[E&A] From Feature Selection to Resource Prediction: An Analysis of Commonly Applied Workflows and Techniques

Ling Zhang
University of Wisconsin-Madison
ling-zhang@cs.wisc.edu

Shaleen Deep
Microsoft GSL
shaleen.deep@microsoft.com

Joyce Cahoon
Microsoft GSL
joyce.cahoon@microsoft.com

Jignesh M. Patel
Carnegie Mellon University
jignesh@cmu.edu

Anja Gruenheid
Microsoft GSL
anja.gruenheid@microsoft.com

Abstract

Understanding and predicting database workload performance on different hardware settings in the cloud is crucial for both the users and providers in order to optimize resource allocation. Recently, machine learning (ML) based techniques have been applied to parts of the end-to-end three-step pipeline for workload prediction: feature selection, workload similarity, and performance prediction. However, despite its practical importance, there exists no principled analysis that studies the performance of such pipelines. In this paper, we examine the state-of-the-art strategies for these three components, with the goal of identifying which techniques work best in practice. Our experimental results reveal that while no universal solution exists for the prediction pipeline, certain best practices can improve prediction performance and reduce computation overhead. Based on our results, we outline important topics for future work that will benefit ML-driven recommendation systems for resource allocation.

1 Introduction

Database workload[1] understanding and prediction in a cloud computing environment is a complex task. Cloud users can spin up hundreds of computing instances at a time, and execute workloads for various tasks from transaction systems over production processes to analytical or ML based optimization systems. Understanding these deployed workloads offers several benefits. From the provider's perspective, being able to reason about and predict a user's workloads enables more efficient task scheduling and resource allocation. From the user's perspective, analyzing their workload allows them to calculate the trade-off between the allocated (and paid for) resources with expected performance implications. Prior work has studied the problem of recommending cloud configurations [18, 50, 101, 102], workload runtime prediction given past observations in a fixed configuration setting [38, 66, 99], and auto-scaling mechanisms [60, 75]. However, to the best of our knowledge, there exists an unmet need to systematically study workload resource (scaling) prediction as part of an *end-to-end* pipeline. More precisely, prior work has only studied parts of the problem (such as query scaling, feature selection, task scheduling, etc.) but has not examined the problem from a holistic point of view. Our work closes this important gap in the literature by examining how different approaches compare, and how they can be utilized in practice effectively.

End-to-end workload prediction pipelines pose a non-trivial challenge that can be decomposed into several parts, as outlined by prior work on workload migration to new or different hardware configurations [12]. Here, the authors focused on modeling a user's workload as a limited set of resource requirements, utilizing features such as CPU and memory consumption but omitting others such as query plan features, which are commonly leveraged for query performance prediction [28, 74, 82, 104]. We observe that in today's cloud databases, extensive telemetry and statistics make *feature selection*, independent of whether the features stem from the queries themselves or the workload's resources, crucial for workload or query prediction [58, 104]. Here, automatic feature selection is preferred over manual feature engineering due to the high-dimensional nature of available features. However, in practice, we observe that practitioners rely on manual feature engineering rather than use automatic feature selection techniques to maximize prediction performance [12, 28]. This is typically due to an insufficient number of training observations that accurately capture the behavior of a workload and/or queries on a given hardware setup. To alleviate this problem, existing algorithms categorize types of workloads using clustering [49, 62, 100] or time-series based approaches [53, 87], focusing on modeling the behavior of workload types rather than individual user workloads. This approach is also referred to as *workload similarity computation*. If these techniques are deployed effectively, *workload prediction* algorithms become more accurate as they can utilize a larger pool of training data based on the workload clusters instead of over-fitting models to a single workload deployment. Moreover, prior work on individual query performance prediction [32, 93, 97, 105] enables us to predict how queries would behave across different hardware configurations. In practice, these query-level prediction models work well for analytical workloads, where the concurrent performance is straightforward to model, but struggle with workloads with transactional queries [76], where the interaction between concurrent workloads are more complex for query-level prediction methods to model. Naturally, a workload-level method (i.e. a technique that considers the entire workload rather than each query in isolation) is more likely to capture the complex interactions and thus, lead to better predictions.

Example 1. *A cloud customer wants to move their workload to a new hardware configuration with more CPUs but they want to maintain their service-level agreements, meaning that prior to moving the workload, they want to check that the queries will execute within a certain timeframe. To calculate the predicted runtime on the new*

[1] Throughout the paper, we will use workload to mean relational database workloads (defined in Section 2).

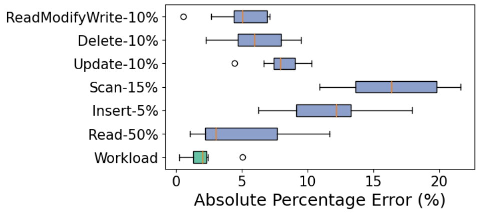


Figure 1: Distribution of absolute percentage error of 10 latency predictions, compared across using the entire workload versus using individual transaction types.

hardware, the cloud provider leverages the performance numbers of other workloads observed on comparable cluster configurations. For this example, we experimentally execute four standardized benchmarks provided by BenchBase [30]: TPC-C, TPC-H, YCSB, and Twitter. Suppose the customer has a workload that consists of a mixture of six different types of transactions from the YCSB workload and wants to predict its scaling behavior. Figure 1 shows the prediction error for query-level scaling prediction (using the prior work on individual query performance prediction to prediction the latency scaling behavior) as well as workload-level scaling prediction (the consolidated latency prediction on the new hardware for the entire user workload). We observe that, despite training the predictors on similar queries and workloads, per-query predictions still result in significant errors, ranging from 4.75% to as high as 16.57%. In contrast, the prediction error using a workload-level scaling technique is only 1.99%.

When comparing the runtime prediction of the workload-level prediction to the aggregated (weighted) query-level , we find that the workload-level approach is 5.6% more accurate. While this may seem modest, cloud compute costs can account for approximately 10% of a mid-sized company's revenue [4], so any reduction in compute costs directly improves profit margins.

Contributions. To the best of our knowledge, this is the first experimental study that analyzes prediction pipelines for database workloads. In this study, we first leverage classical feature selection strategies to understand features that are optimal for workload similarity computation. Second, we show how we can enable workload similarity computation on different types of collected workload-related data (one-off observations vs. temporal data such as resource utilization) and how different algorithms perform in this setting. Finally, we explore how we can model resource scaling behavior and how the choice of modeling context as well as algorithm impacts the prediction performance. With our extensive benchmarking and experimentation, we empirically demonstrate that there is no one-size-fits-all kind of solution to end-to-end prediction pipelines but that there are best practices that we should adhere to in order to avoid certain pitfalls caused by the choice and application of ML algorithms in this space.

2 Problem Description

In this section, we present an overview of end-to-end database workload prediction pipelines, the problems we want to analyze in this paper, and explain the experimental evaluation setup that we use for our analysis. A database workload is a sequence of SQL statements (read-only queries and write queries) over a fixed relational database schema. A transactional workload is a database workload, that executes in real-time, and has a high ratio of write queries (i.e. insert/update/delete) compared to read-only queries. In contrast, analytical workloads are database workloads that exclusively consist of read-only queries. Analytical workload queries can be several orders of magnitude slower compared to transactional workload queries. A mixed workload (also known as hybrid or HTAP workloads) is a mixture of transactional and analytical workloads. It consists of both read-only and write queries, but with a ratio of write to read-only queries that is between transactional and analytical workloads. To understand how we can achieve workload prediction, we first break down the problem into three subproblems, aligned with the following questions: (*i*) *Which workload characteristics identify a workload?*; (*ii*) *How can we compute workload similarity?*; and (*iii*) *How can we predict the performance of workloads on a different set of resources?* These questions correspond to the building blocks of a typical end-to-end pipeline for workload predictions as shown in Figure 2 and are executed as follows:

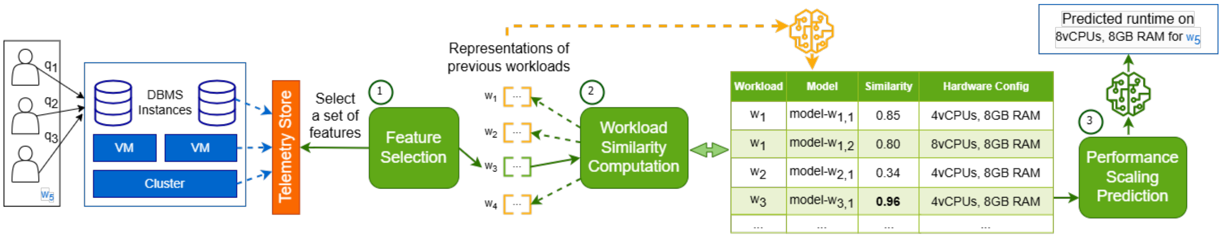
Feature Selection. As input to feature selection, telemetry provides a stream of information about a workload containing data points as well as temporal observations on so-called features such as resource utilization over time, or per-query optimizer statistics such as estimated row counts or row size. In our experimental analysis, we show that there are some features that are important across workloads and some that are specific to a workload. This observation helps us to determine whether (and which) feature selection mechanisms uniquely (and minimally) identify a workload.

Workload Similarity Computation. Workload similarity computation then is the task of determining which workloads have similar feature values and thus similar characteristics. For example, we show that the resource utilization and query patterns observed in transaction-heavy DBMS workloads are different than for analytical workloads. This observation allows us to group similar workloads and use clusters of workloads for downstream prediction tasks alleviating one of the typical pain points in such pipelines, i.e., a lack of training data per workload.

Workload Resource Prediction. As a final step in our pipeline, we look at the problem of predicting resource utilization for a workload, specifically when the workload is executed on a different set of hardware. In a cloud environment, efficient resource provisioning is crucial for customer satisfaction but also for smart load balancing on the provider's side. In our experiments, we evaluate several prediction algorithms and discuss which type of models can and should be deployed in practice. Furthermore, given the in-depth analysis of the other two components presented in this paper, we show that a different feature space as well as adjustments to the prediction techniques can enhance the precision of the algorithms and avoid under-fitting of the prediction algorithms.

2.1 Experimental Setting

To evaluate the pipeline and the methods used in each components, we use BenchBase [30, 40] to run templated standardized benchmarks, monitor their runtime resource usage metrics, log their query plans, and collect performance results.


q1
q2
q3
w5
DBMS Instances
VM
VM
Cluster
Telemetry Store
Select a set of features
1
Feature Selection
Representations of previous workloads
w1
w2
w3
w4
2
Workload Similarity Computation
Predicted runtime on 8vCPUs, 8GB RAM for w5
3
Performance Scaling Prediction

Workload	Model	Similarity	Hardware Config
w_1	model-$w_{1,1}$	0.85	4vCPUs, 8GB RAM
w_1	model-$w_{1,2}$	0.80	8vCPUs, 8GB RAM
w_2	model-$w_{2,1}$	0.34	4vCPUs, 8GB RAM
w_3	model-$w_{3,1}$	**0.96**	4vCPUs, 8GB RAM
...	...	...	...

Figure 2: Interaction between the 1) feature selection, 2) workload similarity, and 3) prediction components.

Table 1: Workload overview for experimental evaluation.

	#Tables	#Columns	#Indexes	Txn Types	%Read-Only Txns	Workload Type
TPC-C	9	92	1	5	8.0%	Transactional
TPC-H	8	61	23	22	100.0%	Analytical
Twitter	5	18	4	5	99.0%	Analytical[2]
YCSB	1	11	0	5	50.0%	Mixed
TPC-DS	24	425	0	99	100.0%	Analytical
PW	-	-	-	500+	Mostly	Mixed

Table 2: Resource utilization and query plans features.

Resource Utilization	Query Plan Statistics	
`CPU_UTILIZATION`	`StatementEstRows`	`CachedPlanSize`
`CPU_EFFECTIVE`	`StatementSubTreeCost`	`AvgRowSize`
`MEM_UTILIZATION`	`CompileCPU`	`CompileMemory`
`IOPS_TOTAL`	`TableCardinality`	`EstimateRows`
`READ_WRITE_RATIO`	`SerialDesiredMemory`	`EstimateIO`
`LOCK_REQ_ABS`	`SerialRequiredMemory`	`CompileTime`
`LOCK_WAIT_ABS`	`MaxCompileMemory`	`GrantedMemory`
	`EstimateRebinds`	`EstimateCPU`
	`EstimateRewinds`	`MaxUsedMemory`
	`EstimatedPagesCached`	`EstimatedRowsRead`
	`EstimatedAvailableDegreeOfParallelism`	
	`EstimatedAvailableMemoryGrant`	

Workload Overview. For our experimental evaluation, we decided to focus on five standardized benchmarks: 1) TPC-C [90] with a scale factor of 100, 2) TPC-H [92] with scale factor of 10, 3)TPC-DS [70, 91]) with a scale factor of 1, 4) Twitter [15, 30] with a scale factor 1600, and 5) YCSB [23] with a skew factor 0.99 and scale factor of 3200 using various hardware settings and concurrency levels. Note that we chose the scale factors of each workload so that the database sizes of all workloads are roughly the same. We run all standardized workloads except TPC-DS with 4, 8, and 32 concurrent terminals. Since TPC-H always runs serially, this workloads runs effectively with 1 terminal for all experiments. The benchmark details of these standardized workloads are summarized in Table 1. Furthermore, we use the execution plans of a production workload (*PW*) that contains data from a decision support system used for querying telemetry data to demonstrate similarity computation of workloads through the use of a real-world example. Note that we only reveal partial information about *PW* due to privacy concerns.

Execution Overview. We execute all of our workloads on a local instance of SQL Server with 2, 4, 8 and, 16 CPUs and collect query plan as well as runtime resource utilization features. An overview of all collected features is given in Table 2. Each experiment runs for one hour and during that time, resource utilization metrics are captured every ten seconds. Each experiment configuration, i.e., hardware configuration and number of terminals where it applies, is executed three times. As a result, we collect raw data of at least 360 observations of system resource utilization features and three query plan observations of each query per workload. For our prediction pipeline, we use systematic sampling to generate ten sub-experiments from one single experiment [14]. To obtain the resource utilization metrics in each experiment, we use the perf utility [1]. The perf command accesses the CPU hardware registers and performance counters for each sample and can take several hundred CPU cycles. Thus to minimize any performance measurement overhead, we set the sampling rate to one sample per second. The query plan statistics are collected by setting the STATISTICS XML flag [2] on in SQL Server, a standard approach that is known to be low overhead for query plan metric collection and used for diagnosing bottlenecks in query execution [16, 39, 84].

3 Related Work

Predicting query performance for different hardware settings can be viewed as part of the workload scaling prediction problem. Yet, most query performance prediction methods are focused on individual queries and are bounded to one fixed database setting which makes predictions unsuitable for transfer to other hardware settings [43]. Prior work predicts individual query performance by creating a performance model that has hardware configuration, query specific features, and concurrent query inference as input [32, 93, 97, 105], but they make strong assumptions on the types of predicted workloads and support usually only analytical ones. As shown in our introduction example, predicting workloads themselves is more accurate instead of averaging across queries. Several works [28, 74, 82, 104] have looked at resource scaling as the database query workload changes dynamically by using specific telemetry. The use of ML based techniques (such as building decision trees over certain features) for resource management [25, 42, 64] is also quite popular. We refer the reader to [68] for an in-depth overview of feature selection algorithms including the methods considered in our work.

Workload performance or resource usage prediction by studying existing workload falls into two genres: single model prediction [22, 77] or an ensemble of models [6, 19, 26, 54]. We focus on single model(s) in our work to compare their effectiveness in our problem setting. Prior work routinely uses resource time-series data to build the models, and selects models based on a cost model [77], peak value [6], or time-series similarity [22]. In our work, the choice of models is made based on the objective of minimizing performance prediction error. Recent work has further explored historical data and how model the workload using CNNs [31], Markov Chain based model [27, 63], LSTM [11, 22, 37, 52, 79, 86], Gated Recurrent

[2] Technically, Twitter is a Mixed workload since 1% of the queries are write. However, for all practical purposes, the read-only queries dominate the workload behavior and thus, we categorize the workload as Analytical.

Unit [41, 73], Autoregressive Integrated Moving Average [13], etc. However, the effectiveness of these techniques, especially LSTM, remains questionable [21]. Therefore, in our experimental study, we intentionally keep the pipeline simple and avoid complicated models for estimation. Finally, database knob tuning using ML methods is also a related well-researched topic [103]. It is related to our pipeline optimization because it uses historical execution data to estimate the workload performance on different hardware configurations to find the best knob setting. This line of work also utilizes a similar pipeline [98] to our work and its drawback is that one needs to train a custom model for each workload.

Beyond ML, [33] describes an intriguing approach to predicting migration costs, duration, and cloud costs of running RDBMS by modeling a database and workload from prior logs to obtain cost and duration estimates. Workload migration to optimize for geographic shifts of load was considered in [81], who propose Supercloud, an architecture for migrating VMs. VM consolidation has also been studied in the context of IoT applications [69]. However, both of the works only consider VM consolidation to minimize network latency and do not consider the impact of co-locating applications that may interfere with each other.

4 Feature Selection

Picking the right features for downstream applications such as workload similarity computation and predictive modeling is crucial to their success. Ideally, a common set of features would be able to clearly identify a variety of workloads, however, we observe that some features are more suited to characterize (specific) workloads than others. To evaluate different features, we use the telemetry collected in our experimental runs, visualized in Table 2. Our evaluation focuses on robust feature selection, as incorrect workload representation has immediate effects on downstream applications.

4.1 Feature Selection Strategies

Feature selection is a well-studied topic and there are many algorithms available for this task [9]. The goal of these algorithms is to ensure that those features most relevant to a given task are selected in a way that provides good predictive performance. In our setting, the given task is to maximize our accuracy for workload similarity computation. To quantify the effectiveness of the respective feature selection algorithms, we base our similarity computation on the selected feature set and compare it with the ground truth of our experiments. That is, a TPC-DS workload characterized with a subset of features should be identified as being similar to TPC-DS and not TPC-H, TPC-C etc. In this work, we focus on feature selection techniques that give a rank or importance score for each feature. The feature selection methods we test can be classified into three different categories: Filter, embedded, and wrapper approaches.

4.1.1 Filter Approach. Feature selection methods in this category evaluate the importance of predictors prior to fitting the model. Predictor relevance is determined separate from the model with the help of various variable importance metrics. As most of these metrics are calculated on a univariate basis, it is possible that we end up selecting too many predictors or correlated predictors. However, filter approaches are simple and fast, so we focus on a subset of four commonly used statistical techniques to compare against more complex feature selection methods.

Variance Threshold. This method examines the variance of each predictor to measure its informative value for model inclusion. It removes any predictor with zero variance, and keeps predictors whose variance exceeds a specific threshold.

Pearson Correlation Coefficient [71]. This coefficient is used to measure the linear dependency of a predictor with the target variable. It is calculated by normalizing covariance between variables with each variables' respective standard deviation. We use the absolute values of the correlation coefficients as a means of weighing the importance of each predictor.

Functional Analysis of Variance (fANOVA) [48]. This metric measures the importance of each feature based on its contribution to the target variable variance. Features that contribute significantly to the variance are considered important and are selected.

Mutual Information Gain [8]. The mutual information gain evaluates the dependency between a feature and the target label. It measures the reduction in uncertainty (via the difference between entropy of the feature and conditional entropy given the target) about the target when the feature is known. A value of zero indicates independence. These values serve as feature importance scores.

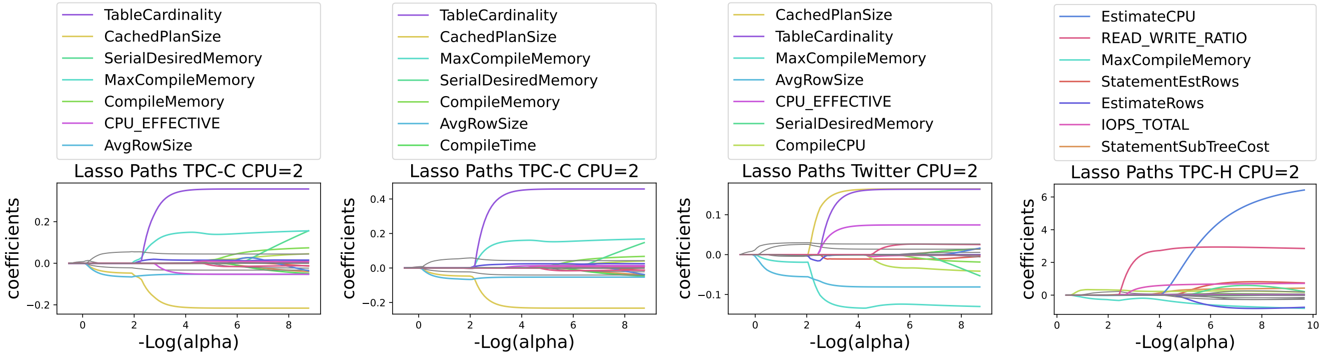
4.1.2 Embedded Approach. Methods in this category consist of models that contain built-in feature selection. In other words, the model itself will include specific predictors that maximize accuracy such that the process of feature selection is embedded in model training. Given that there often exists challenges around collinearity in our feature set, we examine a subset of regularized linear models and ensemble learning methods to control model variance:

Lasso [89]. This method adds a penalty term to the standard sum of squares objective used in ordinary linear regression to control (or regularize) parameter estimates in cases where multi-collinearity might exist in the data. Lasso is capable of using regularization to not only identify a better model but also conduct feature selection.

Elastic Net [106]. While Lasso provides built-in feature selection by zeroing out highly correlated predictors, it is indifferent in its selection, i.e., it can pick an irrelevant predictor from a set of highly correlated ones, resulting in a model that overlooks the true predictor. A popular approach that resolves this challenge is elastic net, which combines two different penalty terms, the L1 regularization from lasso as well as L2 regularization from ridge.

Random Forest [10]. This ensemble learning method aggregates the results of multiple decision trees. Each tree is built with a random subset of data and its features, ensuring diversity among trees. As random forests are attained as the linear combination of many independent learners, e.g., often 1000+ of decision trees, it achieves variance reduction by selecting complex learners that exhibit low bias. The importance of each feature is then evaluated based on its contribution to impurity reduction for all trees in the forest.

4.1.3 Wrapper Approach. Finally, methods in the wrapper approach category optimize model performance by iteratively adding or removing predictors, aiming to find the optimal subset of features.



(a) TPC-C Run 1 **(b) TPC-C Run 2** **(c) Twitter** **(d) TPC-H**

Figure 3: Lasso path of features for each experiment using hardware setting of 2 CPUs as the strength of regularization (alpha) decrease. The y-axis shows the resultant coefficient of each feature. Labels show the top-7 (ranked from most important to least important) with the highest absolute non-zero coefficient.

Wrapper methods involve multiple rounds of training across various models to identify a satisfying feature subset, often resulting in good subsets at the cost of increased computational complexity.

Recursive Feature Elimination (RFE) [20]. This method recursively removes features that have the least feature importance, until an optimal feature subset is attained based on some model objective. This backward selection algorithm can work with many different machine learning strategies as long as they provide some means of calculating variable importance.

Sequential Feature Selection (SFS) [65, 95]. This method iteratively adds or removes one or more features in a greedy manner based on the prediction performance. Compared to RFE, SFS can execute both forward and backward selection and can add (or remove) features based on some user-defined performance metric. SFS does not require the underlying model to output feature importance scores and thus may result in a more balanced set of predictors based on the scoring metric selected.

4.2 Evaluating Feature Importance

As mentioned previously, we focus our experimentation on feature selection strategies that can score or rank features. In general, we observe that the output of the strategies introduced in the previous section can be split into two categories: *Score*-based and *Rank*-based feature selection strategies.

Score-based feature selection. Statistics-based selection strategies, such as filtering predictors using variance threshold or MI gain, and regression-based methods, such as Lasso and elastic net, fall into this category. They calculate a continuous score for each feature, and which feature is more important can be determined by comparing the score between features.

Rank-based feature selection. Wrapper-based feature selection methods like RFE and SFS utilize an estimator to gradually add or remove features. The implementations we used here effectively assign an integer rank to each feature.

In our evaluation, we generate a feature importance ranking per experiment and per feature selection strategy. The output of score-based feature selection strategies is transformed to ranks by ordering the features according to their scores. For top-k feature selection, we aggregate the ranks across experiments and select the top-k features with the lowest aggregate rank.

4.3 Experiments

To evaluate the feature subsets selected by the feature selection strategies, we first create histograms of the feature's value distribution as follows: We normalize the value space of each feature to $[0, 1]$ by using the respective minimum and maximum value and evenly split the feature value range into ten bins. Note that we will discuss further details of choosing a data representation and the implications of such a choice for workload similarity computation in Section 5.1.1. As mentioned previously, we measure the success of feature selection strategies by using similarity computation and computing the 1-NN distance which in essence determines the accuracy of a strategy. For the following set of experiments, we leverage the $L_{2,1}$-norm, see Section 5.1.2 for details on this norm.

4.3.1 Picking Features. Given all available features shown in Table 2, we first observe that there are some features that consistently rank well across methods, for example, the average returned row size (`AvgRowSize`) and the cached plan size (`CachedPlanSize`) have high feature importance scores for the majority of selection strategies on multiple hardware settings. However, features like the estimated degree of parallelism, rebinds, rewinds are usually considered unimportant independent of the selection mechanism.

Diving deeper into results of the different selection strategies, we visualize the results of Lasso, a well-performing embedded technique in Figure 3. Here, we plot the top-7 features selected by Lasso for a workload on the same hardware setting. The larger the deviation from 0, the higher the feature importance for the given workload. We make two observations based on this set of experiments. First, Figure 3a and Figure 3b show the execution of the same workload, TPC-C, on the same hardware for two experimental runs. Although there is an overlap in the respective feature

Table 3: Comparison of Feature Selection Strategies (Accuracy & Elapsed Time).

	# Features					
Strategy	top-1	top-3	top-7	top-15	all	**Time (sec)**
Variance	0.483	0.717	**0.997**	**0.997**	0.994	0.025
fANOVA	0.969	0.983	0.986	0.989		0.052
MIGain	0.976	0.972	0.986	0.986		2.538
Pearson	0.969	0.983	0.986	0.989		0.031
Lasso	0.467	0.969	0.989	0.989		0.051
Elastic Net	0.467	0.969	0.992	0.989		0.095
RandomForest	0.981	0.972	0.989	0.989		9.923
RFE Linear	0.969	0.972	0.969	0.989		1.128
RFE DecTree	0.247	0.953	**0.997**	**0.997**		1.202
RFE LogReg	0.969	0.969	0.989	**0.997**		18.936
Fw SFS Linear	0.725	0.969	0.969	0.972		580.069
Fw SFS DecTree	0.725	0.969	0.969	0.972		722.072
Fw SFS LogReg	0.969	0.972	0.972	0.972		1829.737
Bw SFS Linear	0.247	0.953	**0.997**	**0.997**		2793.939
Bw SFS DecTree	0.969	0.953	**0.997**	**0.997**		3978.708
Bw SFS LogReg	0.969	0.978	0.992	**0.997**		11383.510
Baseline	0.233	0.483	0.975	0.972		0.003

spaces, there are also differences. For example, `CPU_EFFECTIVE` is only considered an important feature in Figure 3a. There are numerous factors in the cloud that affect collected metrics and thus, the downstream feature selection results, but generally, the more often we run feature selection for the same workload, the more stable our selected features become. Our second observation is that conceptually similar workloads tend to have a similar set of important features. Take the features selected by TPC-C Figure 3a and Twitter Figure 3c: both workloads share common important features such as the average returned row size (`AvgRowSize`), table cardinality (`TableCardinality`), and the cached plan sizes (`CachedPlanSize`), for a total of six overlapping features. This is expected since both workloads contain point lookup queries[3]. In contrast, both workloads overlap with only one feature of TPC-H as shown in Figure 3d. Further, while both Twitter and TPC-H are analytical workloads, `READ_WRITE_RATIO` and `IOPS_TOTAL` are important features for TPC-H since the analytical queries are memory intensive and may require large intermediate results that get flushed to disk in case of a buffer overflow. In contrast, the Twitter workload queries are point lookup queries (such as get the tweet for a given tweet id, or get any 20 tweets for a given user), and thus do not require any intermediate result materialization. Thus, I/O related features are not as relevant for this workload. YCSB workloads are significantly more I/O intensive than TPC-C and thus, features such as `EstimateIO` and `EstimatedAvailableMemoryGrant` have increased importance. However, owing to its similarity to TPC-H as both workloads contain write operations, `CPU_EFFECTIVE`, `TableCardinality`, and `SerialDesiredMemory` are also present in the top-7 most important features.

Insight 1. *Which features uniquely identify a workload can be tied to the type of that workload. Workloads with the same characteristics often share an overlap in their representative feature space.*

4.3.2 Strategy Evaluation. In addition to determining the impact of selecting different feature sets for workload representation, we want to examine whether a subset of features is sufficient to achieve the same (or better) accuracy compared to using all available features. Table 3 shows an overview of the accuracy of the various strategies when choosing the top-k features with $k \in \{1, 3, 7, 15\}$ and a hardware configuration of 16 CPUs. Again, we compute the accuracy as the correct 1-NN clustering of workloads using the top-k features as input for the workload similarity algorithm.

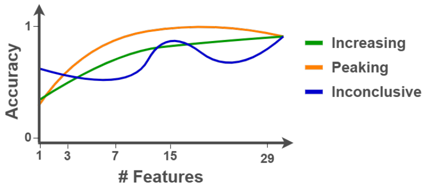


Figure 4: Generalized Accuracy Development Curves.

Insight 2. *We experimentally observe three different types of dependencies between the number of features and accuracy: Accuracy* **increases** *with an increased number of features; accuracy* **peaks** *with a specific number of features; and the number of features has* **inconclusive** *impact on the measured accuracy.*

An overview of the observed behavioral patterns is shown in Figure 4 aligned with the measured strategies in Table 3, where strategy name colors correspond to the patterns. Looking at these numbers in detail, we observe underfitting if only a few features are used across all approaches. RFE with decision tree and Backward SFS with linear regression achieve a low accuracy of 0.247 when selecting only one feature. Both methods select `LOCK_WAIT_ABS` which is not an important feature (according to Lasso) for any of the workloads in the experiment. However, wrapper-based methods tend to select this feature as it has very high variance. High variance in a feature means it has a wider range of values, which can potentially provide more information for the model to learn from, leading to a larger impact on the model's prediction. Other methods that achieves high accuracy with only one feature (e.g. fANOVA, Mutual Information Gain) select either `AvgRowSize` or `TableCardinality`, which appear in the top-7 list of important features for all workloads except TPC-H. When selecting top-3 features, the two under-performing methods additionally select important features (both methods pick `READ_WRITE_RATIO`) which leads to a significant improvement in the accuracy. As the number of features increases, we observe a performance improvement except for the two inconclusive strategies. Interestingly, for some approaches overfitting may occur when all features are used, leading to (slightly) suboptimal accuracy. Here, some strategies show peaking behavior indicating its ability in identifying a relevant subset of features, e.g., 7 or 15 features, reducing the overall number of required features to 24%, resp. 52%, of the number of input features. On average, we observe that with top-15 features chosen, the same average accuracy can be reached across all strategies as if all features are used for clustering, though the actual accuracy varies based on the respective strategy. Furthermore, we want to highlight that strategies such as SFS have near-perfect accuracy. The primary drawback of these strategies is their relatively high execution time, which is two to three orders of magnitude higher than that of simple filter-based methods which can obtain the same level of accuracy with as low as 7 features.

[3] Point lookup queries are read-only queries that access a small number of rows.

4.4 Key Takeaways

In our experimental evaluation, we considered 15 feature selection strategies which were used on 29 resource utilization and plan features. We observe that the importance of a feature is dependent on two factors, the chosen feature selection strategy and the workload that it is applied on. However, we also observe a correlation between the type of workload and the feature importance ordering; for example, memory-intensive workloads tend to prioritize I/O related features compared to workloads containing point lookup queries for which features related to table cardinality, cached plan sizes, and compilation memory are the most important. Mixed workloads (such as YCSB) prioritize both types of features (i.e., I/O as well as query plan related features). This takeaway is further validated by looking at the top-4 features of the production workload (which is also mixed) being `CPU_EFFECTIVE`, `TableCardinality`, `StatementEstRows`, and `EstimateIO`, which are all present in the top-7 features for YCSB as well. We also observe that for most feature selection strategies, there exists a trade-off between choosing too few and too many features, impacting downstream accuracy for workload similarity computation. In essence, too few features fail to capture the characteristics of a workload run while too many features may lead to overfitting.

5 Workload Similarity

The second question that we examine in this paper is how we can compute workload similarity for which we use the output of top-k feature selection as the input. More specifically, we can use a common set of features to compute the similarity between workloads where the difference in feature values or their distribution signifies the dissimilarity of the workloads. In this section, we first introduce state-of-the-art workload similarity computation mechanisms and then experimentally showcase their advantages and disadvantages.

5.1 Overview of Similarity Computation

Workload similarity computation is the problem of aligning two workloads such that their feature spaces become comparable, allowing us to measure the distance between them. The challenges of computing workload similarity are (*i*) determining a suitable feature representation for similarity computation which we refer to as the *data representation* problem, and (*ii*) finding the similarity algorithm that is most suited to a given feature space, in the following referred to as *similarity computation* problem.

5.1.1 Data Representation. During the execution of our workloads, we use DBMS built-in tools to extract telemetry for both query plan and resource utilization statistics. The former contains features characterizing details of a specific query only while the latter provides a time-series about the state of the workload's resource utilization. More generally, these two types of feature sets present two types of feature spaces: Those that are *discrete* and those that are *continuous.* Continuous feature spaces can be represented naturally as *multivariate time-series* (MTS), and thus, time-series based distance measures are a good candidate to measure their similarity as described below. Furthermore, describing a feature's value distribution makes comparison between workloads possible even if the type of observation differs. Equi-range frequency histograms [29] and techniques like cumulative frequency distribution [47] over set bins is commonly used to encode distribution information. We will refer to this technique as *Histogram-Based Fingerprinting* (Hist-FP) in the following. Another strategy is to encode the telemetry values by their statistics like mean and variance. For example, prior work [46] introduces *Phase-Level Statistical Fingerprinting* (Phase-FP) that deploys change-point detection algorithms, such as Bayesian change point detection (BCPD) [61], to pinpoint significant shifts in the statistical properties of the workload. We include the detailed data representation examples in the full version of the paperAppendix A.

5.1.2 Similarity Computation. Similarity computation is the task of capturing the difference between two workloads in a single numeric score. Depending on the data representation, we differentiate between two types of similarity computation strategies. First, *norm-based* distances compute the distance of values or their distributions. Second, *MTS-based* similarity computation relies on the alignment of time-series data to determine the similarity between workloads.

Norm-Based Similarity. For data pre-processed to matrices of same sizes, we can compute the distance of a pair of matrices with the matrix norm. Thus, the norm-based similarity measures calculate the mathematical space-distance of matrices independent of whether the data is stored in a MTS or via distribution fingerprints. In our experiments, we deploy widely utilized distance measurement norms such as $L_{1,1}$, $L_{2,1}$, Frobenius, Canberra, Chi-Square (Chi2), and the Correlation norm [57]. The advantage of these norms is their linear time computation. Furthermore, they are a natural fit for Hist-FP and Phase-FP matrices.

MTS Distance Measure. In addition to norm-based measurements, using MTS-based distance measures allows us to utilize the ordered nature of time-series. For example, Dynamic Time Warping (DTW) [78] matches time-series with shapes that are partially stretched or compressed. Specifically, given two time-series $\boldsymbol{a} = (a_1, a_2, \cdots, a_m)$ and $\boldsymbol{b} = (b_1, b_2, \cdots, b_n)$, we construct a matrix $\boldsymbol{M}$ and $\boldsymbol{M}_{i,j} = (a_i - b_j)^2$. We can represent a time-series alignment by a path traversal from $\boldsymbol{M}_{1,1}$ to $\boldsymbol{M}_{m,n}$ where each step can move to the entry below, to the right, or to the bottom right. A path along the diagonal of $\boldsymbol{M}$ is the Euclidean distance of $\boldsymbol{a}$ and $\boldsymbol{b}$. Different from the one-to-one distance comparison in norm-based similarity measurements, DTW uses one-to-many comparisons which is more robust. The original uni-variate DTW algorithm can be further generalized into *independent* and *dependent* strategies [83]. *Independent* multivariate-DTW sum up the individual DTW distance for each dimension, allowing for more flexibility for uncorrelated dimensions, whereas the *dependent* strategy uses squared Euclidean distance for constructing the matrix $\boldsymbol{M}_{i,j} = \sum_{k \in K}(A_{ik} - B_{jk})^2$ where A and B are both multivariate time-series with K features.

Another commonly used time-series distance measure is Longest Common Sub-Sequence (LCSS) [45] that computes the edit distance between multiple time-series'. More specifically, LCSS measures the length of the most similar sub parts of the two time-series, making it suitable for time-series with different lengths. Similarly to the differentiate of dependent and independent DTW, dependent LCSS aligns and compares all dimensions of a multivariate time-series together, finding the longest common subsequence across all dimensions while independent LCSS aligns each dimension of

Table 4: Similarity computation mechanisms comparison using mean normalized distances as source of similarity confidence. – denotes when a mechanism does not achieve a perfect prediction result.

(a) MTS representation

		Resource		
		3	5	all
$L_{2,1}$-Norm	mAP	1	0.978	0.978
	NDCG	0.993	0.993	0.993
$L_{1,1}$-Norm	mAP	0.986	0.978	0.992
	NDCG	0.993	0.993	0.993
Fro-Norm	mAP	0.986	1	1
	NDCG	0.993	0.993	0.993
Canb-Norm	mAP	1	1	1
	NDCG	1	1	1
Dependent-DTW	mAP	0.978	0.986	0.986
	NDCG	0.993	0.993	0.993
Independent-DTW	mAP	0.981	0.970	0.963
	NDCG	0.993	0.993	0.993
Independent-LCSS	mAP	0.896	0.927	0.931
	NDCG	0.993	0.986	0.986

(b) Hist-FP representation

		Plan			Resource			Combined		
		3	7	all	3	5	all	3	7	all
$L_{2,1}$	mAP	1	1	1	1	1	1	1	1	1
	NDCG	1	1	1	0.993	0.993	0.993	1	1	1
$L_{1,1}$	mAP	1	1	1	0.970	1	1	1	1	1
	NDCG	1	1	1	0.993	0.993	0.993	1	1	1
Fro	mAP	1	1	1	1	1	1	1	1	1
	NDCG	1	1	1	0.993	0.993	0.993	1	1	1
Canb	mAP	1	1	1	1	0.991	0.991	1	1	1
	NDCG	1	1	1	0.993	0.988	0.988	1	1	1

(c) Phase-FP representation

		Plan			Resource			Combined		
		3	7	all	3	5	all	3	7	all
$L_{2,1}$	mAP	1	1	1	0.862	-	1	-	1	1
	NDCG	0.978	0.989	1	1	-	0.970	-	0.956	0.956
$L_{1,1}$	mAP	1	1	1	1	1	1	-	1	1
	NDCG	0.977	0.988	1	0.998	0.973	0.978	-	0.956	0.956
Fro	mAP	1	1	1	-	-	-	-	1	1
	NDCG	0.978	0.990	1	-	-	-	-	0.961	0.956

the multivariate time-series independently, identifying the longest common subsequence for each dimension separately.

5.2 Experiments

In our experiments, we first normalize and transform the data of our benchmarks to fit the MTS, Hist-FP, and Phase-FP. For Hist-FP, we summarize uni-variate time-series data of each feature by evenly splitting the feature value range into n sub-ranges $B_1, B_2, \cdots, B_n$ where each feature value is assigned to a bucket B_i. In our experiments, we set $n = 10$ as default. For Phase-FP, we use the mean, median, and variance to capture the distribution of each phase.

As norm-based similarity techniques, we deploy the $L_{1,1}$, $L_{2,1}$, Frobenius, Canberra, Chi2, and the Correlation norm in conjunction with Hist-FP, Phase-FP and MTS. For MTS, we additionally calculate the dependent and independent versions of DTW and LCSS. Determining which data representation and similarity computation mechanism is most effective is non-trivial. In our evaluation, we focus on three dimensions: (*i*) reliability, (*ii*) discrimination power, and (*iii*) robustness.

Table 5: Top-7 features selected by RFE LogReg for set of only plan query statistics features and set of all features. Top-5 features selected for set of only resource utilization features. The features are arranged in descending feature importance from top to down in each column. Top-3 features for each type are colored.

Top-7 Plan	`MaxCompileMemory`, `CachedPlanSize`, `AvgRowSize`, `EstimateIO`, `StatementSubTreeCost`, `SerialRequiredMemory`, `CompileMemory`
Top-5 Resource	`LOCK_WAIT_ABS`, `MEM_UTILIZATION`, `LOCK_REQ_ABS`, `CPU_UTILIZATION`, `CPU_EFFECTIVE`
Top-7 All	`LOCK_WAIT_ABS`, `MaxCompileMemory`, `AvgRowSize`, `CachedPlanSize`, `EstimateIO`, `MEM_UTILIZATION`, `LOCK_REQ_ABS`

Reliability. This dimension describes whether a method correctly identifies the workloads that exhibit the highest degree of similarity. Comparative analysis against ground truth or expert judgments, i.e. correctly identifying a (non-)match which in essence is the same as finding the most closely related workload run (1-NN). Additionally, we use the mean Average Precision (mAP) to compare the confidence of similarity results.

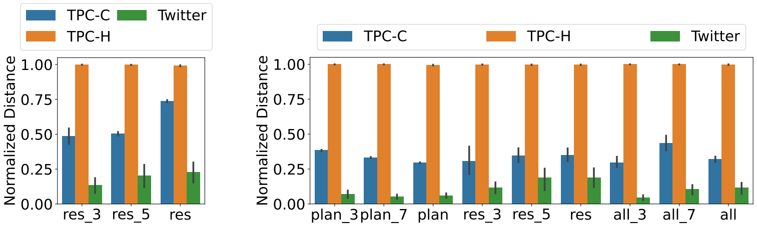
Discrimination Power. An effective approach should differentiate between similar and dissimilar workloads accurately. It should assign higher similarity scores to workloads with comparable behavior and lower scores to those with distinct patterns. Furthermore, it should be sensitive to subtle differences while being robust to noise and variations within the data. After normalizing the distance generated by each methods, we compare the differences between the most similar workloads to the least similar workloads as identified by expert judgement. To quantify the difference, we use the Normalized Discounted Cumulative Gain (NDCG) [51], a metric commonly used to evaluate ranking systems by measuring the quality of the ranking based on relevance. In our case, NDCG rewards shorter distances for workloads that are more similar.

Robustness. This final dimension describes an approach's resilience to noise, outliers, and missing data. In real-world use cases, we often observe measurement irregularities, thus, robust similarity computation ensures reliable results, even in the presence of data imperfections.

5.2.1 Comparison of Similarity Methods. In our first set of experiments, we compare TPC-C, TPC-H, and Twitter on a 16-CPU setup and show the results of those similarity computation methods that achieve perfect 1-NN prediction in Table 4. To determine the top-k features, we use RFE with Logistic Regression as feature selection technique and show the resulting feature sets in Table 5.

Reliability. Looking at the mAP results, we see that no similarity computation dominates the others. However, we generally observe that the $L_{1,1}$, $L_{2,1}$, Frobenius norms work well if combined with Hist-FP and Phase-FP across all feature set combinations. Furthermore, the Canberra and Frobenius norms also work well with MTS and Hist-FP respectively.

Discrimination Power. Looking at the NDCG results in Table 4, we observe that Hist-FP combined with the $L_{2,1}$, $L_{1,1}$, Frobenius, and Canberra norms as well as MTS using the Canberra-norm results in high NDCG scores. This indicates their high discrimination power within this setup. Although some norms on MTS have a relatively



(a) Canberra-norm on MTS.

(b) $L_{2,1}$-norm on Hist-FP.

Figure 5: Similarity results of the Twitter workload.

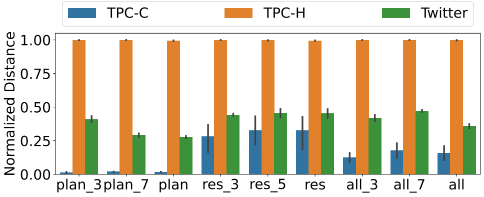


Figure 6: Similarity results of the TPC-C workload of $L_{2,1}$-norm on Hist-FP.

good NDCG score of 0.993, most norms do not perform as well if combined with MTS and Phase-FP. For example, they fail to identify that the TPC-C workload is a transactional workload similar to Twitter, assigning a high distance score instead.

Robustness. Finally, we can visualize the robustness of the different techniques through the error variation of each workload since we execute each of them multiple times and thus has (potentially) different values within a feature space. An example of such a visualization is shown in Figure 6. In essence, the smaller the error bars, the more robust the approach. We observe that in this set of experiments, the error for MTS-based approaches is slightly higher on average. As we will discuss later, this is likely related to the use of resource utilization features only.

Insight 3. *In our experiments, Hist-FP with $L_{1,1}$, $L_{2,1}$, Frobenius, and Canberra norms have similarity computation results that satisfy the reliability, discrimination power, and robustness criteria. Methods based on Phase-FP, DTW, LCSS, Correlation, and Chi2 norm give less satisfactory results across all criteria although they may have good results across a subset.*

5.2.2 *Importance of Feature Selection.*

Next, we investigate how the choice of feature space impacts similarity computation results. We focus on the following feature sets: plan-only features, resource-only features, and a combination of thereof. We choose scenarios with only one type of features (plan or resource) as they are collected using different rationales and requires different implementation in code, and therefore might not be simultaneously available in practice. Additionally, we evaluate the impact of different feature subset sizes, including top-3 and top-7 subsets for plan-only and combined features. Due to the limited number of resource-only features, we consider the top-3 and top-5 subsets (instead of top-7) for this feature set category.

Reliability. Referring again to Table 4, we observe that Hist-FP and Phase-FP tends to achieve perfect mAP with both plan and combined features. Furthermore, two similarity computation techniques ($L_{2,1}$-norm and Frobenius-norm) on Hist-FP and one ($L_{1,1}$-norm) on Phase-FP produce reliable results consistently across majority of the feature subsets. Other similarity computation techniques on Hist-FP and Phase-FP achieve perfect 1-NN prediction accuracy using resource features alone, while getting lower mAPs. MTS uses resource features only and achieves good reliability for the Canberra-norm on MTS, top-3 using $L_{2,1}$-norm on MTS, top-5 and all features for Frobenius-norm. In general, our results show that most fingerprint representations perform better when using plan or combined features compared to resource-only features. We also observe that all top-3 feature sets tend to be less accurate, signified by a low mAP score, as their feature space is not diverse enough to allow for effective similarity computation. This aligns with our insights stated in Section 4.3.2, where we observed underfitting with too few features chosen.

Discrimination Power. For most feature set combinations, $L_{1,1}$, $L_{2,1}$, Canberra, and Frobenius norms combined with Hist-FP correctly identify the identical, similar, and different workloads, indicating high discrimination power. However, using all available features leads to a smaller normalized distance difference between the identical as well as similar workloads, as shown in Figure 5b and Figure 6. In contrast, using the top-7 features more distinctively differentiates the workloads. This aligns with our insights stated in Section 4.3.2, where we observed that the performance may start to decrease if too many features are chosen due to overfitting.

Robustness. Using Figure 5 and Figure 6 for reference, we observe that experiments using resource-only utilization features exhibit a higher standard error compared to other feature (sub)sets. The bin values for TPC-C and Twitter are more skewed, and more spread for TPC-H.

Insight 4. *Using plan-only or a combined feature set improves workload similarity computation across data representation and similarity computation methods. Additionally, the number of features should neither be too small (not enough information) nor too big (too much information).*

5.2.3 *Similarity Computation using a Production Workload.*

Next, we applied our findings to determine whether similarity computation can be used to describe the characteristics of an unknown workload. For that purpose, we compare our production workload *PW* to four reference workloads: TPC-C, TPC-H, TPC-DS, and Twitter. In this set of experiments, we use a different hardware setup consisting of 80 virtual cores. We visualize the results using Hist-FP applied in combination with the Canberra norm in Figure 7. Note that we only have access to plan features for this experiment due to missing resource tracking on the setup instance. Looking at the resulting numbers, we conclude that *PW* seems to be closer related to the TPC-H workload rather than TPC-C, TPC-DS, or Twitter. In fact, we manually confirmed that the queries in *PW* are most commonly simple analytical queries which align better with TPC-H. We furthermore note for this workload that, again, using top-3 or all available features is not as accurate as using the top-7 features, substantiating our findings from our prior synthetic workload experiments.

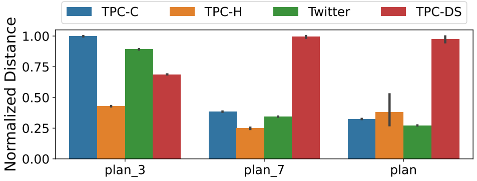


Figure 7: Similarity results of *PW* compared to standardized workloads on different hardware settings.

5.3 Key Takeaways

In our experimental evaluation, we have discussed two key parts of workload similarity computation: The raw data encoding and the similarity computation algorithm. We have furthermore introduced an evaluation scheme for the effectiveness of similarity computation algorithms along three dimensions, i.e., reliability, discrimination power, and robustness. In our experiments with different benchmarks as example workloads, the Hist-FP encoding has shown to be the most promising in providing accurate similarity computation results. Additionally, using this data representation incurs little computational overhead and low storage requirements. We also observe experimentally that plan-based features oftentimes seem to lead to better similarity computation results, substantiating the need for effective feature selection strategies.

6 Workload Resource Prediction

The final question we want to explore in this paper is whether we can predict the resource utilization of a workload on a new set of hardware configurations. Recall that *resource prediction* is the problem of predicting the performance of the same workload on a different system setup, leveraging information collected from similar workloads that are similarly scaled. In this section, we will split the problem into two parts: (*i*) defining the *context* of the prediction behavior and (*ii*) defining the *strategies* that help to model the it. The former answers the question how we should think about the big picture of modeling resource prediction, i.e., whether we can assume linear performance improvements with proportionate hardware requirement changes, while the latter then helps us to determine how to use well-established ML techniques to appropriately represent resource scaling behavior.

6.1 Overview of Modeling Techniques

In cloud settings, providers commonly have the ability to provision more than one type of resources. For example, Microsoft Azure allows users to select how many cores are provisioned for their VM or how much memory is allocated, where each of the hardware configurations is referred to as a stock keeping unit (SKU). For both the user and the provider, one important question in terms of resource utilization is to identify the best trade-off between workload performance and cost. In other words, if we can model the performance per SKU, we can calculate the optimal per-user resource allocation, benefiting both the provider (fewer resources wasted) and the user (less cost).

6.1.1 Modeling Context. The first question that we need to examine is how we can define the context of a resource scaling model. We observe two different approaches:

Single Scaling Model Approach. In this approach, we develop a comprehensive model capturing the relationship between different SKUs and a specific workload, i.e., showcasing how the workload develops as a progression over different hardware settings.

Pairwise Scaling Model Approach. Instead of developing a holistic model, we focus on pairwise scaling factors that describe the relationship between the performance of pairs of SKUs. The relationship of a pair of data points is linear, simplifying the required modeling strategies. The scaling factor then represents the change in performance if one were to move from one SKU to another.

6.1.2 Modeling Strategies. Next, we describe several ML modeling strategies that are commonly used for predicting workload behavior. Note that for this specific problem, we do not want to predict the behavior of a workload itself but of a workload on a different hardware configuration. Thus, we model the array of available SKUs as a vector of size s, where s is the number of available hardware configurations. We then measure the performance of a workload, y, w.r.t. a SKU by using traditional performance metrics such as latency or throughput.

Linear Models. This type of models are used when we presume linearity of the relationship between an independent variable, the SKU, and a response variable, the performance metric in this case.

Regression [56]. Regression models come in a variety of types, such as linear, polynomial, and lasso, to accommodate different relationship between a dependent variable and independent variables.

Linear Mixed Effect Model (LMM) [7]. This is a statistical technique that extends the traditional linear model to accommodate data that is grouped or clustered. LMM allows the model to have group-specific intercepts and slopes, handling both fixed effects, the standard independent variable, and random effects, which accounts for the variation among clusters.

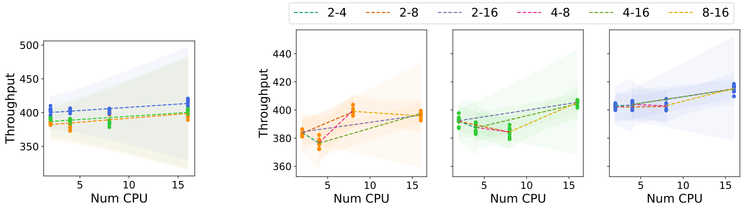
Non-Linear Models. Non-linear models are commonly used if there is no continuous arc across the data.

Multivariate Adaptive Regression Splines (MARS) [34]. MARS partitions the predictor space into regions and fits linear regressions within each one, providing a piece-wise linear fit to the data.

Support Vector Machine (SVM) [24, 59, 85]. SVM regression, with its basic version modeling a linear relationship, is similar to regression, but more effective in modeling non-linear relationships between various features and our target performance metric with the help of different non-linear kernels.

Gradient Boosting (GB) [35, 36]. Gradient Boosting builds a predictive model in a stage-wise fashion, using an ensemble of weak prediction models, typically decision trees, to create a strong overall predictor. It gives more weight to the data points that were incorrectly predicted in the previous iteration.

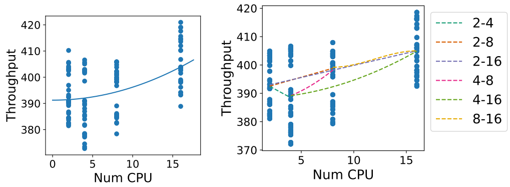
Neural Networks (NNet) [44, 55]. Neural networks model the outcome by intermediary set(s) of unobserved variables, called hidden units or hidden variables, which are linear combinations of the predictors transformed by sigmoidal functions. These non-linear functions enable complex pattern matching between inputs and outputs. We used Multi-Layer Perceptron regressor with 6 layers from Scikit-Learn [72] in our experiments.



(a) Single model. **Each color encodes a different data group.**

(b) Pairwise models. **Each plot shows the models built for a data group. Different colored lines within a plot shows models for pairs of #CPUs (shown in the legend).**

Figure 8: Comparison of scaling model approaches using LMM as the modeling strategy and TPC-C as the observed workload on varying hardware configurations. A model or a set of models is built for each of the three data groups. The shaded region is the Confidence Interval of model prediction.



(a) Single model.

(b) Each color represents a model for a pair of #CPU.

Figure 9: Comparison of scaling model approaches using SVM as the modeling strategy and TPC-C as the observed workload on varying hardware configurations.

6.2 Experiments

To predict the performance of different workloads, we use the same experimental setup as previously, deploying a variety of standardized benchmarks on four SKUs with different hardware characteristics. To simplify the setup, we do not vary all aspects of the hardware configuration but instead focus on the number of available CPUs and the throughput of a workload as the target variable. We pick three times across the day, and each workload and SKU combination is run at those three times to collect the metrics, allowing us to capture potential variations in different runs. The metrics obtained from workloads that were executed at the same time of the day constitute a *data group*. Furthermore, as a data augmentation strategy, we use random sampling without replacement to down-sample a timeseries to ten smaller-sized series, and since we run each experiment setup three times, the result is a total of 30 data points for each workload setting. In the following step, we do a 5-fold cross validation for each of the reported models. Finally, we calculate the normalized root mean square error (NRMSE) [80].

6.2.1 Single vs Pairwise Modeling. In our first set of experiments, we want to examine the impact of different modeling contexts, i.e., using a single vs a pairwise model, on the prediction outcome. To visualize the difference, we present the application of a linear model, LMM, in both single and pairwise scaling scenarios. To determine whether time-of-day is a factor that impacts the workload performance, we build models using the metrics from the three data groups corresponding to the three executions of the workloads on the SKUs (Figure 8). In Figure 8b, we can see that that the transition between hardware configurations is different in each set of experiments although we observe that in general, the throughput seems to increase with an increase in available CPUs as one would expect. However, there are variations in the pairwise model that are not captured if we use the same data and build a single model (Figure 8a). This observation is not unique to linear models but we observe similar behavioral characteristics in non-linear scaling models as shown in Figure 9.

Insight 5. *Although a single model can capture the trend of workload scaling, the transition between specific hardware configurations can be modeled more accurately using pairwise scaling models.*

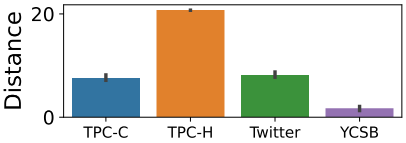


Figure 10: Hist-FP $L_{2,1}$ similarity of YCSB to other workloads.

6.2.2 Modeling Strategies. We now focus our evaluation on the impact of different modeling strategies. Table 6 shows an overview of all evaluated algorithms for single as well as pairwise models using NRMSE as the evaluation metric. We average the NRMSE over all upwards scaling pairs within an experiment, i.e., the six combinations scaling up between 2, 4, 8, and 16 CPU nodes respectively. We setup a baseline that assumes inverse linear scaling relationship between CPU and latency, i.e. if number of CPU increase from 2 to 4, the latency reduce by half. Among all learning methods, gradient boosting performs the best with a mean NRMSE of around 0.271 for both contexts, followed closely by pairwise SVM with a mean NRMSE of 0.279 across all evaluated workloads. However, we observe that on average, the training time for SVM is 10× to 40× faster than gradient boosting. We note that neural-network performs the worst, which is expected as this method is more suitable for datasets with a larger number of data points and features. Across all models, the lowest observed NRMSE for a workload is 0.231 while the highest (excluding NNet) is 0.366, corresponding to a deviation of ≈ 23% and 37% from the actual observed throughput value ranges respectively. Given our limited scope of experiments, such high variation can be partially attributed to noise within the experiment execution. Nevertheless, we observe that most strategies behave similarly within their model restrictions, i.e., independent of whether a single or pairwise model has been chosen. Finally, we observe that all methods performs substantially better than the naive linear scaling baseline.

Insight 6. *Simpler ML models are more suitable for modeling scaling behaviors than complex models. Among the simple models, the choice of model context is comparatively more impactful on the workload prediction performance than the choice of modeling strategy.*

Table 6: Mean throughput prediction (NRMSE) of 5-fold cross validation by models built with each configuration. We use base workload with varying workload types (workload name) and number of concurrent terminals (subscript in workload name). The lowest mean NRMSE for each workload setting and overall for each method is shown in bold. The lowest mean NRMSE among all workloads is shaded green, and the largest (excluding NNet) shaded yellow.

Strategy		Mean Training Time (s)	Mean Test NRMSE							
			TPC-C_4	TPC-C_8	TPC-C_{32}	Twitter$_4$	Twitter$_8$	Twitter$_{32}$	TPC-H_1	Mean
Pairwise	Regression	0.0597	0.293	0.303	0.269	0.343	0.320	0.315	0.236	0.297
	SVM	0.0327	0.276	0.297	0.270	0.304	0.284	0.287	0.237	0.279
	LMM	1.2066	0.281	0.306	0.275	0.305	0.290	0.285	0.240	0.283
	GB	0.5846	0.271	0.292	**0.255**	**0.290**	0.284	**0.276**	**0.231**	**0.271**
	MARS	0.1164	0.304	0.322	0.297	0.286	0.337	0.299	0.258	0.300
	NNet	2.7698	0.734	0.913	1.034	1.200	2.465	7.695	2.781	2.403
Single	Regression	0.0011	0.304	0.315	0.282	0.316	0.359	0.324	0.249	0.307
	SVM	0.0032	0.269	0.290	0.281	0.304	0.294	0.285	0.255	0.283
	LMM	0.4234	0.340	0.312	0.322	0.352	0.366	0.296	0.256	0.321
	GB	0.1105	**0.264**	**0.287**	0.261	**0.290**	**0.283**	0.280	0.245	0.273
	MARS	0.0187	0.301	0.305	0.279	0.302	0.356	0.307	0.249	0.300
	NNet	0.3045	0.667	0.931	1.151	1.185	2.768	8.547	1.971	2.460
Baseline			9.897	12.503	17.959	21.072	67.341	90.970	0.552	31.470

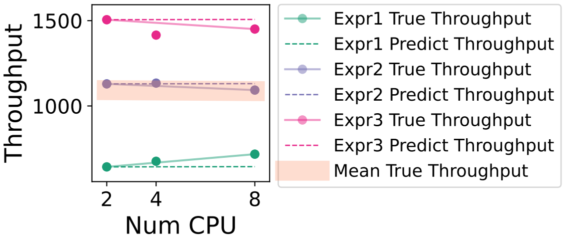


Figure 11: Throughput prediction of YCSB scaling up from 2 CPUs to 8 CPUs using pairwise scaling models of existing TPC-C workloads and the 3 runs of YCSB on SKU with 2 CPUs. The different colors show the performance and scaling trends for the 3 TPC-C runs. A scaling prediction is generated for each experiment run (denoted by the dashed lines).

6.2.3 End-to-End Prediction. To test how well our end-to-end prediction framework works for workloads on different hardware settings, we set up an experiment using YCSB as our target workload and measure its workload on two different SKUs with 2 and 8 CPUs respectively. Using the latter for verification purposes only, we assume that only the data of the 2 CPU hardware configuration is known to the pipeline. Referring back to the insights found in Section 5.1, the query plans and resource utilization metrics are encoded via Hist-FP and we use top-7 among all plan features and resource utilization features obtained by applying RFE with Logistic Regression. To compute the similarity between the new and existing workload types, we use the $L_{2,1}$ norm and calculate the distances as shown in Figure 10. Here, we observe that YCSB is most similar to TPC-C, closely followed by Twitter.

We then use a pairwise model in combination with SVM to determine the scaling behavior from 2 CPUs to 8 CPUs for our previously observed TPC-C experiments. Figure 11 shows the comparison of the actual measurements as well as the prediction based on TPC-C. We first observe that the ground truth has performance variations due to noise on the virtual machines shown in the variation of the measurements at different times of day, each set of experiments is colored differently. However, when averaging the throughput values, which would happen with a sufficient number of repeated experiments or measurements used for training, the resulting scaling trend flattens. This matches our prediction based on TPC-C which results in a NRMSE of 0.0948 for this experiment. To show that with pairwise scaling prediction models, we are able to prediction the workload performance when scaling multiple hardware configurations, we set up a second suite of experiments as follows: We use YCSB with 8 concurrent terminals as our target workload and measure its performance on 2 different SKUs (S1) 4 CPUs and 32 GB memory; and (S2) 8 CPUs and 64 GB memory. We run TPC-C, TPC-H, and Twitter on these two configurations and build pairwise scaling models. To test our scaling predictions, we assume that only metrics from the first configuration is known to us (i.e. YCSB run on S1) and, using the same methodology as in our prior experiment, we then determine the workload most similar to YCSB which is TPC-C. Using the scaling model corresponding to TPC-C, we then predict the workload throughput of YCSB on the second SKU S2. The predicted throughput of the model is ∼ 1100 req/sec, a 0.206 mean average percentage error (MAPE) from the true workload performance (1400 req/sec). On the other hand, if we use Twitter as the reference workload, the predicted throughput is ∼ 600 with an error of 0.563 from the actual performance. This experiment demonstrates the applicability of the prediction framework for multi-dimensional SKUs, an important topic for future research.

6.3 Key Takeaways

In our evaluation, we have found pairwise scaling prediction models to have the highest accuracy when modeling the scaling behavior from one SKU to another. Our observations about the accuracy for single vs pairwise scaling models will amplify for non-linear scaling decisions, i.e., where we want to scale to a SKU that has different resources such memory, network, chip design etc. Furthermore, we observe that a complex model (NNet) can lead to higher prediction error compared to simple models. The simple models differed only minimally from each other in prediction error with gradient boosting and SVM being the best performing. We see a relatively high prediction error which we traced to noise in the experimentation runs. We expect the error to decrease with an increase in training data. Finally, we applied our end-to-end pipeline to two workload runs, varying only CPU and CPU-Memory pairs respectively, and showcased how the framework would function in practice.

7 Discussion

In this work, we have explored an end-to-end pipeline for workload prediction. We now summarize our insights from our experimental evaluations, and outline future work that merits more investigation.

No feature set fits all workloads. In our experiments on feature selection strategies, we have discovered that although the choice of representative features correlates across types of workloads, they can be vastly different when comparing different types of workloads. Our experiments suggest that while finding a universally representative (minimal) feature set across workloads is unlikely if not impossible, there are best practices that we can follow when choosing features sets for pipelines such as ours:

(1) Using too few features may result in overlooking important characteristics of a workload.
(2) Using too many features may result in dilution of the distinctiveness of a workload run, in addition to an increased computational overhead and chance of overfitting.
(3) Workloads with comparable scaling behavior typically show a similar feature importance ordering.
(4) Based on our experiments, plan-only or a combination of plan-based and resource-utilization features often produce better downstream results.

Not all similarity computation techniques are equal. Workload similarity computation is an effective mechanism to reduce the search space for workload prediction if utilized properly. We have found that (*i*) clustering algorithms are highly sensitive to which features are used for similarity computation, and that (*ii*) not all strategies perform well along our evaluation dimensions of reliability, discrimination power, and robustness. Overall, we have found that norm-based algorithms tend to perform better and that fingerprinting-based data representation performs on average better than timeseries-based data representation.

Predict scaling between rather than across SKUs. In our experimental evaluation on workload resource prediction, we have focused on a relatively simple use case, i.e., resource scaling between SKUs that only vary in the number of CPU cores. Even for this use case, we have observed that predicting scaling behavior is not trivial: It is neither strictly linear nor fully predictable in a cloud setting due to environmental noise. Given a wide range of available SKUs for cloud providers, we believe that pair-wise scaling prediction models show more promise than models that assume a continuous performance relationship between hardware configurations. This is substantiated in our experiments, where the choice of the model context dominates the choice and impact of the modeling algorithm. We posit that these observations will amplify if we modify the SKUs not only along one dimension (CPUs) but multiple (memory, network, storage etc.).

Future Work. We believe that our findings w.r.t. workload resource predictions can guide and enhance future research in this area. Our findings indicate that there are opportunities for improving existing pipelines and exploring novel, more targeted pipelines for resource prediction based on workload utilization. They further show that workload modeling and characterization is still a relatively underutilized space: If we can model (the similarity between) workloads more accurately, we can improve our ML prediction tasks significantly. More specifically, our work shows that the wrong choice can oftentimes have detrimental impact on downstream ML algorithms, thus raising the open question of how we can ensure data quality within such pipelines. The study of dimensionality reduction techniques for feature selection is also an important problem. A more detailed discussion can be found in the Appendix of the full version of the paperAppendix B.

In this paper, we have focused on the setting where the resource usage is *not* saturated, as is commonly found in practice [3]. However, there also exist scenarios where resources may get bottlenecked. For such cases, Roofline modeling [96] proves valuable. The Roofline model assumes that any execution on a specific hardware is bounded either by its memory resources or its compute resources. All executions on this particular hardware correspond to points within the space bounded by piecewise linear functions that act as performance ceilings. In our setting, Roofline models can be seamlessly integrated with linear prediction strategies used in the prediction component of the pipeline. For instance, a Roofline style model that plots throughput on the y-axis and #CPUs on the x-axis for a fixed main memory can reveal that adding more compute resources to a memory-bound workload is ineffective and thus, unlikely to enhance throughput. We defer more details to the full version of the paper Appendix B. In more complex settings involving distributed systems, extensions of the Roofline model, such as the Ridgeline model [17], can be combined with non-linear modeling strategies to better capture hardware-specific trends across multiple variables, such as network and memory capacity. These research directions pave the way for an exciting agenda in studying workload scaling prediction. The applicability of our framework to other types of relational workloads (such as batched workloads or high performance computing workloads), as well to non-relational settings (such as key-value stores) is also an important problem. Note that our prediction pipeline performs SKU prediction based on workload similarity by only looking at metrics obtained from prior runs of different workloads. Thus, in principle, our solution can be applied to any type of workload as long as the appropriate metrics can be collected. Since our implementation is built on top of BenchBase, any relational workload that can be executed via BenchBase can potentially benefit from our work without additional overhead.

8 Conclusion

In this work, we have conducted a thorough analysis of a common ML-based end-to-end pipeline for workload resource scaling prediction. We first discussed its potential pitfalls and impact of algorithm choices for each of the different building blocks of the pipeline (feature selection, similarity computation, and resource prediction) and substantiated our insights with experiments targeted to highlight the different techniques and their impact on the prediction pipeline. We then summarized our observations and proposed a series of recommendations based on our experimental observations that are likely to improve the prediction quality of ML-based resource prediction pipelines.

References

[1] [n. d.]. https://man7.org/linux/man-pages/man1/perf.1.html

[2] [n. d.]. https://learn.microsoft.com/en-us/sql/t-sql/statements/set-statistics-xml-transact-sql?view=sql-server-ver16

[3] [n. d.]. 2024 Kubernetes Cost Benchmark Report. https://cast.ai/kubernetes-cost-benchmark/

[4] [n. d.]. Cutting costs in the cloud: six strategies for SaaS companies. https://www.ey.com/en_us/insights/tmt/cutting-costs-in-the-cloud-six-strategies-for-saas-companies

[5] Hervé Abdi and Lynne J. Williams. 2010. Principal component analysis. *WIREs Computational Statistics* 2, 4 (July 2010), 433–459. https://doi.org/10.1002/wics.101

[6] Noman Bashir, Nan Deng, Krzysztof Rzadca, David Irwin, Sree Kodak, and Rohit Jnagal. 2021. Take it to the limit: peak prediction-driven resource overcommitment in datacenters. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21)*. ACM. https://doi.org/10.1145/3447786.3456259

[7] Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. 2014. Fitting linear mixed-effects models using lme4. *arXiv preprint arXiv:1406.5823* (2014).

[8] R. Battiti. 1994. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks* 5, 4 (1994), 537–550. https://doi.org/10.1109/72.298224

[9] Verónica Bolón-Canedo, Amparo Alonso-Betanzos, Laura Morán-Fernández, and Brais Cancela. 2022. *Feature Selection: From the Past to the Future.* Springer International Publishing, Cham, 11–34. https://doi.org/10.1007/978-3-030-93052-3_2

[10] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. https://doi.org/10.1023/a:1010933404324

[11] David Buchaca, Josep LLuis Berral, Chen Wang, and Alaa Youssef. 2020. Proactive Container Auto-scaling for Cloud Native Machine Learning Services. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE. https://doi.org/10.1109/cloud49709.2020.00070

[12] Joyce Cahoon, Wenjing Wang, Yiwen Zhu, Katherine Lin, Sean Liu, Raymond Truong, Neetu Singh, Chengcheng Wan, Alexandra M. Ciortea, Sreraman Narasimhan, and Subru Krishnan. 2022. Doppler: Automated SKU Recommendation in Migrating SQL Workloads to the Cloud. *Proc. VLDB Endow.* 15, 12 (2022), 3509–3521. https://www.vldb.org/pvldb/vol15/p3509-zhu.pdf

[13] Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. 2015. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Transactions on Cloud Computing* 3, 4 (Oct. 2015), 449–458. https://doi.org/10.1109/tcc.2014.2350475

[14] George Casella and Roger Berger. 2024. *Statistical inference.* CRC Press.

[15] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna Gummadi. 2010. Measuring User Influence in Twitter: The Million Follower Fallacy. *Proceedings of the International AAAI Conference on Web and Social Media* 4, 1 (May 2010), 10–17. https://doi.org/10.1609/icwsm.v4i1.14033

[16] Surajit Chaudhuri, Vivek Narasayya, and Ravishankar Ramamurthy. 2008. Diagnosing Estimation Errors in Page Counts Using Execution Feedback. In *2008 IEEE 24th International Conference on Data Engineering*. 1013–1022. https://doi.org/10.1109/ICDE.2008.4497510

[17] Fabio Checconi, Jesmin Jahan Tithi, and Fabrizio Petrini. 2022. Ridgeline: A 2d roofline model for distributed systems. *arXiv preprint arXiv:2209.01368* (2022).

[18] Yanjiao Chen, Long Lin, Baochun Li, Qian Wang, and Qian Zhang. 2021. Silhouette: Efficient cloud configuration exploration for large-scale analytics. *IEEE Transactions on Parallel and Distributed Systems* 32, 8 (2021), 2049–2061.

[19] Zhijia Chen, Yuanchang Zhu, Yanqiang Di, and Shaochong Feng. 2015. Self-Adaptive Prediction of Cloud Resource Demands Using Ensemble Model and Subtractive-Fuzzy Clustering Based Fuzzy Neural Network. *Computational Intelligence and Neuroscience* 2015 (2015), 1–14. https://doi.org/10.1155/2015/919805

[20] Hosik Choi, Donghwa Yeo, Sunghoon Kwon, and Yongdai Kim. 2011. Gene selection and prediction for cancer classification using support vector machines with a reject option. *Computational Statistics & Data Analysis* 55, 5 (may 2011), 1897–1908. https://doi.org/10.1016/j.csda.2010.12.001

[21] Georgia Christofidi, Konstantinos Papaioannou, and Thaleia Dimitra Doudali. 2023. Is Machine Learning Necessary for Cloud Resource Usage Forecasting?. In *Proceedings of the 2023 ACM Symposium on Cloud Computing (SoCC '23)*. ACM. https://doi.org/10.1145/3620678.3624790

[22] Georgia Christofidi, Konstantinos Papaioannou, and Thaleia Dimitra Doudali. 2023. Toward Pattern-based Model Selection for Cloud Resource Forecasting. In *Proceedings of the 3rd Workshop on Machine Learning and Systems (EuroMLSys '23)*. ACM. https://doi.org/10.1145/3578356.3592588

[23] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing (SOCC '10)*. ACM. https://doi.org/10.1145/1807128.1807152

[24] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning* 20, 3 (Sept. 1995), 273–297. https://doi.org/10.1007/bf00994018

[25] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 153–167.

[26] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM. https://doi.org/10.1145/3132747.3132772

[27] Christopher Dabrowski and Fern Hunt. 2009. Using Markov chain analysis to study dynamic behaviour in large-scale grid systems. In *Proceedings of the Seventh Australasian Symposium on Grid Computing and E-Research - Volume 99* (Wellington, New Zealand) *(AusGrid '09)*. Australian Computer Society, Inc., AUS, 29–40.

[28] Sudipto Das, Feng Li, Vivek R Narasayya, and Arnd Christian König. 2016. Automated demand-driven resource scaling in relational database-as-a-service. In *Proceedings of the 2016 International Conference on Management of Data*. 1923–1934.

[29] Shaleen Deep, Anja Gruenheid, Paraschos Koutris, Jeffrey F. Naughton, and Stratis Viglas. 2020. Comprehensive and Efficient Workload Compression. *Proc. VLDB Endow.* 14, 3 (2020), 418–430. https://doi.org/10.5555/3430915.3442439

[30] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB* 7, 4 (2013), 277–288. http://www.vldb.org/pvldb/vol7/p277-difallah.pdf

[31] Javad Dogani, Farshad Khunjush, Mohammad Reza Mahmoudi, and Mehdi Seydali. 2022. Multivariate workload and resource prediction in cloud computing using CNN and GRU by attention mechanism. *The Journal of Supercomputing* 79, 3 (Sept. 2022), 3437–3470. https://doi.org/10.1007/s11227-022-04782-z

[32] Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, and Eli Upfal. 2011. Performance prediction for concurrent database workloads. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD/PODS '11)*. ACM. https://doi.org/10.1145/1989323.1989359

[33] Martyn Ellison, Radu Calinescu, and Richard F Paige. 2018. Evaluating cloud database migration options using workload models. *Journal of Cloud Computing* 7 (2018), 1–18.

[34] Jerome H Friedman. 1991. Multivariate adaptive regression splines. *The annals of statistics* 19, 1 (1991), 1–67.

[35] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics* 29, 5 (Oct. 2001). https://doi.org/10.1214/aos/1013203451

[36] Jerome H. Friedman. 2002. Stochastic gradient boosting. *Computational Statistics and Data Analysis* 38, 4 (Feb. 2002), 367–378. https://doi.org/10.1016/s0167-9473(01)00065-2

[37] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. 2019. Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. ACM. https://doi.org/10.1145/3297858.3304004

[38] Jiechao Gao, Haoyu Wang, and Haiying Shen. 2020. Machine learning based workload prediction in cloud computing. In *2020 29th international conference on computer communications and networks (ICCCN)*. IEEE, 1–9.

[39] Ashit Gosalia and Xin Zhang. 2008. Automatic plan choice validation using performance statistics. In *Proceedings of the 1st international workshop on Testing database systems*. 1–6.

[40] CMU DB Group. 2024. BenchBase. https://github.com/cmu-db/benchbase Accessed on 07.16.2024.

[41] Yanghu Guo and Wenbin Yao. 2018. Applying gated recurrent units pproaches for workload prediction. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE. https://doi.org/10.1109/noms.2018.8406290

[42] Antony S Higginson, Mihaela Dediu, Octavian Arsene, Norman W Paton, and Suzanne M Embury. 2020. Database workload capacity planning using time series analysis and machine learning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 769–783.

[43] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-shot cost models for out-of-the-box learned cost prediction. *Proceedings of the VLDB Endowment* 15, 11 (July 2022), 2361–2374. https://doi.org/10.14778/3551793.3551799

[44] Geoffrey E. Hinton. 1989. Connectionist learning procedures. *Artificial Intelligence* 40, 1–3 (Sept. 1989), 185–234. https://doi.org/10.1016/0004-3702(89)90049-0

[45] Daniel S. Hirschberg. 1977. Algorithms for the Longest Common Subsequence Problem. *Journal of the ACM* 24, 4 (oct 1977), 664–675. https://doi.org/10.1145/322033.322044

[46] Mohammad Hossain, Derssie Mebratu, Niranjan Hasabnis, Jun Jin, Gaurav Chaudhary, and Noah Shen. 2022. CWD: A Machine Learning based Approach to Detect Unknown Cloud Workloads. arXiv:2211.15739 [cs.DC]

[47] Robert Hummel. 1977. Image enhancement by histogram transformation. *Computer Graphics and Image Processing* 6, 2 (1977), 184–195. https://doi.org/10.1016/S0146-664X(77)80011-7

[48] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2014. An Efficient Approach for Assessing Hyperparameter Importance. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32* (Beijing, China) *(ICML'14)*. JMLR.org, I–754–I–762.

[49] Salam Ismaeel, Ayman Al-Khazraji, and Ali Miri. 2019. An Efficient Workload Clustering Framework for Large-Scale Data Centers. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*. IEEE. https://doi.org/10.1109/icmsao.2019.8880305

[50] Gueyoung Jung, Tridib Mukherjee, Shruti Kunde, Hyunjoo Kim, Naveen Sharma, and Frank Goetz. 2013. Cloudadvisor: A recommendation-as-a-service platform for cloud configuration and pricing. In *2013 IEEE Ninth World Congress on Services*. IEEE, 456–463.

[51] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (Oct. 2002), 422–446. https://doi.org/10.1145/582415.582418

[52] Md. Ebtidaul Karim, Mirza Mohd Shahriar Maswood, Sunanda Das, and Abdullah G. Alharbi. 2021. BHyPreC: A Novel Bi-LSTM Based Hybrid Recurrent Neural Network Model to Predict the CPU Workload of Cloud Virtual Machine. *IEEE Access* 9 (2021), 131476–131495. https://doi.org/10.1109/access.2021.3113714

[53] Arijit Khan, Xifeng Yan, Shu Tao, and N. Anerousis. 2012. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*. IEEE. https://doi.org/10.1109/noms.2012.6212065

[54] In Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey. 2018. CloudInsight: Utilizing a Council of Experts to Predict Future Cloud Application Workloads. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE. https://doi.org/10.1109/cloud.2018.00013

[55] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] https://arxiv.org/abs/1412.6980

[56] Max Kuhn, Kjell Johnson, et al. 2013. *Applied predictive modeling*. Vol. 26. Springer.

[57] Avivit Levy, B. Riva Shalom, and Michal Chalamish. 2024. A Guide to Similarity Measures. arXiv:2408.07706 [cs.IR] https://arxiv.org/abs/2408.07706

[58] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. 2012. Robust estimation of resource consumption for SQL queries using statistical techniques. *Proceedings of the VLDB Endowment* 5, 11 (July 2012), 1555–1566. https://doi.org/10.14778/2350229.2350269

[59] Xueyi Liu, Chuanhou Gao, and Ping Li. 2012. A comparative analysis of support vector machines and extreme learning machines. *Neural Networks* 33 (2012), 58–66.

[60] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing* 12 (2014), 559–592.

[61] lschr and luhk. 2023. *schuetzgroup/sdt-python: v18.0*. https://doi.org/10.5281/zenodo.8028374

[62] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD/PODS '18)*. ACM. https://doi.org/10.1145/3183713.3196908

[63] Sayanta Mallick, Gaetan Hains, and Cheikh Sadibou Deme. 2012. A resource prediction model for virtualization servers. In *2012 International Conference on High Performance Computing and Simulation (HPCS)*. IEEE. https://doi.org/10.1109/hpcsim.2012.6266990

[64] Ryan Marcus and Olga Papaemmanouil. 2016. WiSeDB: A Learning-based Workload Management Advisor for Cloud Databases. *Proceedings of the VLDB Endowment* 9, 10 (2016).

[65] T. Marill and D. Green. 1963. On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory* 9, 1 (1963), 11–17. https://doi.org/10.1109/TIT.1963.1057810

[66] Mohammad Masdari and Afsane Khoshnevis. 2020. A survey and classification of the workload forecasting methods in cloud computing. *Cluster Computing* 23, 4 (2020), 2399–2424.

[67] Andrzej Maćkiewicz and Waldemar Ratajczak. 1993. Principal components analysis (PCA). *Computers & Geosciences* 19, 3 (March 1993), 303–342. https://doi.org/10.1016/0098-3004(93)90090-r

[68] Steffi Melinda. 2016. *A survey of feature selection approaches for scalable machine learning*. Ph. D. Dissertation. Doctoral Dissertation, Technische Universität Berlin.

[69] Irfan Mohiuddin and Ahmad Almogren. 2019. Workload aware VM consolidation method in edge/cloud computing for IoT applications. *J. Parallel and Distrib. Comput.* 123 (2019), 204–214.

[70] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases* (Seoul, Korea) *(VLDB '06)*. VLDB Endowment, 1049–1058.

[71] Karl Pearson. [n. d.]. VII. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London* 58 ([n. d.]), 240 – 242.

[72] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[73] Chenglei Peng, Yang Li, Yao Yu, Yu Zhou, and Sidan Du. 2018. Multi-step-ahead Host Load Prediction with GRU Based Encoder-Decoder in Cloud Computing. In *2018 10th International Conference on Knowledge and Smart Technology (KST)*. IEEE. https://doi.org/10.1109/kst.2018.8426104

[74] Adrian Daniel Popescu, Andrey Balmin, Vuk Ercegovac, and Anastasia Ailamaki. 2013. Predict: towards predicting the runtime of large scale iterative analytics. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1678–1689.

[75] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. 2018. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–33.

[76] Jennie Rogers, Olga Papaemmanouil, Ugur Çetintemel, and Eli Upfal. 2014. Contender: A Resource Modeling Approach for Concurrent Query Performance Prediction. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy (Eds.). OpenProceedings.org, 109–120. https://doi.org/10.5441/002/EDBT.2014.11

[77] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. 2020. Autopilot: workload autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*. ACM. https://doi.org/10.1145/3342195.3387524

[78] H. Sakoe and S. Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26, 1 (feb 1978), 43–49. https://doi.org/10.1109/tassp.1978.1163055

[79] Ali Shahidinejad and Mostafa Ghobaei-Arani. 2020. Joint computation offloading and resource provisioning for e<scp>dge-cloud</scp> computing environment: A machine learning-based approach. *Software: Practice and Experience* 50, 12 (Sept. 2020), 2212–2230. https://doi.org/10.1002/spe.2888

[80] Maxim Vladimirovich Shcherbakov, Adriaan Brebels, Nataliya Lvovna Shcherbakova, Anton Pavlovich Tyukov, Timur Alexandrovich Janovsky, Valeriy Anatol'evich Kamaev, et al. 2013. A survey of forecast error measures. *World applied sciences journal* 24, 24 (2013), 171–176.

[81] Zhiming Shen, Qin Jia, Gur-Eyal Sela, Ben Rainero, Weijia Song, Robbert van Renesse, and Hakim Weatherspoon. 2016. Follow the sun through the clouds: Application migration for geographically shifting workloads. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 141–154.

[82] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 1–14.

[83] Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. 2016. Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery* 31, 1 (feb 2016), 1–31. https://doi.org/10.1007/s10618-016-0455-0

[84] Tarique Siddiqui, Saehan Jo, Wentao Wu, Chi Wang, Vivek Narasayya, and Surajit Chaudhuri. 2022. ISUM: Efficiently compressing large and complex workloads for scalable index tuning. In *Proceedings of the 2022 International Conference on Management of Data*. 660–673.

[85] Alex J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and Computing* 14, 3 (Aug. 2004), 199–222. https://doi.org/10.1023/b:stco.0000035301.49549.88

[86] Ahmed A. Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. 2008. Automatic virtual machine configuration for database workloads. *ACM Transactions on Database Systems* 35, 1 (Feb. 2008), 1–47. https://doi.org/10.1145/1670243.1670250

[87] Cédric St-Onge, Nadjia Kara, Omar Abdel Wahab, Claes Edstrom, and Yves Lemieux. 2020. Detection of time series patterns and periodicity of cloud computing workloads. *Future Generation Computer Systems* 109 (Aug. 2020), 249–261. https://doi.org/10.1016/j.future.2020.03.059

[88] G. W. Stewart. 1993. On the Early History of the Singular Value Decomposition. *SIAM Rev.* 35, 4 (Dec. 1993), 551–566. https://doi.org/10.1137/1035134

[89] Robert Tibshirani. 2018. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (12 2018), 267–288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x arXiv:https://academic.oup.com/jrsssb/article-pdf/58/1/267/49098631/jrsssb_58_1_267.pdf

[90] Transaction Processing Performance Council (TPC). 2010. TPC-C Benchmark Revision 5.11.0. https://www.tpc.org/tpcc/

[91] Transaction Processing Performance Council (TPC). 202. TPC-DS Benchmark Revision 3.2.0. https://www.tpc.org/tpcds/

[92] Transaction Processing Performance Council (TPC). 2022. TPC-H Benchmark Revision 3.0.1. https://www.tpc.org/tpch/

[93] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 363–378. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/venkataraman

[94] Michael E. Wall, Andreas Rechtsteiner, and Luis M. Rocha. [n. d.]. *Singular Value Decomposition and Principal Component Analysis*. Kluwer Academic Publishers, 91–109. https://doi.org/10.1007/0-306-47815-3_5

[95] A. W. Whitney. 1971. A Direct Method of Nonparametric Measurement Selection. *IEEE Trans. Comput.* 20, 9 (sep 1971), 1100–1103. https://doi.org/10.1109/T-C.1971.223410

[96] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

[97] Wentao Wu, Yun Chi, Hakan Hacígümüş, and Jeffrey F. Naughton. 2013. Towards predicting query execution time for concurrent and dynamic database workloads. *Proceedings of the VLDB Endowment* 6, 10 (Aug. 2013), 925–936. https://doi.org/10.14778/2536206.2536219

[98] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query Execution Time Prediction in Amazon Redshift. In *Companion of the 2024 International Conference on Management of Data (SIGMOD/PODS '24)*. ACM. https://doi.org/10.1145/3626246.3653391

[99] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. 2014. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers* 16 (2014), 7–18.

[100] Yongjia Yu, Vasu Jindal, I-Ling Yen, and Farokh Bastani. 2016. Integrating Clustering and Learning for Improved Workload Prediction in the Cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE. https://doi.org/10.1109/cloud.2016.0127

[101] Miranda Zhang, Rajiv Ranjan, Michael Menzel, Surya Nepal, Peter Strazdins, Wei Jie, and Lizhe Wang. 2015. An infrastructure service recommendation system for cloud applications with real-time QoS requirement constraints. *IEEE Systems Journal* 11, 4 (2015), 2960–2970.

[102] Xinyi Zhang, Hong Wu, Yang Li, Jian Tan, Feifei Li, and Bin Cui. 2022. Towards dynamic and safe configuration tuning for cloud databases. In *Proceedings of the 2022 International Conference on Management of Data*. 631–645.

[103] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2023. Automatic Database Knob Tuning: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (Dec. 2023), 12470–12490. https://doi.org/10.1109/tkde.2023.3266893

[104] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query performance prediction for concurrent queries using graph embedding. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1416–1428.

[105] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query performance prediction for concurrent queries using graph embedding. *Proceedings of the VLDB Endowment* 13, 9 (May 2020), 1416–1428. https://doi.org/10.14778/3397230.3397238

[106] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67, 2 (2005), 301–320.

Table 7: Example matrix representations directly transformed from query plans and resource utilization statistics.

(a) Query plan matrix with 3 queries and 4 features.

	f_0^i	f_1^i	f_2^i	f_3^i
q_0	63	1	0	1
q_1	9	1	1	0
q_2	134	23.4	4	0

(b) Resource utilization matrix with 3 features captured over 4 timestamps.

	f_0^j	f_1^j	f_2^j
t_0	32.02	175	0.07
t_1	25.23	66	0.069
t_2	20.65	35	0.07
t_3	25.47	27	0.07

Table 8: Equi-width cumulative frequency histogram for hist-FP.

Bin	f_0^i	f_1^i	f_2^i	f_3^i	f_0^j	f_1^j	f_2^j
1	0.333	0.667	0.667	0.667	0.25	0.75	0.25
2	0.667	0.667	0.667	0.667	0.75	0.75	0.25
3	1	1	1	1	1	1	1

Table 9: Matrix of size $2 \times 7 \times 2$, where the maximum number of phases detected is 2, and for each phase we record 2 statistics mean and variance.

Phase		$f_{i,0}$	$f_{i,1}$	$f_{i,2}$	$f_{i,3}$	$f_{j,0}$	$f_{j,1}$	$f_{j,2}$
0	μ	99.91	100.19	100.22	50.16	9.95	99.91	10.13
	σ^2	9.76	10.16	10.43	7.18	3.09	9.76	3.3
1	μ	0	0	0	0	0	9.79	49.76
	σ^2	0	0	0	0	0	3.3	6.75

A Data Representation

If F^i are the available query plan features and the workload consists of q queries, then the query plan feature space can be described as a $i \times q$ matrix. Similarly, if F^j are the resource utilization features measured n times throughout the execution of the workload, then they can be encoded in a $j \times n$ matrix.

An example is shown in Table 7. In Table 7a, each query q has exactly one value per feature. Continuous feature spaces are different in that we do not observe a single value for a feature but a single value *at a certain point in time*. Table 7b shows an example representation of a continuous feature space for our resource utilization measurements.

Histogram-Based Fingerprinting (Hist-FP). The idea behind histogram-based fingerprinting is to generate a normalized frequency histogram of even-sized buckets for each feature. A relative frequency histogram maintains the same value range for different workloads, thus, the comparison between workloads is possible even if the number of different observations differs.

Entry-wise difference is hard represent the difference in shapes for the histograms. For example, we have frequency histograms of 5 bins of the same range: $H_1 = (1, 0, 0, 0, 0)$, $H_2 = (0, 1, 0, 0, 0)$, and $H_3 = (0, 0, 0, 0, 1)$. The distance between all distinct pairs of the 3 histograms are the same, but we should observe that H_1 is more similar to H_2 than H_3, as all values in H_1 and H_2 are on the lower half of the range, where all values in H_3 are in the bin with the largest values of the range.

Comparing cumulative distribution of histograms is an effective and common technique of histogram distance measurement. In this representation, value of bin b'_i is the cumulative count or frequency of all values in bins $b_0, \cdots, b_i$ of the original histogram. Cumulative distribution representations of the 3 frequency $H''_1 = (1, 1, 1, 1, 1)$, $H''_2 = (0, 1, 1, 1, 1)$, and $H''_3 = (0, 0, 0, 0, 1)$. Entry-wise difference summation on the cumulative distribution histograms can also represent the distance relationships better than original histogram representations. An example is shown in Table 8, which shows the equi-width frequency histogram for the example features introduced previously.

Phase-Level Statistical Fingerprinting (Phase-FP). For phase-level fingerprinting, we leverage prior work, [46], that uses change-point detection algorithms, such as the Bayesian change point detection (BCPD) technique [61], in order to pinpoint significant shifts in the statistical properties of the workload. A *phase* then refers to a distinct period or segment within a workload time-series that exhibits unique statistical characteristics or behavior. Note that different features of the same experiment can have unaligned phase ranges and varying number of phases. In practice, we use statistics such as the mean and variance for each phase and zero-pad features with number of phases less than the highest number of phases among all features. If the number of phases for feature i is p_i, then the resulting experiment fingerprint is a 3D matrix of size $(i + j) \times \max_i p_i \times k$, with k denoting the number of statistic measurements used. By considering two types of statistics: mean and variance of each phase, we can summarize the phase-based distribution of our example data as shown in Table 9. In our experiments, the query plan features have only a single phase while BCPD is used to detect phases for the uni-variate time-series for

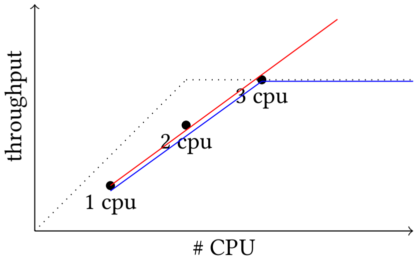


Figure 12: Example Roofline-style model for a fixed workload and memory configuration. Dotted line shows the roofline ceilings. The three points show throughput for varying CPU and the red line is a linear regression model using the points. The blue line combines the ceiling with the linear regression.

each of the resource utilization features. Using the running example, we note that two of the resource utilization features contain two phases while one feature, $f_{j,0}$, has one phase only.

B Incorporating Roofline Models

Figure 12 shows an example of how Roofline modeling can be combined with linear models for performance prediction. Given multiple data points representing runs for different number of CPUs but using the same memory, the red line represents a linear model. Thus, given the model, we can predict the performance with four CPUs. However, the model will give incorrect prediction since the model is unaware of the performance ceiling of the hardware. The dotted lines in the graph show the Roofline performance. It indicates that until the number of CPUs is smaller than three, the workload is compute bound. However, for more than three CPUs, any additional CPU does not increase the performance since memory is the bottleneck. Thus, we can combine the linear model with the Roofline ceiling to obtain the blue line, which is a piece-wise linear model. Thus, the augmented model will correctly use the performance ceiling for prediction and give the correct prediction that the performance on four CPUs is the same as with three CPUs.

C Dimensionality Reduction for Feature Selection

An alternative approach to feature selection methods is to use dimensionality reduction methods that transform the predictor set to a smaller subset of features that explain the variance in the data. Two common methods are *Principal Component Analysis (PCA)* [5, 67] and *Singular Value Decomposition (SVD)* [88, 94]. Dimensionality reduction techniques generally suffer from several drawbacks. They are subject to (i) (semi-)randomness in their feature selection process in that it can generate components that summarize and capture the variance in the data without regard to understanding the predictors (e.g., measurement scales or its distributions) and the modeling objective; and (ii) are of limited use when the feature space is sparse. Moreover, these approaches suffer from reduced interpretability as the new predictor set is a combination of the original feature set.