 2013.

[74] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan S. Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. Is Big Data Performance Reproducible in Modern Cloud Networks? In *NSDI*, pages 513–527. USENIX Association, 2020.

[75] Yang Wang, Miao Yu, Yujie Hui, Fang Zhou, Yuyang Huang, Rui Zhu, Xueyuan Ren, Tianxi Li, and Xiaoyi Lu. A Study of Database Performance Sensitivity to Experiment Settings. *Proc. VLDB Endow.*, 15(7):1439–1452, 2022.

[76] Md. Wasi-ur-Rahman, Nusrat Sharmin Islam, Xiaoyi Lu, Dipti Shankar, and Dhabaleswar K. Panda. MR-Advisor: A comprehensive tuning, profiling, and prediction tool for MapReduce execution frameworks on HPC clusters. *J. Parallel Distributed Comput.*, 120:237–250, 2018.

[77] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. Fast In-Memory Transaction Processing Using RDMA and HTM. In *SOSP*, pages 87–104. ACM, 2015.

[78] Yingjun Wu, Joy Arulraj, Jiexi Lin, Ran Xian, and Andrew Pavlo. An Empirical Evaluation of In-Memory Multi-Version Concurrency Control. *Proc. VLDB Endow.*, 10(7):781–792, 2017.

[79] Juncheng Yang, Yao Yue, and K. V. Rashmi. A Large-scale Analysis of Hundreds of In-Memory Cache Clusters at Twitter. In *OSDI*, pages 191–208. USENIX Association, 2020.

[80] Zhibin Yu, Zhendong Bei, and Xuehai Qian. Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing. In *ASPLOS*, pages 564–577. ACM, 2018.

[81] Irene Zhang, Naveen Kr. Sharma, Adriana Szekeres, Arvind Krishnamurthy, and Dan R. K. Ports. Building Consistent Transactions with Inconsistent Replication. In *SOSP*, pages 263–278. ACM, 2015.

A Testbed

Table 2 records the detailed machine configuration to test each system.

Name	Node Alias	CPU	RAM	Disk	NIC
Calvin	d430	2x Intel E5-2630 v3 8-core CPUs at 2.40 GHz (Haswell w/ EM64T)	64GB	2 x 1TB HDDs	Dual-port Intel 1GbE NIC (i350), Dual-port Intel 10GbE NIC (X710), One Broadcom NetXtreme BCM5720-2P Dual-Port 1GbE NIC
Silo	c6320	2x Intel E5-2683 v3 14-core CPUs at 2.00 GHz (Haswell)	256GB ECC	2x 1 TB 7.2K RPM 3G SATA HDDs	Dual-port Intel 10GbE NIC (X520), Qlogic QLE 7340 40 Gb/s Infiniband HCA (PCIe v3.0, 8 lanes)
HERD	r320	1x Xeon E5-2450 processor (8 cores, 2.1Ghz)	16GB (4 x 2GB RDIMMs, 1.6Ghz)	4 x 500GB 7.2K SATA Drives (RAID5)	1GbE Dual port embedded NIC (Broadcom), 1 x Mellanox MX354A Dual port FDR CX3 adapter w/1 x QSA adapter
MICA	c6220	2 x Xeon E5-2650v2 processors (8 cores each, 2.6Ghz)	64GB (8 x 8GB DDR-3 RDIMMs, 1.86Ghz)	2 x 1TB SATA 3.5" 7.2K rpm hard drives	4 x 1GbE embedded Ethernet Ports (Broadcom), 1 x Intel X520 PCIe Dual port 10Gb Ethernet NIC, 1 x Mellanox FDR CX3 Single port mezz card
DrTM	OSC Owens	2 x Intel E5-2680 v4 (14-core, 2.40 GHz)	128GB	1 x 1TB HDD	100 Gb/s Infiniband EDR
TAPIR	c220g1	2x Intel E5-2630 v3 8-core CPUs at 2.40 GHz (Haswell w/ EM64T)	128GB ECC (8x 16 GB DDR4 1866 MHz dual rank RDIMMs)	2x 1.2 TB 10K RPM 6G SAS SFF HDDs, One Intel DC S3500 480 GB 6G SATA SSDs	Dual-port Intel X520-DA2 10Gb NIC (PCIe v3.0, 8 lanes), Onboard Intel i350 1Gb
Janus	rs630	2 x Xeon E5-2660 v3 processors (10 cores each, 2.6Ghz or more)	256GB (16 x 16GB DDR4 DIMMs)	1 x 900GB 10K SAS Drive	1GbE Quad port embedded NIC (Intel), 1 x Solarflare Dual port SFC9120 10G Ethernet NIC
Cicada	c6320	2x Intel E5-2683 v3 14-core CPUs at 2.00 GHz (Haswell)	256GB ECC	2x 1 TB 7.2K RPM 3G SATA HDDs	Dual-port Intel 10GbE NIC (X520), Qlogic QLE 7340 40 Gb/s Infiniband HCA (PCIe v3.0, 8 lanes)
GAM	c6220	2 x Xeon E5-2650v2 processors (8 cores each, 2.6Ghz)	64GB (8 x 8GB DDR-3 RDIMMs, 1.86Ghz)	2 x 1TB SATA 3.5" 7.2K rpm hard drives	4 x 1GbE embedded Ethernet Ports (Broadcom), 1 x Intel X520 PCIe Dual port 10Gb Ethernet NIC, 1 x Mellanox FDR CX3 Single port mezz card
Star	c6220	2 x Xeon E5-2650v2 processors (8 cores each, 2.6Ghz)	64GB (8 x 8GB DDR-3 RDIMMs, 1.86Ghz)	2 x 1TB SATA 3.5" 7.2K rpm hard drives	4 x 1GbE embedded Ethernet Ports (Broadcom), 1 x Intel X520 PCIe Dual port 10Gb Ethernet NIC, 1 x Mellanox FDR CX3 Single port mezz card
Aria	m510	8-core Intel Xeon D-1548 at 2.0 GHz	64GB ECC (4x 16 GB DDR4-2133 SO-DIMMs)	256 GB NVMe flash storage	Dual-port Mellanox ConnectX-3 10 GB NIC (PCIe v3.0, 8 lanes)
MySQL	m510	8-core Intel Xeon D-1548 at 2.0 GHz	64GB ECC (4x 16 GB DDR4-2133 SO-DIMMs)	256 GB NVMe flash storage	Dual-port Mellanox ConnectX-3 10 GB NIC (PCIe v3.0, 8 lanes)

Table 2. Machine settings used to test different systems.

Received January 2024; revised April 2024; accepted May 2024