applied sciences

MDPI

Article

Fast Performance Modeling across Different Database Versions Using Partitioned Co-Kriging

Rong Cao *, Liang Bao, Shouxin Wei, Jiarui Duan, Xi Wu, Yeye Du and Ren Sun

School of Computer Science and Technology, Xidian University, Xi'an 710071, China; baoliang@mail.xidian.edu.cn (L.B.); imshouxin@gmail.com (S.W.); jrdapril@163.com (J.D.); xiwu@stu.xidian.edu.cn (X.W.); 13700293619@163.com (Y.D.); laura1628@163.com (R.S.)
* Correspondence: 18700939602@163.com

Abstract: Database systems have a large number of configuration parameters that control functional and non-functional properties (e.g., performance and cost). Different configurations may lead to different performance values. To understand and predict the effect of configuration parameters on system performance, several learning-based strategies have been recently proposed. However, existing approaches usually assume a fixed database version such that learning has to be repeated once the database version changes. Repeating measurement and learning for each version is expensive and often practically infeasible. Instead, we propose the Partitioned Co-Kriging (PCK) approach that transfers knowledge from an older database version (source domain) to learn a reliable performance prediction model fast for a newer database version (target domain). Our method is based on the key observations that performance responses typically exhibit similarities across different database versions. We conducted extensive experiments under 5 different database systems with different versions to demonstrate the superiority of PCK. Experimental results show that PCK outperforms six state-of-the-art baseline algorithms in terms of prediction accuracy and measurement effort.

Keywords: database; version; performance modeling; co-kriging; transfer

Citation: Cao, R.; Bao, L.; Wei, S.; Duan J.; Wu X.; Du Y.; Sun, R. Fast Performance Modeling across Different Database Versions Using Partitioned Co-Kriging. *Appl. Sci.* **2021**, *11*, 9669. https://doi.org/10.3390/app11209669

Academic Editor: Arcangelo Castiglione

Received: 3 September 2021
Accepted: 13 October 2021
Published: 16 October 2021

1. Introduction

Database systems are increasingly becoming more configurable [1,2]. The large number of configuration parameters can directly influence the functional and non-functional properties of database systems [3,4]. Performance (e.g., latency, throughput, and requests per second) is one of the most important non-functional properties as it directly affects user experience [5]. Appropriate configurations can improve the performance for database systems [2,6]. For example, the throughput difference between the best and worst configurations for Cassandra can be as high as 102.5% for a given workload [7]. To distinguish the optimal configuration, users are interested in knowing the consequences of changing the configuration parameters that are available to them. However, the exponentially growing configuration space and complex interactions among configuration parameters make it difficult to understand the performance of the system [8,9].

In recent years, learning-based approaches have become the mainstream to solve this problem. These methods often collect the performance measurements of only a limited set of configurations (called samples), then build a performance model with these samples, and use the model to predict the system performance of the unseen configurations [7,10–14]. In this way, performance can be predicted before a variant of the database system is configured and deployed.

Existing approaches usually focus on the performance modeling problem under a constant environment, including fixed hardware [11,14], workload [9,10], and database version [2,7]. We will concentrate on performance modeling for a newer database version in this work. In practice, previous performance prediction models often lead to high prediction error once the database version changes [15]. A new performance prediction

model may need to be learned from scratch to meet the prediction accuracy requirements. However, the demand for high quality samples in learning-based methods contradicts the need for rapid acquisition of a new performance prediction model. Specifically, researchers often need a large number of samples to build a good performance prediction model [16]. However, the collection of samples requires a lot of time and resources [6,10], which makes the construction of the new performance model time-consuming and laborious.

Fortunately, performance models typically exhibit similarities across different database versions [17]. We introduce the concept of transfer learning into performance modeling for database systems. Similar to humans that learn from previous experience and transfer the knowledge learned to accomplish new tasks easier [18], here, knowledge about performance behavior observed in an older database can be reused effectively to learn models for a newer database with a lower cost. The problem is to identify the transferable knowledge and make use of them to ease learning of performance models.

In this paper, we propose a co-kriging-based performance prediction method to efficiently learn models by reusing information gained previously when database version changes. The challenge is to predict system performance with high accuracy while utilizing a small sample in target domain. As it takes time and effort to configure the database system and collect performance measurement data, it is desirable that the sample size is kept minimum. Co-kriging allows data on an auxiliary variable to be used to make up for an insufficient amount of data in undersampled case [19]. We regard the performance responses in older (source) and newer (target) database systems as the auxiliary variable and primary variable, respectively. The measurement data in source can facilitate building an accurate performance model in target by co-kriging with a small sample in target.

Further, Partitioned Co-Kriging (PCK) is proposed to better satisfy the assumption of co-kriging method. The assumption of co-kriging method is that the performance response is stationary [20]. The accuracy of performance model can increase obviously by applying co-kriging in several regions which performance responses are smooth. The partition of these smooth regions is regarded as transferable knowledge, and it can be obtained by clustering the measurement data in source domain.

In a nutshell, we regard the partition of smooth regions and the measurement data in source as transferable knowledge. Furthermore, the PCK is presented to make use of the transferable knowledge to fast construct performance models in target. Finally, we demonstrate that our PCK approach will enable accurate performance predictions by using only a small number of measurements in target domain. We evaluate our approach on five different database systems: MySQL, PostgreSQL, SQLite, Redis, and Cassandra.

In summary, our work makes the following contributions.

- We perform a proof of concept that transferring the knowledge across different database versions using PCK can facilitate the fast performance modeling for a newer database version.
- We verify the feasibility and validity of PCK through extensive experiments under different categories of database systems with different versions. Experimental results show that our proposal outperforms six state-of-the-art baseline algorithms by 30.73–60.83% on average.

2. Related Work

Researchers make an effort to understand the relationship of parameters and performance. Several performance prediction models [3,5,10–14,21–23] and tuning strategies [6,16,24–31] have been proposed to explore the relationship and recommend good configurations further.

Performance Prediction for Databases. The performance prediction methods fall into two major categories: analytical prediction models and learning-based prediction models. The first class of approaches require an in-depth analysis of the constraints on system performance, and the analysis models are given accordingly [24–26]. The latter class of approaches train prediction models using machine learning techniques,

including GP regression [2,6,27,32], neural networks [7,10,28,29], CART [3,5,11–14], Fourier Learning [21,22], and so on.

Existing approaches usually assume a fixed environment such that the modeling/tuning process has to start from the scratch once the environment changes, which exacerbates the performance modeling problem. Therefore, transferring knowledge across environments to assist the modeling/tuning task has become a hot area of research in recent years.

Knowledge Transfer for Performance Prediction. The most relevant research to this paper is to transfer performance prediction models across environments.

To cope with the workload changes, OtterTune [2,32] reuses past experience to reduce the tuning cost for a new application. Rafiki [7] includes workload characteristics and the most impactful parameters directly in its surrogate model.

Valov et al. [33] analyzed different hardware platforms to understand the similarity on performance prediction. A simple linear regression model is used to transfer knowledge from a related environment to a target environment. Differently, there is another kind of approach that reuses source data with the hope to capture correlation between environments using learners such as GP Models [34].

Further, it is worth exploring why and when transfer learning works for performance modeling. Jamshidi et al. [17] combine many statistical and machine learning techniques to study this research question. Javidian et al. [35] exploit causal analysis to identify the key knowledge pieces that can be exploited for transfer learning.

Jamshidi et al. [15] propose a sampling strategy L2S, which is inspired by the research results in [17]. L2S extracts transferable knowledge from the source to drive the selection of more informative samples in the target environment.

Moreover, transfer learning can only be useful in cases where the source environment is similar to the target environment. BEETLE [36] focuses on the problem of whence to learn. A racing algorithm is applied to sequentially evaluate candidate environments to discover which of the available environments are best suited to be a source environment.

In this paper, we concentrate on transferring knowledge across different database versions. There are few relevant researches on this issue at present. The studies on version change scenario is the least among the three main environment change scenarios. The exploratory analysis [17] and causal analysis [35] give us insights into performance prediction across environment change, but no transfer scheme is given in these studies. BEETLE [36] places emphasis on identifying suitable sources to construct transfer learner. Other work [15,34] has made some discussion about the version change scenario, but they have not conducted in-depth research and experimental verification on this issue.

3. Problem Overview

3.1. Problem Statement

In this paper, we focus on building an accurate performance model rapidly with a small amount of samples for a database system in a newer database version. For a given database and a specific workload, our object is to enable the reuse of previous performance measurements in an older database version to facilitate the performance prediction in a newer version. In order to formalize this problem concisely, we introduce some mathematical symbols to represent related concepts.

Configuration. A configuration of a database system is represented as a set $x = (c_1, c_2, \cdots, c_n)$, where c_i indicates the i-th configuration parameter of the database system and n is the number of configuration parameters. The configuration parameters can either be (1) an integer variable in the valid configuration bound of the parameter, (2) a categorical variable or a Boolean variable. The configuration space is a Cartesian product of all configuration parameters $\mathbb{X} = Dom(c_1) \times \cdots \times Dom(c_n)$, where $Dom(c_i)$ is the valid range of each parameter.

Performance model. Performance (e.g., throughput or latency) is an essential non-functional property of database systems. Different configurations may lead to different performance values. We treat the performance as a black-box function, which describes

how configuration parameters and their interactions influence the system performance. Given a database system A and a workload W with configuration space $\mathbb{X}$, a performance model is a black-box function $f : \mathbb{X} \to \mathbb{R}$ that maps each configuration $x \in \mathbb{X}$ of A to the performance of the system.

Sample. Due to the large configuration space and expensive measurement cost, researchers usually propose to measure the performance values of a limited number of configurations (called sample), then construct a performance model from these data to predict the performance values of any new configuration. We run a database A in a certain version $v \in V$ on various configurations $x_i \in \mathbb{X}$, and record the resulting performance values y_i. The training data for learning a performance model for system A with version v are $D_{tr} = (x_i, y_i),\ i = 1, 2, \cdots, m$, where m is the number of performance measurements. The database in an older version is called source domain, while the database in a newer version is called target domain. S_S and S_T indicate the samples in source and target, respectively.

The object is to learn a reliable performance model $\hat{f}(x)$ in target domain. Specifically, we aim to minimize the prediction error over the configuration space:

$$\arg\min_{x \in \mathbb{X}} pe = |\hat{f}(x) - f(x)|. \tag{1}$$

The difficulty of this problem is to use only a small sample while still be able to predict the performance of other unseen configurations with a high accuracy. In this paper, we import the measurements in source domain to help constructing a reliable performance prediction model in target domain. Thus, the inputs to this problem are a large number of samples from the source and a small number of samples from the target, while the output is a reliable performance model in target, as described in Figure 1.

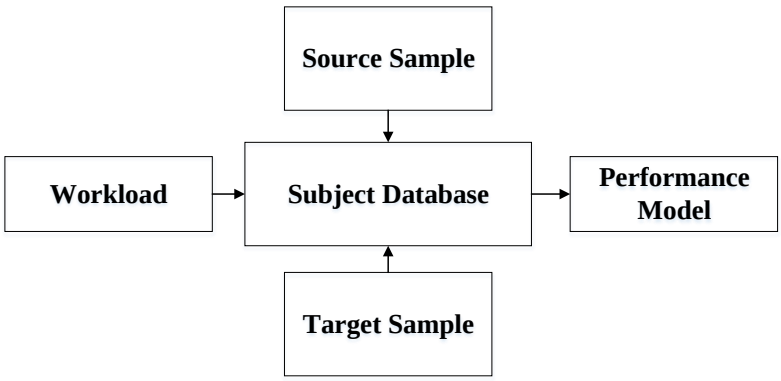

Figure 1. Problem overview of performance modeling across different database versions.

3.2. Key Observations

Generally, performance values are expected to differ in different database versions for a given configuration. Fortunately, performance models typically exhibit similarities across different database versions in the configuration space. An experiment is carried out to verify this observation, and the results are shown in Figure 2. We randomly select 100 valid configurations in MySQL, and obtain the performance measurements in both 5.5 and 8.0 versions, which are indicated by source and target, respectively. On the bases of the experimental results and some previous researches [17,34], we accept that the performance responses in source domain and target domain are similar in configuration space when database version changes.

The second key observation is that the performance response of a database system is reasonably smooth within a certain range, and it changes dramatically when some key parameters change. Intuitively, we expect that for nearby input points x_1 and x_2 in the smooth range, their corresponding output points y_1 and y_2 to be near-by as well. However,

on the contrary, the performance responses of different regions vary greatly. The similar observations have been found in former studies [2,8].

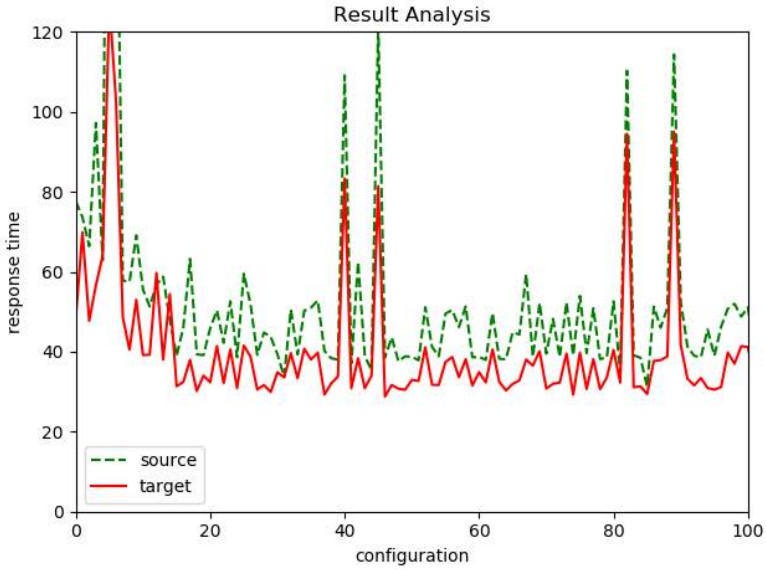

Figure 2. Performance models exhibit similarities across different database versions.

3.3. Assumptions and Limitations

In reality, the correlations are quite strong in some version change situations, while for others, the correlations are extremely weak or even nonexistent. We assume that the strength of the correlation is related to the details of upgrades between versions. In some version update cases, the optimization features that are determined by the configuration options may undergo a substantial revision between different versions because algorithmic changes may significantly improve the way how the optimization features work. Nonetheless, some version updates do not influence the internal logic controlled by configuration parameters, thus the correlation remain strong in these cases. We focus on the latter case because that is the majority in database version update cases.

In addition, the number of configuration parameters changes as the database version updates. Some parameters are added and some are forbidden in the upgraded database version. We currently focus on the unchanged parameter from version to version for simplicity, and defer the problem of taking parameter change into consideration when transferring knowledge across different database versions as future work.

4. Partitioned Co-Kriging Based Performance Prediction

4.1. Overview

Our goal is to construct an accurate performance prediction model using a small sample in target domain. To overcome the challenge of insufficient sample, we reuse prior information that we can get from source domain to learn a performance model more efficiently. The foundation of our method is that performance responses in source and target have high correlations. Thus, the large samples in source can be applied to build a performance model in target. The performance prediction is realized by the co-kriging method.

The assumption of co-kriging method is that the performance response is stationary. In practice, the performance response of database is not stationary in the entire configuration space, but it is relatively smooth in different regions. Based on this characteristic, we propose to learn performance prediction models within each region. Therefore, we first need to solve the partition problem: how to divide the entire configuration space into several smooth regions?

Generally, there are only sparse samples in target due to the high cost in collecting the samples. The small samples provide too little information to perform the partitioning task. In contrast, the sample in source is usually readily available. As a consequence, we use the concept of relatedness between source and target to address the problem. The partition of configuration space can be achieved by clustering with the adequate samples from source domain. The prediction accuracy can be improved effectively by using this partition scheme.

Figure 3 illustrates the performance prediction process for unseen configurations in target. The input of PCK method incorporates performance measurements from both source and target. We use random sampling method to generate a small number of samples in target, while a large number of samples are available in source. According to the database performance characteristics, we consider dividing the entire configuration space into several smooth regions by clustering the sufficient performance measurements in source. With the region information of source samples, each target sample can determines its region by Euclidean distance. A target sample adopt the region information of a source sample which has the nearest Euclidean distance with it. Subsequently, we can learn corresponding performance models in different regions using all the samples and the partition information in configuration space. Finally, we can estimate the performance value for any unseen configuration with this performance prediction model.

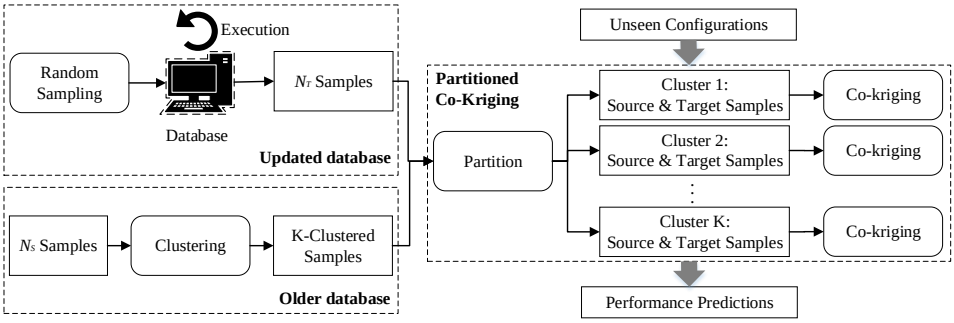

Figure 3. Overview of partitioned co-kriging approach.

In summary, we propose an innovative way of fast performance modeling in updated databases (newer database versions) using co-kriging. It makes use of the large number of source samples in two ways: (1) the abundant and easily accessible source samples can be reused to facilitate the construction of an accurate performance model, and (2) they can further provide partition information about the smooth region in configuration space due to the correlation of source and target, thus improve the prediction accuracy of the co-kriging-based prediction model.

4.2. Performance Prediction with Partitioned Co-Kriging

Using the above-mentioned prediction process, we now discuss the PCK method in Algorithm 1.

Random sampling. We collect training data in target domain by running a subject database with updated version on various configurations and record the resulting performance values. The selected configurations are generated by random sampling (RS) strategy, as it can effectively search for a large configuration space, especially when configuration parameters are not equally important [37] (line 1).

Clustering. The partition of configuration space is achieved by clustering the adequate source samples (line 2). We use a well-studied technique, called k-means [38], to cluster the source samples into meaningful groups. The goal of the k-means algorithm is to divide M points in N dimensions into K clusters so that the within-cluster sum of squares is minimized. M is the number of the source samples. N equals to the number of configuration

parameters add one; it means that we consider not only each configuration parameter, but also its performance value in clustering. This is because we aim to identify the different smooth regions. K clusters correspond to K different regions.

Algorithm 1 $PCK(A, W, P, CB, S_S, K, VM, C_U)$.

Input: A: the subject database; W: workload; P: configuration parameters; CB: configuration bounds; S_S: the available samples in source domain; K: the number of clusters; VM: variogram model; C_U: the unseen configurations in target domain.
Output: P_U: performance predictions of the unseen configurations in target domain.
1: $S_T \leftarrow RS(N_T, A, W, P, CB)$;
2: K-clustered $S_S \leftarrow k - means(S_S, K)$;
3: K-clustered $S_T \leftarrow partition(S_T, K\text{-clustered } S_S)$;
4: K-clustered $C_U \leftarrow partition(C_U, K\text{-clustered } S_S)$;
5: **for** each $c_i \in C_U$ **do**
6: $\quad p_i \leftarrow co - kriging(S_S, S_T, c_i, VM)$ in corresponding cluster;
7: **end for**
8: **return** P_U;

One of the drawbacks of using k-means is that we have to specify the number of clusters (K) before starting. On the one hand, the accuracy of prediction models can be increased by clustering different smooth regions. On the other hand, K should not be selected too large because of the small sample in target. Otherwise, the sample size in each region will be too small to learn a reliable prediction model. In conclusion, choosing K is a trade-off, and we are going to explore this problem in our experiments.

Partition. After the clustering is complete, we get K smooth regions in configuration space. Then, K performance prediction models can be constructed using co-kriging. The input of co-kriging includes not only the source samples in the cluster, but also the target samples in corresponding region. In target domain, the region of a sample is determined by the Euclidean distances with source samples. In order to facilitate the co-kriging-based performance prediction in next step, the samples in both source and target are partitioned using the clustering results (line 3–4).

Co-kriging. Kriging, a regression method, is a minimum variance unbiased linear estimator and has received wide use for ore reserve estimation applications in mining [20]. Kriging utilizes the spatial correlation, of the variable of interest with itself, to determine the weights in an optimal manner. Co-kriging is the logical extension of ordinary kriging to situations where two or more variables are spatially interdependent and the one whose values are to be estimated is not sampled as intensively as the others with which it is correlated [19]. Its advantage is that when the primary variable is difficult or expensive to obtain, the co-kriging method adopts the auxiliary variable that is easier to obtain and is correlated with the primary variable to predict the primary variable, thus improving the prediction accuracy (line 6).

When the auxiliary variable is readily available and changes smoothly, such variable can be introduced into the co-kriging method as an auxiliary influencing factor. In this work, we employ co-kriging method to predict the performance of new configuration in an updated database version (target). Introducing the performance response in an older database version (source) as an auxiliary variable is conducive to the prediction result. In the application of co-kriging, a first step is to model the variogram for each variable as well as a cross-variogram for the two variables. Under the second-order stationary hypothesis, its expectation is

$$E[Z_2(x)] = m_2. \tag{2}$$

The cross-variogram is

$$\gamma_{12}(h) = E[Z_1(x+h) - Z_1(x)][Z_2(x+h) - Z_2(x)]. \tag{3}$$

Thus, the interpolation formula of co-kriging method is

$$Z_2^*(x_0) = \sum_{i=1}^{N_1} \lambda_{1i} Z_1(x_{1i}) + \sum_{j=1}^{N_2} \lambda_{2j} Z_2(x_{2j}). \tag{4}$$

where $Z_2^*(x_0)$ is the performance prediction value of configuration x_0 in target; $Z_2(x_{2j})$ indicates the performance measurement of each configuration in target; λ_{2j} is the weight coefficient for performance measurement of each configuration in target; $Z_1(x_{1i})$ is the performance measurement of each configuration in source; λ_{1i} is the weight coefficient for performance measurement of each configuration in source; N_1, N_2 refer to the sample size of source and target, respectively; and $N_1 > N_2$.

Two Lagrange multipliers u_1 and u_2 are introduced for derivation:

$$\begin{gathered}
\sum_{i=1}^{N_1} \lambda_{1i}\gamma_{11}(x_{1i}-x_I)+\sum_{j=1}^{N_2} \lambda_{2j}\gamma_{21}(x_{2j}-x_I)+u_1=\gamma_{21}(x_0-x_I),\\
\sum_{i=1}^{N_1} \lambda_{1i}\gamma_{21}(x_{1i}-x_J)+\sum_{j=1}^{N_2} \lambda_{2j}\gamma_{22}(x_{2j}-x_J)+u_2=\gamma_{22}(x_0-x_J),\\
i = 1,2,\cdots,N_1, \qquad j = 1,2,\cdots,N_2,\\
\sum_{i=1}^{N_1} \lambda_{1i} = 0, \qquad \sum_{j=1}^{N_2} \lambda_{2j} = 1.
\end{gathered} \tag{5}$$

where γ_{11} and γ_{22} are the variograms of Z_1 and Z_2, separately. γ_{12} and γ_{21} are the cross-variograms for the two variables, and $\gamma_{12} = \gamma_{21}$.

By solving linear Equation (5), the weight coefficients ($\lambda_{1i}, i = 1,2,\cdots,N_1; \lambda_{2j}$, $j = 1,2,\cdots,N_2$) and two Lagrange multipliers u_1 and u_2 can be obtained. Thus, the performance estimation of any configuration in the configuration space can be acquired from Equation (4).

Performance prediction. Given any unseen configuration in target, we first identify the region where it belongs according to the clustering results. The performance of the configuration is then predicted by co-kriging method using the samples and variogram model in this region, as described in Algorithm 1 (lines 5–8).

5. Experimental Evaluation

We have implemented the PCK and other algorithms, and conducted extensive experiments in diverse databases. The source code and the data can be found in the online repository: https://github.com/xdbdilab/PCK. In this section, we first describe our experiment setup, and then present the experimental results to prove the efficiency and effectiveness of the proposed approach.

5.1. Experimental Settings

Subject databases, versions, and benchmarks. We carried out an investigation into the subject systems of researches on configuration tuning for database systems [2,6–10,14,16,28,29,31,39–44], and chose five widely used database systems to evaluate PCK approach: MySQL (https://www.mysql.com/ (accessed on 25 September 2020)), PostgreSQL (https://www.postgresql.org/ (accessed on 26 September 2021)),SQLite (https://www.sqlite.org/ (accessed on 26 September 2021)), Redis (https://redis.io/ (accessed on 11 October 2020)), and Cassandra (http://cassandra.apache.org/ (accessed on 13 October 2020)). MySQL is an open-source relational database management system (RDBMS). PostgreSQL is an open-source object-relational database management system (ORDBMS). SQLite is an open-source embedded RDBMS. Redis is an open-source in-memory data structure store. Cassandra is an open-source column-oriented NoSQL database management system. In this experiment, we choose two (PostgreSQL, SQLite, and Cassandra) or three (MySQL,

Redis) representative versions for convenience. The time gaps among these releases of three database are about five years for PostgreSQL, nearly a half year for SQLite, about five and a half years for Cassandra, and approximate three years for MySQL and Redis. In addition, we use sysbench (https://github.com/akopytov/sysbench (accessed on 25 September 2020)) for MySQL, pgbench (https://www.postgresql.org/docs/11/pgbench.html (accessed on 26 September 2021)) for PostgreSQL, a customized workload for SQLite, YCSB [45] for Cassandra, and Redis-Bench (https://redis.io/topics/benchmarks (accessed on 11 October 2020)) for Redis.

Parameters. For each database system, we use domain expertise to identify a subset of parameters that are considered critical to the performance, as in [2,9,10]. Reducing the number of considered parameters can reduce the search space exponentially, and numerous existing approaches [9,12] also adopt this manual feature selection strategy. Note that even with only these parameters, the search space is still enormous, and exhaustive search is infeasible. Table 1 summarizes the database systems and versions, along with the benchmarks, the numbers of selected parameters, and performance metrics, respectively.

Table 1. Subject databases and versions, benchmarks, parameters, and performance metrics.

Subject Database	Category	Subject Versions	Benchmark	# of Selected Parameters	Performance
MySQL	RDBMS	5.5, 5.7, 8.0	sysbench	10	Latency (ms)
PostgreSQL	ORDBMS	9.3, 11.0	pgbench	9	Transactions per second
SQLite	Embedded DB	3.31.1, 3.36.0	Customized	8	Transactions per second
Redis	In-memory DB	4.0.1, 5.0.0, 6.0.5	Redis-Bench	9	Requests per second
Cassandra	NoSQL DB	2.1.0, 3.11.6	YCSB	28	Throughput (MB/s)

Running environment. In order to avoid interference in collecting samples from different subject database systems, we conduct experiments on different servers and computer. In addition, we ensure a consistent running environment for different versions of the same subject system. The running environments for different subject databases systems are listed as follows.

MySQL, PostgreSQL: The physical server is equipped with two 2-core Intel(R) Core(TM) i5-4590 CPU @3.30GHZ processors, 4GB RAM, 64GB disk, and running CentOS 6.5 and Java 1.8.0.

SQLite: The computer is equipped with a Intel(R) Core(TM) i5-4460 CPU @3.20GHZ processors, 8GB RAM, 1TB disk, and running Windows 10 and Java 1.8.0_291.

Redis: The cloud server is equipped with two 2-core Intel(R) Xeon(R) Platinum 8163 CPU @2.50GHz processors, 4GB RAM, 53.7GB disk, and running CentOS 7.6 and Java 1.8.0_261.

Cassandra: The physical server is equipped with two 4-core Intel(R) Xeon(R) CPU E5-2683 V3 @2.00GHz processors, 32GB RAM, 86GB disk, and running CentOS 6.5 and Java 1.8.0_211.

Baseline Algorithms. To evaluate the performance of PCK approach, we compare it with six state-of-the-art algorithms: DeepPerf [10], CART [3], Finetune [46], DDC [47], Model-shift [33], and Ottertune [2]. We provide a brief description for each algorithm as follows.

DeepPerf and CART establish performance prediction models in target domain directly. They consider the performance prediction problem as a nonlinear regression problem and apply different machine learning methods, namely Deep Neural Network (DNN) and the Classification and Regression Trees (CART) technique, to find this nonlinear model.

Finetune and DDC are two widely used transfer learning schemes. Finetune is a network-based transfer learning method. It freezes the partial network that pre-trained in the source domain, and transfers it to be a part of DNN which used in target domain. DDC is a mapping-based transfer learning, which maps instances from the source and target into a new data space.

Model-shift approach shifts the model that has been learned in the source to predict the system performance in the target using linear regression models. CART is applied to build performance prediction models in source domain.

Ottertune is a transfer learning approach that exploits source samples to learn a performance model in the target. The Gaussian Process (GP) model is used to learn a performance model that can predict unobserved response values.

5.2. Evaluation of Prediction Accuracy

Data collection. We use the random sampling strategy to generate a set of configurations for each database and test them on the database system with given workloads on different versions to get the performance measurements. A configuration–performance pair is regarded as a sample. The numbers of samples for MySQL, PostgreSQL, SQLite, Redis, and Cassandra are 294, 300, 280, 323, and 1129, respectively. The collection of all source samples serves as auxiliary training data. In the target domain, a subset of these samples is selected randomly for training dataset; the remaining samples serve as the testing dataset.

Evaluation metric. Holdout validation is employed to compare the prediction accuracy between different methods. We use the training dataset and auxiliary training data to generate a performance model for each method, and then use this model to predict performance values of configurations in the testing dataset. We select *Mean Relative Error (MRE)* as a metric for evaluating prediction accuracy, which is computed as follows:

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\frac{|a_i - p_i|}{a_i} \times 100 \quad (6)$$

where N is a total number of configurations in the testing dataset, and a_i and p_i represent the actual performance value and predicted performance value, respectively. We choose this metric as it is widely used to measure the accuracy of prediction models [5,10,33,34].

Performance results. We run PCK method and six baseline algorithms for five subject database systems independently. The size of training dataset in target domain is set to $c \times n$, where c is the number of selected configuration parameters for each subject database system (which is shown in the column # of selected parameters of Table 1), and n ranges from 1 to 15. To evaluate the consistency and stability of the approaches, for each sample size of each subject database system, we repeat the random sampling, training and testing process 5 times. We then show and compare the mean of the MREs obtained with the 7 different approaches for each sample size. The experiment results of five subject database systems for larger version changes are listed in Tables 2–6, respectively.

As expected, our proposed method has achieved better performance than all other six algorithms. Specifically, for five subject database systems, PCK outperforms all other six baseline algorithms: 24.99–53.18% improvement over DeepPerf, 14.20–51.19% improvement over CART, 4.42–46.58% improvement over Finetune, 25.26–53.13% improvement over DDC, 9.34–50.03% improvement over Model-shift, and 46.80–69.42% over Ottertune. The improvement percentages are shown in Figure 4. For the sake of simplicity, the data in Figure 4 is obtained by averaging the MRE data of different sample sizes in each subject database system.

Table 2. MRE comparison among different approaches for MySQL (version 5.5–8.0).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	21.829	24.653	21.446	22.757	23.050	23.234	21.508	21.313	20.849	21.033	15.444	19.558	16.970	16.493	17.168
CART	23.281	22.511	16.529	19.268	18.071	13.550	13.899	12.642	12.086	13.035	12.651	11.144	9.665	12.937	10.359
Finetune	14.532	13.854	13.320	12.640	12.649	13.376	13.610	14.906	14.467	12.527	14.550	13.703	14.050	14.037	13.087
DDC	19.255	25.563	26.350	21.325	21.912	22.096	24.017	18.917	19.936	19.174	19.285	19.935	17.632	15.661	16.233
Model-shift	15.848	16.171	11.314	11.248	12.902	**10.128**	11.342	10.224	10.697	11.064	11.419	10.822	10.217	11.198	11.246
Ottertune	16.846	19.041	18.605	21.737	19.475	19.258	19.293	18.322	19.081	18.945	17.393	19.100	17.991	17.379	16.504
PCK	**12.399**	**12.177**	**10.222**	**10.734**	**10.747**	10.922	**9.618**	**9.268**	**9.645**	**9.273**	**9.303**	**9.230**	**9.183**	**9.280**	**9.197**

Table 3. MRE comparison among different approaches for PostgreSQL (version 9.3–11.0).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	9.637	6.384	4.997	3.835	3.577	3.525	3.209	3.219	3.276	3.12	3.031	2.828	2.816	2.828	2.819
CART	3.19	3.023	3.232	3.128	3.039	3.065	2.839	2.932	2.99	2.905	2.961	2.886	2.868	2.831	2.807
Finetune	2.778	2.767	2.838	2.677	2.725	2.591	2.729	2.763	2.783	**2.522**	2.595	**2.552**	2.621	2.61	2.593
DDC	7.415	5.619	4.507	3.749	3.385	3.382	2.968	2.928	3.095	2.943	2.767	2.756	2.714	2.809	2.757
Model-shift	2.718	2.603	2.645	2.991	2.858	2.932	2.75	2.676	2.902	2.831	2.799	2.674	2.798	2.886	2.745
Ottertune	11.934	9.289	9.373	7.792	6.812	6.431	5.127	4.76	4.486	4.173	3.824	3.574	3.312	3.274	3.217
PCK	**2.634**	**2.616**	**2.593**	**2.590**	**2.606**	**2.585**	**2.568**	**2.576**	**2.572**	2.570	**2.573**	2.568	**2.558**	**2.564**	**2.167**

Table 4. MRE comparison among different approaches for SQLite (version 3.31.1–3.36.0).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	9.109	2.846	2.249	1.945	2.007	2.007	1.857	1.917	1.841	1.799	1.736	1.711	1.606	1.544	1.56
CART	1.692	1.793	1.932	1.686	1.795	1.575	1.623	1.593	1.516	1.632	1.573	1.578	1.535	1.541	1.522
Finetune	2.39	2.271	1.876	1.722	1.801	1.692	1.754	1.66	1.64	1.67	1.539	1.584	1.547	1.517	1.567
DDC	5.452	2.873	2.286	2.058	1.819	1.926	1.91	1.806	1.673	1.713	1.861	1.633	1.692	1.628	1.705
Model-shift	1.593	1.352	1.724	1.471	1.708	1.519	1.61	1.57	1.596	1.565	1.655	1.572	1.598	1.55	1.577
Ottertune	8.804	8.768	8.621	7.754	7.501	6.283	6.271	5.755	5.509	5.161	4.445	3.944	3.664	2.718	2.09
PCK	**1.208**	**1.101**	**1.058**	**1.052**	**1.028**	**1.033**	**1.012**	**1.024**	**1.010**	**1.016**	**1.000**	**0.997**	**1.002**	**1.001**	**0.999**

Table 5. MRE comparison among different approaches for Redis (version 4.0.1–6.0.5).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	10.405	7.598	6.304	5.608	5.873	6.060	5.947	5.482	5.727	5.552	5.341	5.135	5.355	5.187	5.294
CART	6.270	5.499	5.560	5.964	5.763	5.408	5.198	5.384	6.024	5.832	5.876	5.443	5.878	5.776	5.700
Finetune	4.790	5.721	5.227	4.779	5.331	5.007	5.451	4.931	5.150	4.664	4.638	4.780	4.796	4.667	4.666
DDC	9.688	7.430	6.805	5.377	5.847	5.292	5.562	5.711	5.229	5.417	5.549	5.442	5.415	5.248	5.084
Model-shift	4.682	4.643	4.927	5.460	5.314	5.024	5.342	5.061	5.279	5.280	5.217	5.127	5.013	5.110	4.960
Ottertune	31.177	16.059	11.747	11.413	10.210	9.895	10.033	9.929	10.089	9.111	9.168	8.769	8.299	8.222	7.718
PCK	**3.278**	**3.083**	**2.931**	**2.877**	**3.036**	**2.889**	**2.886**	**2.879**	**2.889**	**2.868**	**2.873**	**2.877**	**2.867**	**2.870**	**2.867**

Table 6. MRE comparison among different approaches for Cassandra (version 2.1.0–3.11.6).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	25.608	24.309	24.907	25.054	24.571	25.485	25.249	24.674	24.507	24.125	24.274	24.451	24.114	24.105	24.219
CART	23.452	23.868	23.432	23.187	23.509	23.197	23.718	23.410	23.277	23.236	23.325	23.207	23.668	23.547	23.800
Finetune	22.369	21.408	21.742	22.306	21.454	21.770	20.984	21.282	21.503	21.465	21.113	21.622	21.340	21.554	21.443
DDC	26.090	24.983	24.865	24.938	24.103	24.390	24.333	24.385	24.702	24.362	24.157	24.356	23.959	24.671	24.206
Model-shift	23.307	22.721	22.923	22.66	23.196	23.039	22.883	23.086	22.971	22.860	22.914	22.868	22.912	22.900	22.987
Ottertune	36.852	32.771	30.581	30.592	29.660	29.379	29.003	28.677	28.792	28.450	28.333	28.266	27.925	28.056	27.732
PCK	**11.832**	**11.222**	**11.555**	**11.381**	**11.406**	**11.306**	**11.512**	**11.464**	**11.511**	**11.477**	**11.463**	**11.397**	**11.482**	**11.437**	**11.474**

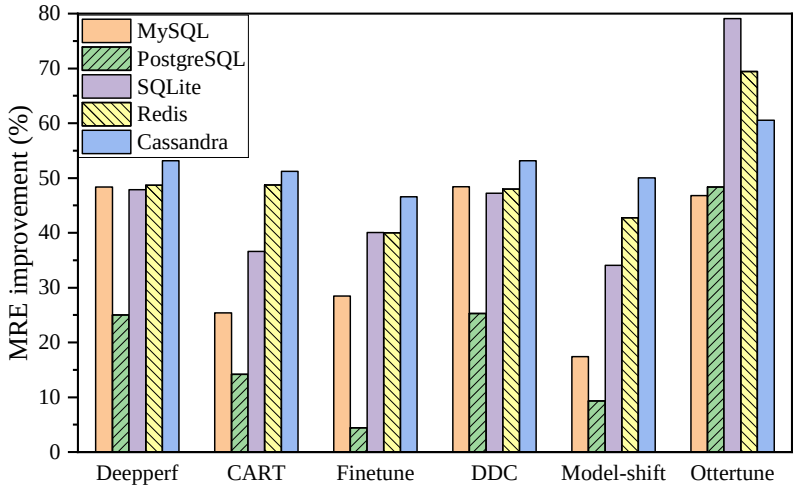

Figure 4. MRE improvement of PCK compared with six baseline algorithms.

DeepPerf tackles the performance prediction problem directly by training a performance model with DNN. Further, Finetune and DDC are two DNN-based transfer learning approaches. The above experimental results show that the MRE results of PCK are better than these three DNN-based performance modeling methods. It is because that training a high-performance DNN often requires a large amount of training data, but we can only provide a small sample due to the high cost of performance measurement.

In another aspect, we find out that the two DNN-based transfer learning methods in most cases perform higher prediction accuracy than simple DNN-based performance prediction model in five database systems. Similarly, the Model-shift approach transfers the CART-based performance model in the source to the target using linear regression models, and it also achieves better MRE than CART. This observation verifies the effectiveness of transfer learning on performance prediction task.

The validity of above-mentioned transfer learning methods proves that there exists a correlation between the source and target. Our proposed PCK method aims to take advantage of this correlation to build a high-performance prediction model in target domain. In the experiment results, the PCK achieves higher prediction accuracy compared with other four transfer learning method under almost all the sample sizes.

From another point of view, PCK requires a much smaller number of target samples to reach the same standard of prediction accuracy, compared with other baseline approaches. It means that PCK outperforms six baseline algorithms not only in terms of prediction accuracy, but also in terms of measurement effort.

To further verify our conclusion, we conduct the similar experiments across different version changes. The MRE comparison among different approaches for MySQL and Redis under different version change scenarios (version 5.5–5.7, 5.7–8.0 for MySQL, and version 4.0.1–5.0.0, 5.0.0–6.0.5 for Redis) are listed in Tables 7–10, respectively. Experimental results confirm that the PCK outperforms the state-of-the-art baselines in terms of prediction accuracy and measurement cost in almost all experimental settings in this paper.

Table 7. MRE comparison among different approaches for MySQL (version 5.5–5.7).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	23.382	9.871	12.726	11.228	7.129	9.979	8.657	11.102	6.753	9.426	8.412	9.59	10.216	10.35	10.188
CART	5.894	7.346	10.773	8.867	9.343	6.282	9.692	7.313	7.324	6.977	6.874	6.681	6.202	5.336	5.266
Finetune	7.018	6.408	5.473	5.665	5.822	5.011	4.897	5.149	4.644	5.357	3.908	4.81	4.726	4.048	3.823
DDC	13.168	9.108	8.034	14.538	9.597	13.554	11.181	9.321	10.081	10.472	10.152	10.928	9.201	10.797	8.821
Model-shift	5.587	5.627	12.67	5.461	6.82	8.996	6.34	4.789	7.892	9.289	5.382	9.685	5.586	5.871	10.351
Ottertune	18.722	17.439	15.247	18.019	16.598	15.49	15.321	14.539	19.944	21.559	15.15	19.487	19.186	13.52	18.009
PCK	**2.658**	**2.368**	**2.646**	**2.573**	**2.639**	**2.594**	**2.616**	**2.659**	**2.552**	**2.544**	**2.594**	**2.556**	**2.504**	**2.547**	**2.523**

Table 8. MRE comparison among different approaches for MySQL (version 5.7–8.0).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	16.05	14.429	14.079	15.59	16.058	14.952	15.827	15.063	15.694	16.011	15.847	15.171	15.283	16.054	16.123
CART	14.216	15.69	15.623	15.188	14.899	14.761	14.673	15.138	14.829	15.332	14.804	15.179	15.111	14.978	15.068
Finetune	11.484	11.893	11.596	12.333	12.052	11.973	12.393	11.949	12.068	12.029	12.23	12.277	12.291	12.262	12.21
DDC	12.338	14.678	15.911	15.801	15.371	15.87	15.262	15.743	15.239	15.53	15.51	16.239	15.431	15.362	15.826
Model-shift	15.19	14.767	14.529	14.161	14.14	14.306	13.889	14.123	14.087	14.105	13.985	13.948	14.121	14.079	14.219
Ottertune	26.182	20.683	18.483	16.688	17.756	13.857	14.196	14.553	11.754	12.433	11.537	11.065	10.648	**10.298**	**9.311**
PCK	**10.522**	**10.392**	**10.355**	**10.487**	**10.256**	**10.332**	**10.326**	**10.301**	**10.272**	**10.295**	**10.331**	**10.359**	**10.319**	10.303	9.997

Table 9. MRE comparison among different approaches for Redis (version 4.0.1–5.0.0).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	9.058	5.308	4.421	3.654	2.879	2.442	2.014	2.118	2.143	2.037	1.854	1.737	1.81	1.761	1.821
CART	1.533	1.553	1.556	1.543	1.451	1.538	1.624	1.534	1.455	1.483	1.524	1.533	1.496	1.465	1.519
Finetune	1.561	1.756	1.653	1.577	1.555	1.558	1.516	1.521	1.513	1.52	1.491	1.485	1.539	1.458	1.446
DDC	8.308	6.838	4.144	3.133	2.28	2.291	2.063	2.068	2.004	1.728	1.748	1.88	1.834	1.824	1.704
Model-shift	1.561	1.66	1.531	1.558	1.602	1.529	1.547	1.55	1.554	1.562	1.528	1.555	1.502	1.534	1.549
Ottertune	36.173	15.021	9.428	7.813	8.428	7.331	7.361	6.491	6.146	5.836	5.603	5.279	5.18	4.906	4.496
PCK	**1.322**	**1.32**	**1.314**	**1.313**	**1.284**	**1.261**	**1.275**	**1.202**	**1.161**	**1.113**	**1.088**	**0.999**	**0.904**	**0.911**	**1.01**

Table 10. MRE comparison among different approaches for Redis (version 5.0.0–6.0.5).

Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
DeepPerf	10.168	4.634	4.533	3.752	2.413	2.436	2.092	1.805	1.87	1.879	1.718	1.732	1.601	1.641	1.612
CART	1.266	1.391	1.259	1.416	1.349	1.311	1.298	1.32	1.308	1.31	1.334	1.257	1.271	1.323	1.362
Finetune	1.781	1.588	1.495	1.457	1.534	1.528	1.451	1.339	1.407	1.303	1.347	1.306	1.335	1.317	1.272
DDC	8.53	6.708	3.56	2.529	2.61	1.861	2.113	1.907	1.973	1.838	1.654	1.717	1.61	1.646	1.601
Model-shift	1.196	1.425	1.41	1.406	1.372	1.163	1.243	1.271	1.378	1.302	1.346	1.264	1.337	1.283	1.33
Ottertune	27.598	12.593	9.313	9.703	8.023	7.651	7.166	6.636	6.768	6.597	5.941	5.592	5.433	5.01	4.565
PCK	**0.711**	**0.648**	**0.641**	**0.64**	**0.636**	**0.636**	**0.635**	**0.634**	**0.634**	**0.635**	**0.634**	**0.633**	**0.633**	**0.632**	**0.631**

5.3. Trade-Off on Choosing K

In this part, we will discuss the trade-off on choosing the number of clusters (K), and illustrate the influence of different K values on the prediction accuracy. In order to explore this problem, we systematically vary the value of K from 1 to 10 in each subject system and we measure the prediction accuracy in each case. Taking MySQL (version 5.5–8.0) as an example, the experiment results are shown in Table 11.

Our results, in Table 11, indicate that PCK achieves the highest prediction accuracy for almost all sample sizes when K equals to 3 in MySQL. The *MRE* of PCK decreases when K increases appropriately. This demonstrates that clustering different smooth regions could help boost the performance of prediction model. However, the prediction accuracy will not continue to grow when K exceeds a certain threshold, such as 3 in the above example. This is due to the insufficient target samples. If a large K is chosen in PCK, the available target sample in each region will be too little to learn a reliable performance model. Consequently, the choice of K is the key to whether the PCK method can achieve high prediction accuracy.

Table 11. MRE comparison with different K for MySQL.

Cluster Size \ Sample Size	1n	2n	3n	4n	5n	6n	7n	8n	9n	10n	11n	12n	13n	14n	15n
1	15.044	12.459	12.017	12.191	12.138	12.042	12.119	12.192	12.166	12.237	12.285	12.187	12.199	12.303	12.246
2	14.272	14.492	12.353	10.992	11.229	10.969	10.314	10.285	10.454	10.515	10.429	10.445	10.364	10.357	10.337
3	**12.399**	**12.177**	**10.222**	**10.734**	**10.747**	10.922	**9.618**	**9.268**	**9.645**	**9.273**	**9.303**	**9.2304**	**9.183**	**9.280**	**9.197**
4	14.247	14.326	11.447	11.927	11.372	**10.819**	10.606	10.524	10.207	10.735	10.251	10.239	10.019	10.100	9.981
5	16.045	13.674	12.458	11.943	12.435	11.119	10.381	10.569	11.027	10.347	10.097	10.010	10.084	10.027	9.931
6	16.070	13.497	12.611	11.596	11.941	11.295	11.491	11.129	11.732	10.668	10.818	10.776	10.166	10.695	10.298
7	15.287	15.504	13.382	13.005	12.271	11.748	11.547	12.520	11.918	11.731	11.631	11.924	11.408	11.821	11.858
8	15.300	14.245	13.270	11.910	12.629	11.490	11.636	11.928	11.258	11.040	11.284	10.901	10.944	10.785	10.732
9	15.464	14.066	12.943	12.756	12.254	12.332	11.842	11.601	11.929	10.984	10.949	10.999	11.042	10.920	10.892
10	17.349	14.897	13.553	13.476	12.945	12.484	11.818	11.894	11.514	12.064	10.908	11.383	11.270	10.762	10.811

In order to further verify the necessity of PCK, we compare the *MRE* in different clusters ($K = 3$) with the *MRE* without clustering ($K = 1$) in MySQL, the result is shown in Figure 5. We observe that the *MRE* without clustering is higher than the *MRE* in three different clusters in almost all sample sizes. Finally, the optimal prediction accuracy is achieved in this case ($K = 3$). In addition, PCK achieves the best prediction performance in Redis (version 4.0.1–6.0.5) and PostgreSQL (version 9.3–11.0) when $K = 2$, and $K = 1$ in Cassandra (version 2.1.0–3.11.6), $K = 4$ in SQLite (version 3.31.1–3.36.0), respectively. The reason for the small K in Cassandra is its larger configuration space, thus the insufficient measurements may lead to inaccurate partitioning.

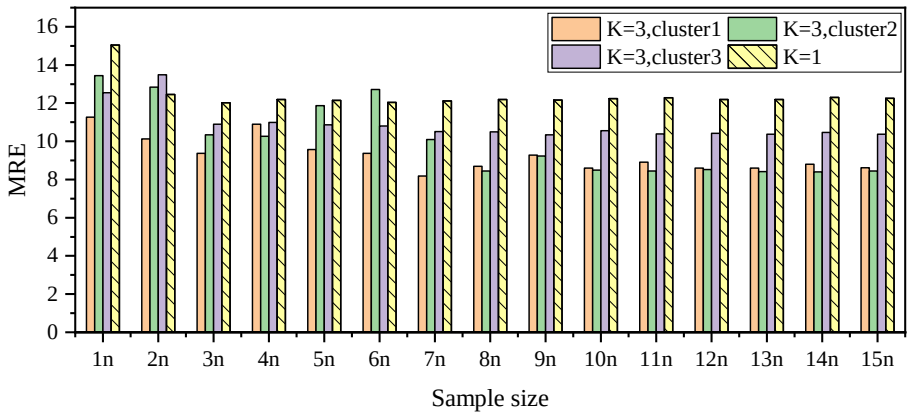

Figure 5. MRE comparison in different cluster ($K = 3$).

6. Discussion

6.1. Prediction Accuracy

Experimental results on five different database systems are shown in Tables 2–6, respectively. Our PCK method achieved better prediction accuracy than all the baseline algorithms. The MRE reduction of PCK over six state-of-the-art baseline algorithms rages from 30.73% to 60.83% on average. The reason for this result is that PCK can leverage the transferable knowledge in source to facilitate the performance modeling in target. The existence of transferable knowledge is based on the strong correlation between the performance responses of different versions.

We conducted experiments in different version change scenarios of the same subject database system, and the results confirmed this fact. The MRE comparisons of PCK among different version change scenarios for MySQL (as shown in Tables 2, 7, and 8) and Redis (as shown in Tables 5, 9, and 10) indicate that the smaller the version changes, the higher the prediction accuracy achieved by PCK. This result is intuitive because the smaller version change often means a stronger correlation between source and target.

In addition, an appropriate value of K can guarantee the high prediction accuracy achieved by PCK, as demonstrated in Table 11. The trade-off of choosing the value of K mainly lies in the tradeoff between the precise partition of configuration space and the insufficient samples in target. The optimal K is usually small because of the limited samples in target, thus it can be obtained simply through a few experiments.

6.2. Measurement Effort

In this paper, we assume that a number of available performance measurements in the source are available, while in the target there is a lack of samples. This is reasonable because the source database has been running for a relative long time and has been deeply studied by database administrators. Thus, we can obtain sufficient samples at a low cost and avoid the overhead of collecting a large number of samples in source. Therefore, the measurement effort in this paper refers specifically to performance measurements in the target. Experimental results on five different database systems show that PCK can achieve better performance prediction accuracy with less samples in target. To reach the same standard of prediction accuracy, almost all the baseline algorithms require more than 15 times of performance measurements in target compared with our PCK method. These results suggest that PCK can effectively decrease the measurement cost in performance prediction tasks across different database versions.

6.3. Effectiveness of Transfer Learning

We select DeepPerf and CART as baselines to verify the effectiveness of transfer learning. These two approaches establish performance prediction models in target domain directly. We also evaluate the performance of transfer learning schemes based on DeepPerf and CART given the measurements of source database. Finetune and DDC are two widely

used transfer learning schemes, and we use these two schemes on the basis of DeepPerf in this paper. Similarly, Model-shift approach uses linear regression models to shift the CART model that has been learned in the source to predict the performance of the system in the target.

Experimental results confirm the effectiveness of transfer learning. Among all the experimental settings (namely different database systems, different version change scenarios, and different sample sizes) in this paper, the probabilities that the transfer learning-based approaches perform better than the directly learning approaches are 99.3% for Finetune, 61.5% for DDC, and 74.8% for Model-shift, respectively. In other words, Finetune and DDC perform higher prediction accuracy than DeepPerf, meanwhile, Model-shift approach obtains better predication performance than CART under most conditions. Therefore, utilizing the transferable knowledge across environments is a promising direction to contribute to learning faster, better, and less costly performance models.

7. Conclusions and Future Work

The current approaches target a specific database version where one needs to learn a performance model from scratch for a newer database version. In this paper, we target the use case when the database version updates. We proposed PCK, a fast performance modeling strategy, which is orthogonal to the previously proposed modeling method. By exploiting knowledge pieces from source via both clustering and co-kriging, our proposed PCK approach significantly improves prediction accuracy and reduces excessive measurement effort of performance modeling. Experimental results on five different database systems show that PCK can achieve better performance prediction accuracy with fewer data in target. The MRE reduction of PCK over six state-of-the-art baseline algorithms rages from 30.73% to 60.83% on average. To achieve the same prediction accuracy, PCK can save more than 15 times of measurements in target compared with other approaches. Furthermore, the experimental results verify the effectiveness of transfer learning on performance prediction task.

Currently, PCK is suitable for the version change scenarios that the correlation of performance responses remains strong. It is possible to introduce a scheme to identify whether the correlation is strong or weak after the database version changes, and this will be one of our future work. Besides, our proposed method focuses on the unchanged parameter in source and target. In the future, we will continue to improve the PCK to take parameter changes (added and forbidden parameters) into consideration when transferring knowledge across different database versions. Note that PCK is proposed for the database version change scenarios. However, there are varying use cases, namely different kinds of environmental changes, such as workload change and hardware change scenarios. We will explore the usability of PCK and its variants in other environmental change scenarios, this is also an interesting future issue.

Author Contributions: Conceptualization, L.B. and R.C.; methodology, L.B. and R.C.; software, S.W. and J.D.; validation, S.W., X.W., Y.D. and R.S.; formal analysis, R.C., L.B. and S.W.; investigation, S.W., X.W., Y.D. and R.S.; resources, L.B.; data curation, S.W., X.W., Y.D. and R.S.; writing—original draft preparation, R.C.; writing—review and editing, L.B. and J.D.; visualization, R.C. and J.D.; supervision, L.B.; project administration, R.C. and X.W.; funding acquisition, L.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the National Key R&D Program of China under Grant No. 2018YFC0831200, in part by National Natural Science Foundation of China under Grant No. 61202040 with Xidian University, in part by the Key R&D Program of Shaanxi under Grant No. 2019ZDLGY13-03-02, in part by Natural Science Foundation of Shaanxi Province, China under Grant No. 2019JM-368, and in part by the Key R&D Program of Hebei under Grant No. 20310102D.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/xdbdilab/PCK.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Xu, T.; Jin, L.; Fan, X.; Zhou, Y.; Pasupathy, S.; Talwadker, R. Hey, You Have given Me Too Many Knobs!: Understanding and Dealing with over-Designed Configuration in System Software. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 30 August 2015–4 September 2015; pp. 307–319.
2. Van Aken, D.; Pavlo, A.; Gordon, G.J.; Zhang, B. Automatic database management system tuning through large-scale machine learning. In Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, 14–19 May 2017; pp. 1009–1024.
3. Guo, J.; Czarnecki, K.; Apel, S.; Siegmund, N.; Wąsowski, A. Variability-aware performance prediction: A statistical learning approach. In Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), Silicon Valley, CA, USA, 11–15 November 2013; pp. 301–311.
4. Lu, J.; Chen, Y.; Herodotou, H.; Babu, S. Speedup your analytics: Automatic parameter tuning for databases and big data systems. *Proc. VLDB Endow.* **2019**, *12*, 1970–1973. [CrossRef]
5. Guo, J.; Yang, D.; Siegmund, N.; Apel, S.; Sarkar, A.; Valov, P.; Czarnecki, K.; Wasowski, A.; Yu, H. Data-efficient performance learning for configurable systems. *Empir. Softw. Eng.* **2018**, *23*, 1826–1867. [CrossRef]
6. Duan, S.; Thummala, V.; Babu, S. Tuning database configuration parameters with iTuned. *Proc. VLDB Endow.* **2009**, *2*, 1246–1257. [CrossRef]
7. Mahgoub, A.; Wood, P.; Ganesh, S.; Mitra, S.; Gerlach, W.; Harrison, T.; Meyer, F.; Grama, A.; Bagchi, S.; Chaterji, S. Rafiki: A middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads. In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Las Vegas, NV, USA, 11–15 December 2017; pp. 28–40.
8. Bao, L.; Liu, X.; Wang, F.; Fang, B. ACTGAN: Automatic Configuration Tuning for Software Systems with Generative Adversarial Networks. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; pp. 465–476.
9. Zhu, Y.; Liu, J.; Guo, M.; Bao, Y.; Ma, W.; Liu, Z.; Song, K.; Yang, Y. Bestconfig: Tapping the performance potential of systems via automatic configuration tuning. In Proceedings of the 2017 Symposium on Cloud Computing, Santa Clara, CA, USA, 24–27 September 2017; pp. 338–350.
10. Ha, H.; Zhang, H. Deepperf: Performance prediction for configurable software with deep sparse neural network. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 25–31 May 2019; pp. 1095–1106.
11. Nair, V.; Menzies, T.; Siegmund, N.; Apel, S. Faster discovery of faster system configurations with spectral learning. *Autom. Softw. Eng.* **2018**, *25*, 247–277. [CrossRef]
12. Sarkar, A.; Guo, J.; Siegmund, N.; Apel, S.; Czarnecki, K. Cost-efficient sampling for performance prediction of configurable systems (t). In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 9–13 November 2015, pp. 342–352.
13. Valov, P.; Guo, J.; Czarnecki, K. Empirical comparison of regression methods for variability-aware performance prediction. In Proceedings of the 19th International Conference on Software Product Line, Nashville, TN, USA, 20–24 July 2015; pp. 186–190.
14. Nair, V.; Menzies, T.; Siegmund, N.; Apel, S. Using bad learners to find good configurations. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; pp. 257–267.
15. Jamshidi, P.; Velez, M.; Kästner, C.; Siegmund, N. Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista, FL, USA, 4–9 November 2018; pp. 71–82.
16. Zhang, J.; Liu, Y.; Zhou, K.; Li, G.; Xiao, Z.; Cheng, B.; Xing, J.; Wang, Y.; Cheng, T.; Liu, L.; et al. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In Proceedings of the 2019 International Conference on Management of Data, Amsterdam, The Netherlands, 30 June 2019–5 July 2019; pp. 415–432.
17. Jamshidi, P.; Siegmund, N.; Velez, M.; Kästner, C.; Patel, A.; Agarwal, Y. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, USA, 30 October–3 November 2017; pp. 497–508.
18. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **2009**, *22*, 1345–1359. [CrossRef]
19. Oliver, M.A.; Webster, R. Kriging: A method of interpolation for geographical information systems. *Int. J. Geogr. Inf. Syst.* **1990**, *4*, 313–332. [CrossRef]
20. Myers, D.E. CO-KRIGING: Methods and alternatives. In *The Role of Data in Scientific Progress*; Glaeser, P.S., Ed.; Elsevier Science Publisher: North-Holland, The Netherlands, 1985; pp. 425–428.

21. Zhang, Y.; Guo, J.; Blais, E.; Czarnecki, K. Performance prediction of configurable software systems by fourier learning (t). In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 9–13 November 2015; pp. 365–373.
22. Zhang, Y.; Guo, J.; Blais, E.; Czarnecki, K.; Yu, H. A mathematical model of performance-relevant feature interactions. In Proceedings of the 20th International Systems and Software Product Line Conference, Beijing, China, 16–23 September 2016; pp. 25–34.
23. Kolesnikov, S.; Siegmund, N.; Kästner, C.; Grebhahn, A.; Apel, S. Tradeoffs in modeling performance of highly configurable software systems. *Softw. Syst. Model.* **2019**, *18*, 2265–2283. [CrossRef]
24. Narayanan, D.; Thereska, E.; Ailamaki, A. Continuous resource monitoring for self-predicting DBMS. In Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Atlanta, GA, USA, 27–29 September 2005; pp. 239–248.
25. Tran, D.N.; Huynh, P.C.; Tay, Y.C.; Tung, A.K. A new approach to dynamic self-tuning of database buffers. *ACM Trans. Storage (TOS)* **2008**, *4*, 1–25. [CrossRef]
26. Tian, W.; Martin, P.; Powley, W. Techniques for automatically sizing multiple buffer pools in DB2. In Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research, Toronto, ON, Canada, 6–9 October 2003; pp. 294–302.
27. Thummala, V.; Babu, S. iTuned: A tool for configuring and visualizing database parameters. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, Indianapolis, IN, USA, 6–10 June 2010; pp. 1231–1234.
28. Tan, J.; Zhang, T.; Li, F.; Chen, J.; Zheng, Q.; Zhang, P.; Qiao, H.; Shi, Y.; Cao, W.; Zhang, R. ibtune: Individualized buffer tuning for large-scale cloud databases. *Proc. VLDB Endow.* **2019**, *12*, 1221–1234. [CrossRef]
29. Li, G.; Zhou, X.; Li, S.; Gao, B. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.* **2019**, *12*, 2118–2130. [CrossRef]
30. Tan, Z.; Babu, S. Tempo: Robust and self-tuning resource management in multi-tenant parallel databases. *Proc. VLDB Endow.* **2016**, *9*, 720–731. [CrossRef]
31. Mahgoub, A.; Wood, P.; Medoff, A.; Mitra, S.; Meyer, F.; Chaterji, S.; Bagchi, S. SOPHIA: Online reconfiguration of clustered nosql databases for time-varying workloads. In Proceedings of the 2019 USENIX Annual Technical Conference, Renton, WA, USA, 10–12 July 2019; pp. 223–240.
32. Zhang, B.; Van Aken, D.; Wang, J.; Dai, T.; Jiang, S.; Lao, J.; Sheng, S.; Pavlo, A.; Gordon, G.J. A demonstration of the ottertune automatic database management system tuning service. *Proc. VLDB Endow.* **2018**, *11*, 1910–1913. [CrossRef]
33. Valov, P.; Petkovich, J.C.; Guo, J.; Fischmeister, S.; Czarnecki, K. Transferring performance prediction models across different hardware platforms. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, L'Aquila, Italy, 22–26 April 2017; pp. 39–50.
34. Jamshidi, P.; Velez, M.; Kästner, C.; Siegmund, N.; Kawthekar, P. Transfer learning for improving model predictions in highly configurable software. In Proceedings of the 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Buenos Aires, Argentina, 22–23 May 2017; pp. 31–41.
35. Javidian, M.A.; Jamshidi, P.; Valtorta, M. Transfer learning for performance modeling of configurable systems: A causal analysis. *arXiv* **2019**, arXiv:1902.10119.
36. Krishna, R.; Nair, V.; Jamshidi, P.; Menzies, T. Whence to Learn? Transferring Knowledge in Configurable Systems using BEETLE. *IEEE Trans. Softw. Eng.* **2020**. [CrossRef]
37. Bergstra, J.; Bengio, Y. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
38. Wong, J.A.H.A. Algorithm AS 136: A K-Means Clustering Algorithm. *J. R. Stat. Soc.* **1979**, *28*, 100–108.
39. Siegmund, N.; Grebhahn, A.; Apel, S.; Kästner, C. Performance-influence models for highly configurable systems. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 30 August–4 September2015; pp. 284–294.
40. Ishihara, Y.; Shiba, M. Dynamic Configuration Tuning of Working Database Management Systems. In Proceedings of the 2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech), Kyoto, Japan, 10–12 March 2020; pp. 393–397.
41. Zheng, C.; Ding, Z.; Hu, J. Self-tuning performance of database systems with neural network. In *International Conference on Intelligent Computing, Proceedings of the Intelligent Computing Theory, ICIC 2014, Taiyuan, China, 3–6 August 2014*; Springer: Cham, Switzerland, 2014; pp. 1–12.
42. Debnath, B.K.; Lilja, D.J.; Mokbel, M.F. SARD: A statistical approach for ranking database tuning parameters. In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering Workshop, Cancun, Mexico, 7–12 April 2008; pp. 11–18.
43. Kanellis, K.; Alagappan, R.; Venkataraman, S. Too many knobs to tune? towards faster database tuning by pre-selecting important knobs. In Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20), Virtual, 13–14 July 2020.
44. Mahgoub, A.; Medoff, A.M.; Kumar, R.; Mitra, S.; Klimovic, A.; Chaterji, S.; Bagchi, S. OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20), Virtual, 15–17 July 2020; pp. 189–203.
45. Cooper, B.F.; Silberstein, A.; Tam, E.; Ramakrishnan, R.; Sears, R. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing, Indianapolis, IN, USA, 10–11 June 2010; pp. 143–154.

46. Yosinski, J.; Clune, J.; Bengio, Y.; Lipson, H. How Transferable Are Features in Deep Neural Networks? In Proceedings of the 27th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; MIT Press: Cambridge, MA, USA, 2014; Volume 2, pp. 3320–3328.
47. Tzeng, E.; Hoffman, J.; Zhang, N.; Saenko, K.; Darrell, T. Deep domain confusion: Maximizing for domain invariance. *arXiv* **2014**, arXiv:1412.3474.