From Feature Selection to Resource Prediction: A Survey of Commonly Applied Workflows and Techniques [Experiment, Analysis & Benchmark]

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 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 or the end-to-end the three-step pipeline for workload scaling prediction: feature selection, workload similarity, and performance prediction. In this paper, we examine the state-of-the-art strategies for the three components, with the goal to identify 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

Workload understanding in a cloud computing environment is a complex task. Cloud users can spin up dozens or 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 has several benefits: From the provider's point of view, understanding and predicting a user's workloads allows for better task scheduling and resource allocation. Similarly, from the user's point of view, understanding their workload allows them to calculate the trade-off between the allocated (and paid for) resources with expected performance implications. Mapping how a workload behaves (or would behave) on a (new) hardware setup is a non-trivial challenge that can be decomposed into several parts. First, *feature selection* describes how we can characterize a workload. Second, *workload similarity computation* allows us to identify workloads that behave similarly to our current workload. Subsequently, observing the behavior of similar workloads allows us to reason about the behavior of the current workload on various hardware configurations, which we refer to as *workload prediction*.

We are not the first to use this type of workflow to optimize resource utilization. In fact, a similar workflow was introduced in prior work in the context of suggesting hardware configuration for a user's workload [6]. Here, the authors focused on modeling a user's workload as resource requirements only, utilizing features such as CPU and memory consumption but ignoring other features that are used to describe a workload such as query plan features that are commonly leveraged for query performance prediction, [20, 53, 58, 68]. Characterizing workloads as a (set of) feature(s) allows us to then reason about and predict workload behavior using machine learning (ML) techniques. However, when applying any ML algorithm in this context, one of the biggest challenges that we face in practice is to collect a sufficient number of training observations that capture the behavior of a workload on a given hardware setup. To alleviate this problem, many algorithms categorize types of workloads using clustering [36, 43, 67] or time-series based approaches [38, 62], modeling the behavior of workload types rather than single-user workloads. If these techniques are deployed effectively, downstream prediction algorithms become more accurate as they can utilize a larger pool of training data based on the workload clusters instead of overfitting models to a single workload deployment. Each of the three components of an end-to-end prediction pipeline has significant impact on how we can model and estimate how a workload would behave in a different hardware setting, i.e., making the wrong algorithm(s) choice(s) can lead to significant differences in downstream prediction quality.

Table 1: Final YCSB workload scaling prediction normalized root mean square error by using different strategy combinations in the pipeline.

	Pipeline 1	Pipeline 2
Model	Single Linear Regression	Pairwise SVM Regressor
Features Selection	Chi2 top-3 resource	Lasso top-7 plan+resource
Similarity Method	Histogram L21-norm	Multivariate timeseries Correlation-norm
Similar Workloads	TPC-C_8, TPC-C_{32}, Twitter$_8$	TPC-C_4, Twitter$_4$
Normalized error	0.310	0.251

Example 1. *A cloud provider creates an end-to-end prediction framework that allows the use of different algorithms for feature selection, workload clustering, and prediction. In their first attempt to model a customer's workload observed on a four-node hardware configuration, they use Lasso as feature selection strategy and model the workload minimally using the resulting top-3 among the resource utilization features as shown in Table 1. They then compare their workload against workloads that have been observed prior on the same hardware configuration with Correlation-norm of multivariate timeseries matrices as workload similarity strategy, determining that it most closely resembles a mostly transactional workload (Twitter*

and TPC-C with 8 concurrent terminals). Using this observation, they predict that the new workload's performance on a different hardware configuration (16 CPUs) will merely increase from 1014 to 1021 queries per second, deploying a Linear Regression model to do workload prediction. With exactly the same pipeline but a different choice in the deployed ML algorithms, feature selection strategy, workload clustering, and prediction algorithm, we observe that a) a different prior workload (Twitter with 8 terminals and TPC-C with 8 and 32) are chosen as the most similar workload and that b) the prediction accuracy improves with a prediction of 1057, where the actual mean throughput is 1081. Comparing the prediction error of the first configuration, we observe a normalized error of 0.310 while in the second example, we reduce the error to 0.251.

Although the gap between the error values of the two pipelines in Table 1 may seem small, every percentage point difference has a significant impact to the user and/or cloud provider's cost. Specifically, provisioning a hardware configuration that has too few resources may lead to bottlenecks and thus performance degradation while provisioning too many resources leads to added cost and suboptimal resource allocation. Thus, it is imperative that the predictions are as accurate as possible to ensure that the chosen configuration is the best fit for this workload.

Contributions. In this work, we examine the three-step pipeline for workload scaling prediction as outlined above in detail. First, we experimentally examine state-of-the-art feature selection strategies and discuss what type of features are most beneficial and which features have the most impact on 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 workload scaling behavior and how the choice of modeling context as well as algorithm impacts the prediction performance. With our experimental exploration, we show 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 can determine to avoid certain pitfalls due to the choice and application of various ML algorithms in this space.

2 Overview

With this paper, we want to examine the following three questions: a) Which workload characteristics identify a workload?; b) How can we compute workload similarity?; and c) How can we predict the performance of workloads on a different set of resources? These questions correspond to the three building blocks of our end-to-end pipeline for workload scaling predictions as shown in Figure 1. As input, we require that *telemetry* is collected during workload execution which encompasses execution statistics such as the latency of queries, resource utilization etc., as well as one-off characteristics such as information about the query plans. This telemetry is then used to characterize a workload and, subsequently, for workload similarity computation and prediction as outlined next.

Feature Selection. The challenge of feature selection is twofold. First, we want to explore whether (and which) state-of-the-art feature selection mechanisms can help us to uniquely identify a workload. As mentioned above, features are extracted as telemetry while a workload is being executed, thus providing us with a stream of information about a workload containing data points as well as temporal observations. We elaborate on the feature space in our experiments, showing that there are some features that seem to be important across workloads and some that are specific to a workload. The second challenge then is to determine whether there is a representative set of features that minimally captures workloads.

Workload Similarity Computation. Workload similarity computation allows us to cluster workloads based on their similarities. 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 reason about grouping similar workloads and using clusters of workloads for downstream prediction tasks. In this work, we first explore what role data representation plays for aligning the characteristics of workloads and then discuss different similarity computation strategies, evaluating them based on their ability to cluster workload types appropriately and efficiently.

Workload Resource Prediction. As final step in our pipeline, we look at the problem of predicting resource utilization for a workload, 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. Thus, we extend work done previously in the context of modeling the trade-off between cost and quality in [6]. We evaluate several prediction algorithms and discuss which type of models can be 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.

3 Feature Selection

Feature selection is an important challenge as incorrect representation of the workload has immediate impact on any downstream applications. We next discuss different feature selection strategies that are the state-of-the-art, after which we will experimentally show which features they choose and how well they work in practice.

3.1 Overview of Selection Strategies

Feature selection is well-studied topic and there are many algorithms available for this task. Those algorithms help to ensure that relevant predictors are collected and represented in a way that provides good predictive performance. In this work, we focus on feature selection techniques that give a rank or importance score for each feature. We then constrain the chosen feature space by selecting features that fall within a specific rank or importance score from a feature selection algorithm. Different feature selection algorithms use different rationales to rank the utility of features, and therefore resulting in a different feature space if we select the top-k features for the same fixed k. The feature selection methods we test can be classified into 3 categories.

3.1.1 Filter Approach. Feature selection methods in this category evaluate the importance of predictors prior to fitting the model. Predictor relevance is thus determined separate from the model

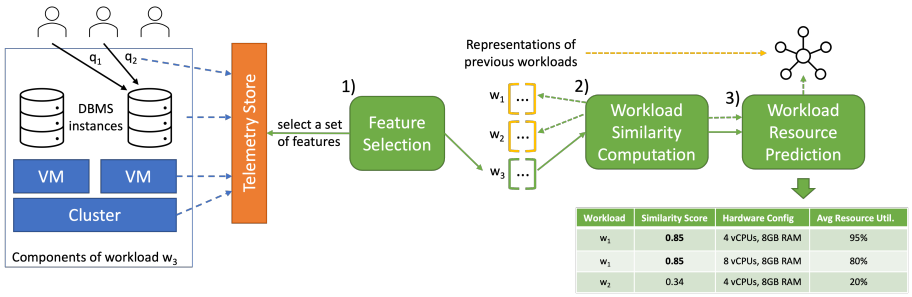


Figure 1: Interaction between the 1) feature selection, 2) workload similarity, and 3) prediction components.

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 the more complex feature selection methods below:

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. While this approach does not take into account the relationship between the predictor and target, this is a simple baseline approach for predictor filtration.

Pearson Correlation Coefficient [50]. 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. Here, we use the absolute values of the correlation coefficients as a means of weighing the importance of each predictor.

Functional Analysis of Variance (fANOVA) [35]. This method measures the fraction of explained variability in the total variability of data. Specifically, the F-test calculates ratio of variability between two sets of data over variability within the two sets of data respectively. In feature selection, we use data values from on feature in the feature space and label values as two sets of input in F-test. The F-test will then generate results measuring the ratio of variability of the current feature getting reflected in the variability of the result labels.

Mutual Information Gain [4]. This metric is used to evaluate the dependency of a feature with the target label. It is the difference of the entropy of data values in a predictor and the conditional entropy of data values in the predictor given the target labels. The difference in entropy signifies a reduced uncertainty in the target due to information gained from the predictor. A difference of zero suggests independence. This method generates mutual information values that can be used as feature importance scores.

3.1.2 Embedded Approach. Methods in this category consists of models that contain built-in feature selection. In other words, the model itself will only 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 to control model variance:

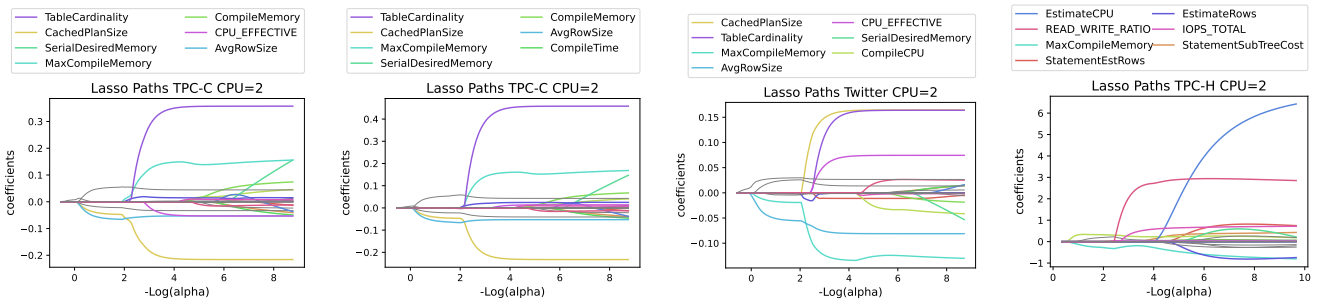
Lasso [63]. This method adds a penalty term to the standard sum of squares objective used in ordinary linear regression to control (or regularize) parameter estimates in cases where multi-collinearity might exist in the data. Lasso is capable of using regularization to not only identify a better model but also conduct feature selection.

Elastic Net [69]. While Lasso provides built-in feature selection by zeroing out highly correlated predictors, it is indifferent in this 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. Ridge regression is effective in dealing with highly correlated predictors by shrinking their estimates toward each other; however, ridge does not do feature selection. By combining L1 and L2 regularization, elastic net model can serve to not only identify models with lower variance but also higher predictor relevance.

3.1.3 Wrapper Approach. These feature selection methods compare a wide range of models that add or remove predictors to optimize model performance. They are search algorithms that seek to find the optimal subset of features that result in the highest model performance. Wrapper methods require multiple rounds of training across multiple types of models until a satisfying subset of features is identified. Methods in this category can typically find good feature subsets at the cost of increased computational complexity.

Recursive Feature Elimination (RFE) [12]. 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) [46, 64]. This method iteratively adds or removes one or more features in a greedy manner based on the prediction performance of the underlying machine learning model. 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



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

Figure 2: Lasso path of features for each experiment using hardware setting of 2 CPUs. Labels only top-7 most important features with non-zero weights.

may result in a more balanced set of predictors based on the scoring metric selected.

3.2 Experiments

To evaluate the different feature selection strategies, we utilize a variety of benchmarks and observe the executed workload over time on different hardware. We use and extend BenchBase, [22], to execute the benchmarks on a local instance of SQL Server, capturing information on the executed query plans as well as system features such as *CPU utilization*, *effective CPU*, *memory utilization*, *total IOPS*, *read/write ratio*, *lock request rate*, and *average lock wait time*. By doing so, we collect multivariate time-series datasets for each experimental run. For each feature, we 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 10 bins. The data representation will be discussed further in Section 4.1.1.

In our evaluation, we use the dataset containing 4 standardized benchmarks (TPC-C, TPC-H, Twitter, and YCSB) and 1 real-world workload with varying hardware settings and concurrency levels to 1) discuss why feature selection is a workload-specific task, 2) identify how we can combine these features to compare workloads, and 3) evaluate different feature selection strategies.

3.2.1 Picking Features 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, in practice, we observe that some features are more suited to characterize specific workloads over others and which features are picked can vary based on the applied feature selection algorithm. We start with a very high dimensional feature space that includes various components of the query plan and runtime resource utilization features that were collected during workloads executed on SQL Server. We summarize all features extracted in Table 2. We first observe that there are some features that consistently ranked 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 across selection mechanisms.

In Figure 2, we visualize the results of one of our evaluated feature selection strategies, Lasso. In these figures, we plot the top-7 features selected by Lasso for a workload on the same hardware setting. Here, the larger the deviation from 0, the higher the feature importance. Figure 2a and Figure 2b 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 spaces, there are also differences. For example, `CPU_EFFECTIVE` is only considered an important feature in Figure 2a. There are numerous factors in the cloud environment affecting the collected metrics and thus, the feature selection results but generally, the more often we run feature selection for the same workload, the more stable our selected features will become.

The second observation is that workloads that are conceptually similar tend to have a similar set of important features. Take the features selected by TPC-C Figure 2a and Twitter Figure 2c as an example. Both workloads share common important features such as the row size, table cardinality, and the cached plan sizes, for a total of six overlapping features. In contrast, both workloads overlap with only one feature of TPC-H in Figure 2d.

Insight 1

The output of feature selection strategies is impacted by the workload, hardware, and even execution characteristics. However, workloads that are similar in nature oftentimes have a similar feature space.

3.2.2 Top-K Feature Selection We have discussed the feature selection strategies that we use for evaluation in Section 3.1. In general, we observe that the output of these strategies can be split into two categories: *Score*-based and *Rank*-based feature selection.

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.

Table 2: Features extracted from resource utilization metrics and query plans.

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	

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		0.025
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
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	0.994	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

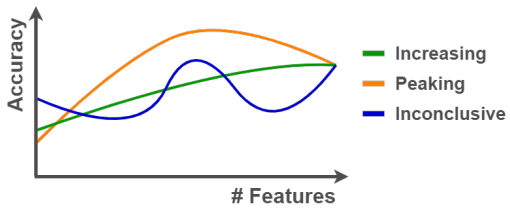


Figure 3: Generalized Accuracy Development Curves.

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. We aggregate the ranks across experiments and select the top-k features with the lowest aggregate rank.

3.2.3 Comprehensive Strategy Evaluation Now that we have determined how we can pick and evaluate feature selection strategies across workloads, we can evaluate the different strategies that we introduced in Section 3.1. 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] when using 2 CPUs. Here, we compute the accuracy as the correct 1-NN clustering of workloads using the top-k features as input for the clustering algorithm.

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 these behavioral patterns is shown in Figure 3 and strategies in Table 3 are color coded to reflect the pattern. As expected, we observe that performance often increases as the number of features included increase; however, this trend may indicate potential overfitting to the dataset. Conversely, we also see that 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 level of accuracy can be reached across all strategies as if all features are used for clustering, though the accuracy varies based on the respective strategy. Some strategies such as SFS have near-perfect accuracy. The primary drawback is the relatively high execution time, also shown in Table 3, which is two to three orders of magnitude higher than that of simple filtration methods which can obtain the same level of accuracy with as low as 7 features.

Key Takeaways. In our experimental evaluation, we considered 16 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 feature selection strategy and the workload that it is applied on. However, there is a correlation between the type of workload and the feature importance ordering; for example, transactional workloads tend to prioritize different features compared to analytical workloads. 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.

4 Workload Similarity

The second question that we examine in this paper is how we can compute workload similarity. As explained in the previous section, a common feature space can be computed across workloads by selecting those top-k features using one of the many available feature selection strategies. This common feature space can then be used to compute the similarity between workloads in more detail. That is, we have observed that the feature space is similar across workloads with the same general characteristics such as observed for transactional vs. analytical workloads (Section 3.2.1). However, the distribution of values within each feature, the load over time, and many more factors may further uniquely impact a workload. Thus,

workload similarity computation has been used as a means to find and group similar workloads to generalize "types" of workloads. In this section, we first introduce state-of-the-art workload similarity computation mechanisms and then experimentally showcase their advantages and disadvantages.

4.1 Overview of Similarity Computation

Workload similarity computation is the problem of aligning two workloads such that their feature spaces become comparable and then measuring the distance for each feature (or aggregate across features) to determine how close these workloads are. The challenges of computing workload similarity are 1) Determining a suitable feature representation for similarity computation, and 2) finding the similarity algorithm that is most suited to the given feature space.

4.1.1 Data Representation During the execution of our workloads, we use DBMS built-in tools to extract 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*.

We now discuss three different techniques to abstract the feature space into a representation that allows users to compare two workloads. Continuous feature spaces can be represented naturally as *mutlivaraiate time-series* (MTS), and thus, time-series based distance measures are a good candidate to measure similarity (which are discussed below). For both of the continuous and discrete feature spaces, describing the feature value distributions makes comparison between workloads possible even if the number of different observations (in our case queries or temporal events) differs. Equi-range frequency histograms [21] and techniques like cumulative frequency distribution [34] over the bins is commonly used to encode distribution information, and we will refer to it as *Histogram-Based Fingerprinting* (Hist-FP). Another strategy is to encode the data by their statistics like mean and variance. Prior work [33] presents *Phase-Level Statistical Fingerprinting* (Phase-FP) that uses change-point detection algorithms, such as the Bayesian change point detection (BCPD) [42], to pinpoint significant shifts in the statistical properties of the workload. We include the detailed data representation examples Appendix A.

4.1.2 Similarity Computation This is the task of calculating the difference of two distinct experiments into in a single numeric score. Depending on the data representation, we can either choose to apply the similarity computation that computes the distance between each feature or, alternatively, we can compute the matrix distance which evaluates the structural dissimilarity between the matrices.

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. The norm-based similarity measures calculate the mathematical space-distance of matrices. It can be applied to MTS and distribution fingerprints. For example, the Canberra distance is a weighted variant of the L_1-norm that is more robust to outliers [25], and Chi-Square distance is a weighted variant of L_2 norm [66] such that:

$$d_{Canberra}(x,y) = \sum_{i=1}^{L} \frac{|x_i - y_i|}{|x_i| + |y_i|} \quad d_{chi2}(x,y) = \sum_{i=1}^{L} \frac{(x_i - y_i)^2}{y_i}$$

Here, each column or row of data matrices can be viewed as a vector. The distance of a matrix pair can be obtained from summing up the distances between vector pairs using methods like mean, sum, weighted sum, etc. When the results are aggregated using summation, we get an special case of $L_{p,q}$ norm, where $q = 1$. Formally, for distance of matrix $A, B \in \mathbb{R}^{m \times n}$, we have $L_{p,q}(A - B) = \left(\sum_{j=1}^{n} \left(\sum_{i=1}^{m} |A_{ij} - B_{ij}|^p\right)^{\frac{q}{p}}\right)^{\frac{1}{q}}$ $L_{1,1}$-norm is the sum of Manhattan distances for all columns of the matrix. $L_{2,1}$-norm is the sum of Euclidean norms for all columns of the matrix. $L_{2,2}$ norm, or Frobenius norm, is another commonly used norm for representing matrix distance.

MTS Distance Measure. In addition to norm-based measurements, the ordered nature of the MTS data representation allows us to consider the shapes of time-series for alignment. For example, there exist algorithms that allow one-to-many or many-to-one point comparison that represent the local distortion of the time-series. For example, Dynamic Time Warping (DTW) [55] matches time-series which has 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. The sum of all entries along a path is the distance of the two time-series according the alignment. 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 generalized into *independent* and *dependent* strategies [59]. *Independent* multivariate-DTW sum 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) [32] that computes the edit distance between multiple time-series'. Here, ϵ is used as an editing threshold and a pair of data points in the two time-series are considered matched as long as their difference are within the threshold ϵ. LCSS measures the length of the most similar sub parts of the two time-series, making it suitable for time-series with different lengths. It also makes LCSS more robust to outliers, as it uses only the subsequence within the threshold rather than summing up pair-wise distances for all points when computing similarity scores. Similarly, dependent LCSS aligns and compares all dimensions of a multivariate time-series together, finding the longest common subsequence across all dimensions; independent LCSS aligns each dimension of the multivariate time-series independently, identifying the longest common subsequence for each dimension separately.

4.2 Experiments

We normalize and transform the data for three standardized workloads, TPC-C, TPC-H, and Twitter to fit the MTS, Hist-FP, and Phase-FP. For Hist-FP, we summarize uni-variate time-series data

Table 4: Similarity computation mechanisms with perfect 1-NN prediction accuracy. We use ✓to show that the mechanism achieves perfect prediction result for the corresponding feature subset, x for non-perfect result, and – to denote when a feature subset is not applicable to the mechanism. Methods not shown achieves no perfect results for any feature set.

Rep.	Methods	Feature Set								
		Plan			Resource			Combined		
		3	7	all	3	5	all	3	7	all
MTS	L11	-	-	-	✓	✓	✓	-	-	-
	L21	-	-	-	✓	✓	✓	-	-	-
	Frob.	-	-	-	✓	✓	✓	-	-	-
	Canb.	-	-	-	✓	✓	✓	-	-	-
	Ind.-DTW	-	-	-	✓	✓	✓	-	-	-
	Dep.-DTW	-	-	-	✓	✓	✓	-	-	-
	Ind.-LCSS	-	-	-	✓	✓	✓	-	-	-
Hist-FP	L21	✓	✓	✓	✓	✓	✓	✓	✓	✓
	L11	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Frob.	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Canb.	✓	✓	✓	✓	✓	✓	✓	✓	x
	Chi2	✓	x	x	x	x	x	x	x	x
Phase-FP	L21	✓	✓	✓	✓	x	✓	x	✓	✓
	L11	✓	✓	✓	✓	✓	✓	x	✓	✓
	Frob.	✓	✓	✓	x	x	x	✓	✓	✓

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 similarity computation mechanisms, we showcase the L11, L21, Frobenius, Canberra, Correlation, and Chi2 norms as distance-based calculation methods for Hist-FP, Phase-FP and MTS as these mechanisms have shown the best overall performance for (a subset of) data representations in our experiments as shown in Table 4. 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 the following dimensions:

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 (no-)match which in essence is the same as finding the most closely related workload run (1-NN).

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 different in distances between the most similar workloads to the least similar workloads as identified by expert judgement.

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

4.2.1 Comparison of Similarity Methods For TPC-C, TPC-H, and Twitter, we execute 9 runs each on our 16-CPU setup. TPC-C and TPC-H are run at 3 different concurrency levels. The raw data is transformed to Hist-FP, Phase-FP, and MTS. For each representation, we choose applicable similarity computation methods among dependent and independent DTW, dependent and independent LCSS, L11, L21, Frobenius, Canberra, Correlation, and Chi2 norms.

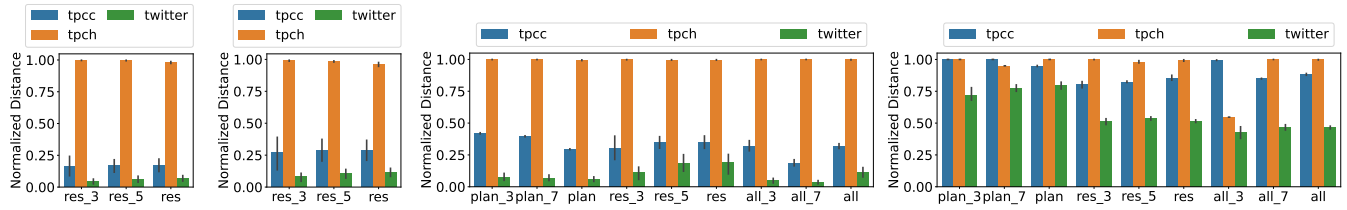
Reliability. Examining the three objectives outlined above, we first focus on the reliability of different norms. We looked at all possible combinations of data representation and similarity computation mechanisms and show the subset that achieves perfect 1-NN prediction in Table 4. Our experiments show that no similarity computation dominates the others, but we generally observe that the L11, L21, and Frobenius norms work well with Hist and Phase-FP across feature set combinations, except if combined with resource-based feature sets only.

Discrimination Power. In order to compare across methods, distances are normalized for each set of features for each method. We present the **normalized distances** from one Twitter and one TPC-C experiment runs compared to other runs in Figures 4 and 5 respectively. As an example, take a look at the results of the L21-norm, executed on Hist-FP, shown in Figure 4c. For most feature set combinations, it correctly identifies the same workloads as identical (Twitter, low distance score), similar (TPC-C, medium distance score), and different (TPC-H, high distance score). This indicates its high discrimination power within this setup. The observation holds for most of the mechanisms for Hist-FP besides Chi2 and Correlation norms, which assign 0 distances to experiments in both Twitter and TPC-H for some feature subsets. In contrast, MTS with L11-norm in Figure 4b and Canberra-norm on Phase-FP in Figure 4d do not perform nearly as well, failing to identify that TPC-C workload behaves similarly to Twitter (both are transactional workloads) and assigning a high distance score instead.

Robustness. Finally, robustness is shown in these figures by the variation in the error bars as each workload has been executed multiple times and thus has (potentially) different values within a feature space. In essence, the smaller the error bars, the more robust the approach. We observe first that experiments utilizing the same data representation shares similar variation in distance computation. Second, most similarity computation mechanisms using Hist-FP showed an acceptable standard error similar to those shown in Figure 4c. In contrast, experiments using independent DTW have a considerably higher error bar, Figure 4a, and the mechanism is therefore considered less robust. This is true for all methods when using only resource utilization features.

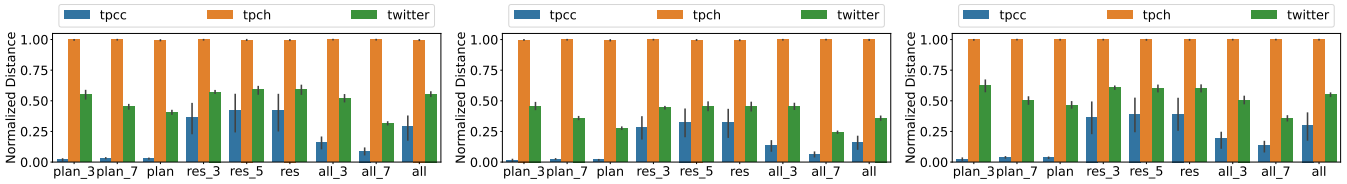
Insight 1

In our experiments, Hist-FP with L21, L11, Frobenius, and Canberra norms have similarity computation results that satisfies the reliability, discrimination power, and robustness criteria. Methods 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.



(a) Ind. DTW on MTS. (b) L11-norm on MTS. (c) L21-norm on Hist-FP. (d) Canberra-norm on Phase-FP.

Figure 4: Similarity results of the Twitter workload.



(a) L11-norm (b) L21-norm (c) Frobenius-norm

Figure 5: Similarity results of the TPC-C workload on Hist-FP data representation.

Table 5: Top-7 features selected by fANOVA 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-7 Plan	Top-5 Resource	Top-7 All
AvgRowSize	LOCK_WAIT_ABS	AvgRowSize
StatementSubTreeCost	MEM_UTILIZATION	StatementSubTreeCost
CachedPlanSize	LOCK_REQ_ABS	LOCK_WAIT_ABS
MaxCompileMemory	CPU_UTILIZATION	CachedPlanSize
CompileMemory	CPU_EFFECTIVE	MEM_UTILIZATION
TableCardinality	-	MaxCompileMemory
EstimateIO	-	TableCardinality

4.2.2 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. 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 top-3 and top-5 subsets for this feature set category. To determine the top features, we use fANOVA as feature selection technique, leveraging our findings from Table 5.

Reliability. Referring again to Table 4, we observe that Hist-FP and Phase-FP tends to achieve perfect prediction accuracy on top-3 plan, top-7 plan, and top-7 all features. Furthermore, the two similarity computation techniques using Phase-FP and the four using Hist-FP produce reliable results consistently across majority of the feature subsets. In general, our results show that most fingerprint representations perform better when using plan compared to resource features. In our experiments, most MTS norms, DTWs, and independent LCSS produce reliable results according to the 1-NN prediction accuracy shown in Table 4.

Discrimination Power. For most feature set combinations, L11, L21, Canberra, and Frobenius norms combined with Hist-FP correctly identify the identical, similar, and different workloads, indicating high discrimination power. However, as we show in Figure 5, we observe an anomaly as three of these methods do not correctly identifying the similar workload for TPC-C on the top-3 plan-features, resource-features, and combined features. More generally, we observe that top-3 feature sets are the least accurate as the feature space is not diverse enough to allow for effective similarity computation. Although the various norms combined with Hist-FP correctly identify identical, similar, and different workloads, the normalized distances using all available features are larger than using the top-7 combined features. This aligns with our insights stated in Section 3.2.3, where we observed that the accuracy may start to decrease as the number of features increases due to noise.

Robustness. In Figures 4 and 5, we observe that experiments using only resource 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. This difference is captured more significantly in plan features, as signified by the feature selection results among all plan and resource utilization features in Table 5.

Insight 2

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).

4.2.3 Similarity Computation using a Production Workload Next, we applied our findings to determine whether similarity computation can be used to describe the characteristics of a workload. We

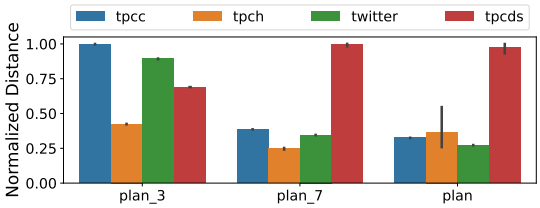


Figure 6: Similarity results of *PW* compared to standardized workloads on different hardware settings.

compare our production workload *PW* to four reference workloads: TPC-C, TPC-H, TPC-DS, and Twitter. These are executed four times on the same hardware setup consisting of 80 virtual cores. We visualize the results using Hist-FP applied in combination with the Canberra norm in Figure 6 using plan features only due to missing resource tracking on the setup instance.

As these numbers indicate, *PW* seems to be more closely related to the TPC-H and Twitter workloads rather than TPC-C and TPC-DS. In fact, we manually confirmed that the queries in *PW* are most commonly simple analytical queries which align better with those found in the first two comparison workloads. We furthermore note in this study that, again, using the top-3 as well as all features is not as accurate as using the top-7 features, substantiating our findings from our synthetic-only workload experiments.

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 promise in providing accurate similarity computation results. Additionally, using these strategies incurs little computational overhead and low storage requirements. We also observe experimentally that plan-based features seem to lead to better similarity computation results, substantiating the need for effective feature selection strategies.

5 Workload Scaling

The final question we want to explore is how we can predict the scaling behavior of an unknown workload based on previously observed workload execution. Recall that *workload scaling* 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: a) defining the *context* of the scaling behavior and b) defining the *strategies* that help to model the scaling behavior. The former answers the question how we should think about the big picture of modeling scaling behavior, 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 scaling behavior.

5.1 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 provisioned for their VM or how much memory 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). Thus, the first question that we need to examine is how we can define the context of a scaling model. We observe two different approaches:

Single Scaling Model Approach. Using 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 performance numbers for pairs of SKUs. The relationship of each pair of data points is linear, thus simplifying the required modeling strategies. The scaling factor then represents the change in performance if one were to move from one SKU to another.

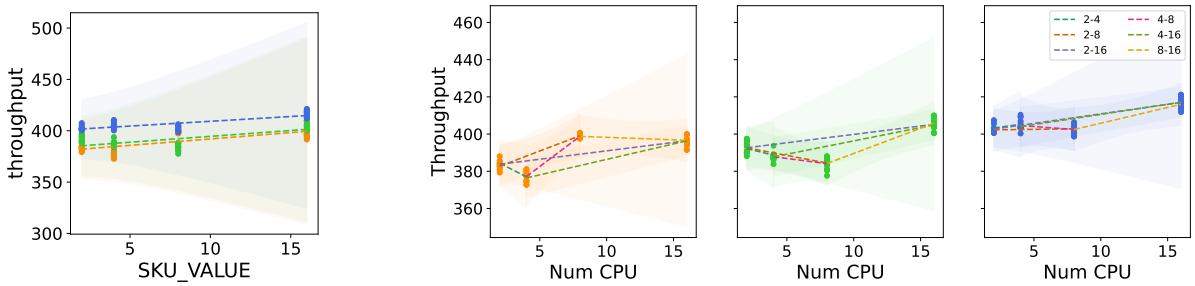
5.1.1 Modeling Strategies Next, we describe several ML modeling strategies that are commonly used for predicting workload behavior. Note that for this specific problem, we do not want to predict the behavior of a workload itself but of a workload on a different hardware configurations. Thus, we model 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 that the relationship between an independent variable, the SKU, and a response variable, the performance metric in this case, is linear.

Regression [40]. Regression analysis is the task of modeling the relationship between a dependent variable and one or more independent variables. It is frequently used for forecasting and prediction, where data trends are used to forecast future points. Regression models come in a variety of types, such as linear, polynomial, and lasso, to accommodate different data patterns, allowing for a flexible approach to describing the associations within the dataset.

Linear Mixed Effect Model (LMM) [3]. 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. They are particularly useful for data that is collected in groups, where observations are not independent of each other within the same group.

Non-Linear Models. Non-linear models are commonly used if there is no continuous arc across the data. They allow us to focus



(a) Single model. **(b) Pairwise models, each model showing experiments run at a different time of day.**

Figure 7: Comparison of scaling model approaches using LMM as the modeling strategy and TPC-C as the observed workload on varying hardware configurations. The shaded region is the Confidence Interval of model prediction.

on the relationship between pairs of data points to model more hardware-specific trends across within the same (type of) workload.

Multivariate Adaptive Regression Splines (MARS) [26]. MARS models non-linear relationships by partitioning the predictor space into regions and fitting separate linear regressions within each one. It automatically selects variables and interaction terms, providing a piece-wise linear fit to the data.

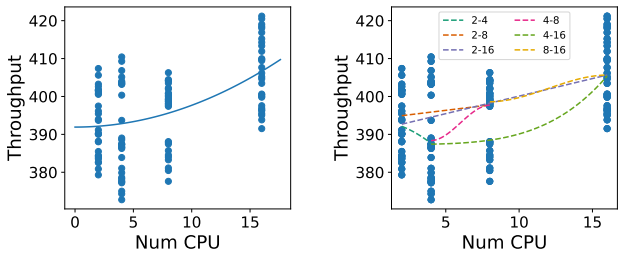
Support Vector Machine (SVM) [16, 41, 60]. SVM is a model that performs well in both classification and regression tasks. In the context of database scaling modeling, SVM is effective in modeling non-linear relationships between various SKU attributes (like CPU, memory, and storage) and our target performance metric.

Gradient Boosting (GB) [27, 28]. Gradient Boosting is a technique that 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. By giving more weight to the data points that were incorrectly predicted in the previous iteration, the model gets better at capturing complex patterns in the data.

5.2 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. Furthermore, we use resampling of our measurements as a data augmentation strategy [52]. Specifically, 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 mean normalized root mean square error (NRMSE), where we normalize against the true throughput values: $NRMSE = \frac{\sqrt{\frac{1}{n}\sum_i(\hat{y}_i - y_i)^2}}{y_{max} - y_{min}}$

5.2.1 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



(a) Single model **(b) Pairwise model**

Figure 8: Comparison of scaling model approaches using SVM as the modeling strategy and TPC-C as the observed workload on varying hardware configurations.

model, LMM, in both single and pairwise scaling scenarios Figure 7. Recall that to capture our experiments, we run each workload at different times of the day multiple times, marked in the figures with different colors. In Figure 7b, 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 7a). This observation is not unique to linear models but we observe similar behavioral characteristics in non-linear scaling models as shown in Figure 8.

Insight 1

Although a single model can capture the overall trend of the data, accurate predictions for the transition between specific hardware configurations can be modeled more accurately using pairwise scaling models.

5.2.2 Modeling Strategies After showcasing the impact of the modeling context, we next 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

Table 6: Model NRMSE for each configuration with varying number of concurrent terminals (subscript in workload name).

Strategy		Mean Training Time (s)	Mean Test NRMSE for 5-Fold Cross Validation							
			$TPC\text{-}C_4$	$TPC\text{-}C_8$	$TPC\text{-}C_{32}$	$Twitter_4$	$Twitter_8$	$Twitter_{32}$	$TPC\text{-}H_1$	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
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

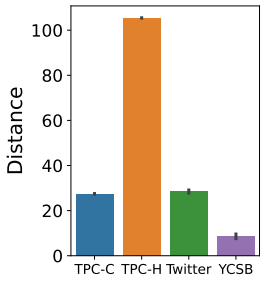


Figure 9: Distance to YCSB experiment run with other workloads.

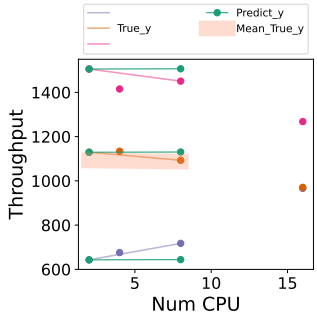


Figure 10: YCSB prediction from SKU with 2 CPUs to SKU with 8 CPUs.

all scaling pairs within an experiment, i.e., the six combinations scaling up between 2, 4, 8, and 16 CPU nodes respectively. Among all learning methods, gradient boosting performs the best with a mean NRMSE of around 0.27 for both contexts , followed closely by pairwise SVM with a mean NRMSE of 0.279 across all evaluated workloads. However, on average, the training time for SVM is 10× to 40× faster than gradient boosting. Across all models, the lowest observed NRMSE for a workload is 0.231 while the highest is 0.356, corresponding to a deviation of ≈ 23% and 37% from the actual observed throughput values respectively. Given our limited scope of experiments, such high variation can be partially attributed to noise within the experiment execution as previously discussed in Section 5.2.1. Nevertheless, we observe that most strategies behave similarly within their model restrictions, i.e., whether a single or pairwise model has been chosen.

Insight 2

The choice of model context is comparatively more impactful on the prediction performance than the choice of modeling strategy.

5.2.3 End-to-End Prediction To test how well our end-to-end prediction framework works for unknown workloads, we set up a final experiment using YCSB [15] 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 4.1, 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 fANOVA. To compute the similarity between the new and existing workload types, we use the L21-norm and calculate the distances as shown in Figure 9 noting that YCSB is most similar to TPC-C, closely followed by Twitter.

We then use a pairwise model in combination with SVM as reference for the scaling behavior from 2 CPUs to 8 CPUs for our previously observed TPC-C experiments. Figure 10 presents the comparison of the actual measurements for the experiments as well as the prediction based on TPC-C. We first observe that the ground truth has some performance variation due to noise on the virtual machines shown in the variation of the measurements at different times of data, 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 0.0787 for this experiment.

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 experiments have focused on the execution of workloads on the same hardware configuration, only varying the CPU nodes. Thus, we believe that 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 for our limited experimentation space, the scaling strategies differed only minimally from each other in terms of their achieved prediction error with strategies such as gradient boosting and SVM leading the way. For all of the applied strategies, 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 a new workload

and showcased how the framework would function in practice, highlighting its advantages and disadvantages.

6 Discussion

Next, we summarize our insights from our experimental evaluations, and outline future work that merits a deeper 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 for all 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 a) clustering algorithms are highly sensitive to which features are used for similarity computation, and that b) not all strategies perform well along our evaluation dimensions of reliability, discrimination power, and robustness. Overall, we have found that norm-based algorithms generally outperform their alternatives 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 scaling prediction, we have focused on a relatively simple use case, i.e., scaling between SKUs that only vary in the number of CPU cores. Even for this use case, we have observed scaling to be non-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 settings. This is substantiated in our experiments, where the choice of the model context dominates the 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 scaling predictions can guide and enhance future research in this area. As far as we are aware, this is the first study that methodically examines the impact of feature selection as well as workload similarity computation on workload prediction tasks. 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.

7 Related Work

Resource prediction and allocation is a well studied topic in several communities. Several works [20, 53, 58, 68] have looked at resource scaling as the database query workload changes by using specific properties from telemetry. Use of ML based techniques (such as building decision trees over certain features) for resource management [17, 31, 45] is also quite popular. We refer the reader to [47] for an in-depth overview of feature selection algorithms (including the methods considered in our work).

There are also works targeting workload characterization, mostly for query level tasks. Those works uses statistical methods [33], or ML methods like Auto-Encoders [49] and Hidden Markov Model [38]. Calzarossa et al. did comprehensive survey on techniques for workload characterization [8, 9]. Workload performance or resource usage prediction by studying existing workload falls into two genres: single model prediction [14, 54] or a collection of models [2, 11, 18, 39]. They employ the resource usage time-series data to build the models, and select model(s) based on cost model [54], largest value [2], or time-series similarity [14].

Many recent works learn on historical data and model the workload using CNNs [23], Markov Chain based model [19, 44], LSTM [5, 14, 29, 37, 56, 61], Gated Recurrent Unit [30, 51], Autoregressive Integrated Moving Average [7], etc.. However, a recent paper post question on whether ML techniques, especially LSTM, is as effective in these prediction tasks as they appears to be [13]. Another subset of research comes from a monetary perspective, involving pricing of services on cloud platforms, and predict the best deployment strategy for given workloads [1, 10, 65]. There are also works focus specifically on large workload under distributed settings, or grid systems [18, 19].

Beyond ML, [24] described 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 [57], who proposed Supercloud, an architecture for migrating VMs. VM consolidation has also been studied in the context of IoT applications [48].

8 Conclusion

In this work, we have conducted a thorough analysis of a common ML-based end-to-end pipeline for workload scaling prediction. We 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). We propose a series of recommendations based on our experimental observations and discuss open questions that are likely to improve the prediction quality of ML-based resource prediction pipelines.

References

[1] Mohammad Aldossary, Karim Djemame, Ibrahim Alzamil, Alexandros Kostopoulos, Antonis Dimakis, and Eleni Agiatzidou. 2019. Energy-aware cost prediction and pricing of virtual machines in cloud computing environments. *Future Generation Computer Systems* 93 (2019), 442–459. https://doi.org/10.1016/j.future.2018.10.027

[2] 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

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

[4] 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

[5] 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

[6] 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

[7] 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

[8] M. Calzarossa and G. Serazzi. 1993. Workload characterization: a survey. *Proc. IEEE* 81, 8 (1993), 1136–1150. https://doi.org/10.1109/5.236191

[9] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. 2016. Workload Characterization: A Survey Revisited. *Comput. Surveys* 48, 3 (Feb. 2016), 1–43. https://doi.org/10.1145/2856127

[10] Dawei Chen, Xiaoqin Zhang, Li Wang, and Zhu Han. 2021. Prediction of Cloud Resources Demand Based on Hierarchical Pythagorean Fuzzy Deep Neural Network. *IEEE Transactions on Services Computing* 14, 6 (Nov. 2021), 1890–1901. https://doi.org/10.1109/tsc.2019.2906901

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

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

[17] 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.

[18] 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

[19] 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.

[20] 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.

[21] 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

[22] 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

[23] 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

[24] 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.

[25] Syed Masum Emran and Nong Ye. 2002. Robustness of Chi-square and Canberra distance metrics for computer intrusion detection. *Quality and Reliability Engineering International* 18, 1 (jan 2002), 19–28. https://doi.org/10.1002/qre.441

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

[27] 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

[28] 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

[29] 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

[30] 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

[31] 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.

[32] 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

[33] 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]

[34] 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

[35] 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.

[36] 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

[37] 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

[38] 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

[39] 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

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

[41] 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.

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

[43] 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

[44] 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

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

[46] 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

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

[48] 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.

[49] Debjyoti Paul, Jie Cao, Feifei Li, and Vivek Srikumar. 2021. Database workload characterization with query plan encoders. *Proceedings of the VLDB Endowment* 15, 4 (Dec. 2021), 923–935. https://doi.org/10.14778/3503585.3503600

[50] 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.

[51] 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

[52] Dimitris N. Politis. 2003. The Impact of Bootstrap Methods on Time Series Analysis. *Statist. Sci.* 18, 2 (May 2003). https://doi.org/10.1214/ss/1063994977

[53] 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.

[54] 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

[55] 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

[56] 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

[57] 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.

[58] 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.

[59] 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

[60] 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

[61] 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

[62] 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

[63] 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

[64] 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

[65] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. 2013. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers* 16, 1 (Oct. 2013), 7–18. https://doi.org/10.1007/s10796-013-9459-0

[66] Nong Ye and Qiang Chen. 2001. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International* 17, 2 (2001), 105–112. https://doi.org/10.1002/qre.392

[67] 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

[68] 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.

[69] 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, [33], that uses change-point detection algorithms, such as the Bayesian change point detection (BCPD) technique [42], 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

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.