The Journal of Systems and Software 209 (2024) 111936

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

ETune: Efficient configuration tuning for big-data software systems via configuration space reduction[☆]

Rong Cao [*], Liang Bao, Kaibi Zhao, Panpan Zhangsun

School of Computer Science and Technology, Xidian University, No. 2 South Taibai Road, Xi'an, 710071, ShaanXi, China

ARTICLE INFO

Keywords:
Configuration tuning
Configuration space reduction
Transfer learning
Parameter selection
Big-data software systems

ABSTRACT

Configuration tuning for big-data software systems is generally challenging due to the complex configuration space and costly performance evaluation. The workload changes over time in the context of big data making this problem even more difficult. To address the low efficiency issue caused by the need to retune from scratch after workload changes, we present ETune based on a combination of an Bayesian Optimization (BO) based tuner and configuration space reduction techniques to efficiently find high-performance configurations for big-data systems. For configuration space reduction, we develop two approaches: (1) shrinking the configuration space by trading off the exploitation and exploration between the impactful parameters and other parameters in the tuning process, (2) generating the promising regions of configuration space by transferring knowledge across past tuning tasks. The two configuration space reduction methods aim to reduce the huge configuration space to a compact but promising one, and searching in the reduced configuration space can further accelerate the configuration tuning. The extensive experiments show that our configuration space reduction methods considerably boost the BO-based tuner, and ETune significantly improves the tuning efficiency compared with the transfer learning based state-of-the-arts.

1. Introduction

The employment of big-data frameworks has become the de-facto standard when developing large-scale data-driven applications (Krishna et al., 2020). The frameworks such as Spark are highly configurable as they provide users with a large number of configurable parameters. These parameters allow users to customize the big-data system to meet their specific requirements, and are critical to system performance such as execution time, which directly affect user experience (Zhu and Liu, 2019). Hence, it is practically very important to make rational decisions on system configurations. However, finding a good configuration for a big-data system with a given workload is generally challenging due to the huge configuration space and expensive performance evaluation, and this problem is proven to be NP-hard (Sullivan et al., 2004). Moreover, the workloads from real-world applications change over time. In big-data software systems, a workload is an application deployed on the cloud and its input (Hsu et al., 2018b). The dynamic changes in workloads (including the applications and corresponding input data sizes) complicate the configuration tuning for big-data systems. Users struggle with this difficulty, but manually tuning possibly tens to hundreds of parameters do not guarantee the performance across various workloads and could not scale. Therefore, automatic configuration tuning becomes an appealing feature for cloud providers.

Automatically tuning the configurations of big-data software systems has attracted lots of interest from academia and industry. Most of the existing studies for configuration tuning assume a fixed workload (Wang et al., 2016; Bei et al., 2015; Bao et al., 2018a; Li et al., 2018a; Dalibard et al., 2017; Jamshidi and Casale, 2016; Nair et al., 2018b; Trotter et al., 2017; Fischer et al., 2015; Yeh et al., 2016; Liu et al., 2020). When the workload changes, a re-tuning from scratch is performed, leading to the low-efficiency issue (Jamshidi et al., 2017). To address this issue, researchers introduce the idea of transfer learning to improve the tuning efficiency, which leverages the tuning experience from the previous tasks (source tasks) to speed up the current task (target task) (Fekry et al., 2020; Van Aken et al., 2017; Zhang et al., 2021b). This part of the related work usually follows the Bayesian Optimization (BO) framework, and learns surrogate models with the aid of past tuning history, and the surrogate models are used to guide the search of good configurations. Orthogonal to the existing approaches,

[☆] Editor: Nicole Novielli.
[*] Corresponding author.
E-mail addresses: caorong@stu.xidian.edu.cn (R. Cao), baoliang@mail.xidian.edu.cn (L. Bao), 23031110056@stu.xidian.edu.cn (K. Zhao), zhangsunpanpan@stu.xidian.edu.cn (P. Zhangsun).

https://doi.org/10.1016/j.jss.2023.111936
Received 17 May 2023; Received in revised form 12 October 2023; Accepted 16 December 2023
Available online 20 December 2023

our intuition is that reducing the huge configuration space to a compact but promising configuration space, and searching in this reduced configuration space can further accelerate the configuration tuning.

In this paper, we perform Configuration Space Reduction (CSR) in two ways: on the one hand, CSR by considering the importance of parameters, which uses the knowledge gained during target tuning task; and on the other hand, CSR by transferring knowledge from similar source workloads. **The former** is motivated by the observation that although the big-data systems have hundreds of parameters, only a small subset of them significantly impacts system performance for each workload (Kanellis et al., 2020; Li et al., 2018b). Focusing on influential parameters in configuration tuning not only improves the efficiency of the tuning algorithm, but also avoids wasting resources. Existing tuning frameworks (Fekry et al., 2020; Van Aken et al., 2017) perform parameter selection prior to configuration tuning, and only the selected parameters are tuned. This approach achieves a certain degree of performance improvements, but there are still some problems. The first problem is that a fraction of the tuning time is dedicated to parameter selection, making the already small tuning time even more limited, especially in scenarios where performance evaluations are costly. The second problem is that it requires a high degree of accuracy in parameter selection, which is often not feasible given a small number of performance observations. To solve the above problems, we propose the Parameter Selection based Configuration Space Reduction (PS-CSR) which puts the parameter selection in the process of configuration tuning, and trades off the exploitation and exploration between the impactful parameters and other parameters in the tuning process. This removes the preceding parameter selection phase and allows updating the subset of important parameters during the tuning process, avoiding the omission of impactful parameters and consequently wasted potential for configuration tuning.

The latter aims to accelerate configuration tuning by leveraging the knowledge about promising regions in the configuration space acquired from past tuning tasks. Similarities across workloads offer an opportunity for the tuner to improve the tuning efficiency via transfer learning (Jamshidi et al., 2017). The similarities arise for several reasons: (1) most workloads are parameterized, (2) early processing tasks in analytics pipelines are often similar, (3) repeated workloads with similar input data sizes (Zaouk et al., 2019). For cloud providers, the performance observations saved from configuration tuning of a particular big-data system under a large number of different workloads is readily available. The configuration tuning problem for different workloads has the same configuration space. There is generally a high degree of sparsity in the configuration space, and the promising configurations are close to each other and possibly locate at a few promising regions for a given workload (Zhu and Liu, 2019). Furthermore, the promising regions of different workloads are somewhat similar because of the similarities across workloads. Therefore, transferring promising regions (i.e., regions where high-performance configurations are densely distributed) among similar workloads is a feasible way to improve tuning efficiency while guaranteeing tuning effectiveness, and we propose a Transfer Learning based Configuration Space Reduction (TL-CSR) approach based on this idea. We first select some source tasks in the data repository that are similar to the target task, then extract promising regions in each selected source task and generate the reduced configuration space of the target task by an ensemble model. We assume the existence of a data repository with past tuning history (i.e., performance observations in this work) for different workloads, which is common in similarity-aware configuration tuning (Sampaio et al., 2023; Zhang et al., 2021b; Van Aken et al., 2017; Fekry et al., 2020).

The above two CSR methods can be integrated in an BO-based tuner to achieve efficient configuration tuning for big-data software systems via configuration space reduction, and the entire tuning framework is named ETune. In summary, our work makes the following contributions:

Table 1
Execution time savings when using the respective best configuration for each workload, compared against reusing the best configuration found for Sort-10G.

Application-Input	Sort-20G	WC-30G	WC-40G	WC-50G
Execution time savings (%)	22.95	5.19	2.71	21.88

- We propose ETune, which integrates PS-CSR and TL-CSR methods in an BO-based tuner to perform efficient configuration tuning for big-data software systems under workload change scenarios.
- We propose two CSR methods to design a compact but promising configuration space, further accelerate the tuning task. These methods are orthogonal to the existing methods and can be seamlessly combined to pursue better performance of configuration tuning.
- We evaluate the performance of ETune through extensive experiments using 12 workloads under five representative Spark applications, which show that ETune is able to find configurations comparable to two state-of-the-art baselines, while reducing tuning time by 46.21%–52.14% on average.
- We combine CSR methods with the existing tuners, and compare the performance with original tuners under different workloads. The results show that the combined approaches lead to a 18.57%–31.12% reduction in tuning time on average, which validates the interoperability of our CSR methods.

2. Preliminary

2.1. Workload-specific configuration tuning

A workload refers to all input received by a given technological infrastructure (Calzarossa et al., 2016). The workloads from real-world applications change over time, and the frequent changes in workload usually occur in two ways, either when the execution logic changes or when the size of the data to be processed changes significantly. In particular, changes in the data size are very common in the context of big data.

However, there is no one-size-fits-all configuration for different workloads. To verify this, we randomly sample 50 configurations and then measure the performance (i.e., execution time) of these configurations under different workloads (i.e., applications and input data sizes). We use the best configuration under the Sort-10G workload as the baseline and measure the execution time saved by using the respective best configuration under other workloads compared to using that configuration directly, as shown in Table 1. We observe that re-tuning the configuration for each workload achieves performance gains of greater than 20%. It is worth noting that the results are obtained in the 50 randomly sampled configurations for each workload, and that the different input data sizes do not vary significantly. Moreover, executing a big-data workload takes minutes to hours or even days, making it possible to achieve a large performance gain for a smaller percentage of execution time savings. The above results demonstrate the urgent need for workload-specific configuration tuning, and the similar observations are found in other research work (Fekry et al., 2020; Zhang et al., 2021b; Yu et al., 2018).

2.2. Problem formulation

Consider a big-data system having configuration parameters $c_1, c_2, \dots, c_m$, the configuration space is defined as the Cartesian product of all configuration parameters $C = Dom(c_1) \times \cdots \times Dom(c_m)$, where $Dom(c_i)$ is the valid range of each parameter. A configuration parameter c_i can be either (1) a numerical variable (real or integer) within the valid range of the parameter, or (2) a categorical variable or a Boolean variable. We denote the system performance as f which can be any chosen performance metric to be optimized. Given a workload and a specific configuration $\boldsymbol{c}$, the corresponding performance $f(\boldsymbol{c})$ can be observed after evaluating it in the big-data system. We denote the performance observations as $H_t = \{\boldsymbol{c}_i, f(\boldsymbol{c}_i)\}_{i=1}^{t}$, where t is the number of performance observations.

The configuration tuning can be modeled as a black-box optimization problem. Given a configuration space C and tuning history $H_t^1, \ldots, H_t^N$ from previous tuning tasks, where N is the number of previous tasks, we need to optimize the current tuning task. Assuming that the objective is a minimization problem, configuration tuning aims to find a configuration $\boldsymbol{c}^* \in C$, where

$$\boldsymbol{c}^* = \arg\min_{c \in C} f(\boldsymbol{c}). \tag{1}$$

In this work, we consider methods that output a compact but promising configuration space $\hat{C} \subseteq C$ with the aid of configuration tuning history from past tasks. Instead of Eq. (1), we solve the following problem:

$$\boldsymbol{c}^* = \arg\min_{c \in \hat{C}} f(\boldsymbol{c}). \tag{2}$$

While $\hat{C}$ is much smaller than C, tuning methods may find the optimal configurations faster. Therefore, the goal of this work is to improve the tuning efficiency by reducing the original configuration space C to a compact space $\hat{C}$ such that it contains a proper subset of candidate configurations, which is close to the good regions in the original space C.

2.3. Bayesian optimization

Bayesian optimization (BO) is a powerful global optimization algorithm that can obtain optimal solutions for complex objective functions with few evaluations (Shahriari et al., 2015). Its core idea is to use the complete historical information to improve the search efficiency. BO consists of two main components: (1) a surrogate model for modeling the objective function, and (2) an acquisition function for deciding where to sample next. The BO framework uses a surrogate model to approximate the objective function and actively selects the most promising point for evaluation based on the surrogate model and acquisition function to avoid unnecessary sampling.

To solve the configuration tuning problem, BO optimizes the performance function f through iterative sampling the promising configurations in the configuration space. BO starts by evaluating the performance of some initially selected configurations and recording the corresponding configuration-performance pairs. Then, it iterates the following three steps: (1) build a surrogate model based on the configuration-performance pairs evaluated so far; (2) employ the surrogate model and acquisition function to select a promising configuration to evaluate next; (3) evaluate the performance of the newly selected configuration. Finally, the configuration with the best observed performance is output.

3. Our approach

3.1. Overview

In this paper, we propose an ETune approach that can improve tuning efficiency by shrinking the configuration space. Fig. 1 shows the overview of ETune. The **base tuner** is the foundation of ETune, and is used to recommend the optimal configuration in the target task. We use a BO-based tuner due to the fact that BO is an efficient and effective global optimization method and is widely used in configuration tuning. To improve the efficiency of the base tuner, we introduce **configuration space reduction**, i.e., reducing the configuration space by selecting some candidate configurations from the huge configuration space in each iteration of the base tuner. Configuration space reduction consists of two steps, firstly PS-CSR and secondly TL-CSR, which are based on parameter selection and transfer learning, respectively. The input of ETune includes performance observations from each tuning task (including the target task and N source tasks), and the output is the optimal configuration in the target task.

With the surrogate model (i.e., Random Forest (RF)) of the base tuner, parameter importance can be easily obtained. We should not only exploit the important parameters to improve the tuning efficiency, but also explore the whole configuration space to avoid the negative impact of wrong parameter selection in the initial iterations. **PS-CSR** takes advantage of the knowledge gained during the tuning process of target task and gives initial candidate configurations by employing Latin Hypercube Sampling (LHS) to the important parameters and applying Random Sampling (RS) to the entire configuration space (i.e., all parameters). RS effectively searches a large configuration space, especially when the configuration parameters are not equally important (Bao et al., 2019b). LHS is a type of stratified sampling, and experiments conducted through LHS provide much better space coverage (Duan et al., 2009).

Moreover, similarities across workloads provide an opportunity for **TL-CSR**. The similar source tasks are selected from the data repository, then the promising regions are extracted and used to generate the reduced configuration space of target task by an ensemble model. This ensemble model is used to test whether the initial candidate configurations previously selected by PS-CSR are in the promising regions of the target task. In the candidate configurations selection, the number of candidate configurations that pass the test needs to satisfy a predefined threshold. If it is not satisfied, a supplementary sampling is performed until the required number of candidate configurations is reached. Ultimately, these candidate configurations are provided to the base tuner, which selects the most potential of them via acquisition function (i.e., Expected Improvement (EI)) for the next evaluation.

In the following, we introduce our base tuner in Section 3.2, then present the PS-CSR and TL-CSR in Sections 3.3 and 3.4 respectively, finally the overall algorithm of CSR is described in Section 3.5.

3.2. Base tuner

BO provides a certain degree of freedom in its implementation, such as the type of surrogate model used and the acquisition function. The surrogate models can be set to different types of machine learning models. The most popular choice is Gaussian Process (GP) since it provides good predictions in low-dimensional numerical input spaces and allows the computation of the posterior GP model in closed form (Rasmussen, 2003). Another popular choice is the tree-based models, they are particularly well suited for high-dimensional input spaces and partially categorical input spaces (Feurer et al., 2014), which are more in line with the characteristics of the configuration spaces of big-data systems. Moreover, the tree-based models can well model the non-linear interactions (Hsu et al., 2018a). Among them, RF is particularly well suited for BO in high dimensions due to its robustness and automated feature selection (Breiman, 2001; Fekry et al., 2020). These abovementioned three advantages make RF particularly suitable as a surrogate model for BO frameworks used in configuration tuning for big-data systems. Therefore, we take RF as the surrogate model of BO in this work.

The role of acquisition function is a trade-off between exploring new regions in the configuration space and exploiting regions that are already known to have well-performed configurations. The popular acquisition functions are Expected Improvement (EI) (Jones et al., 1998), Probability of Improvement (PI) (Hoffman et al., 2011), Upper Confidence Bound (UCB) (Srinivas et al., 2010) and Entropy Search (Hennig and Schuler, 2012). Among them, EI is the most commonly-used one because of its excellent empirical performance. EI considers not only the probability of improvement, but also the magnitude of the possible improvement. BO decides the most potential configuration to evaluate next by maximizing the EI acquisition function $a(\boldsymbol{c}; \mathcal{M})$ in each iteration, where $\mathcal{M}$ refers to the surrogate model that provides performance

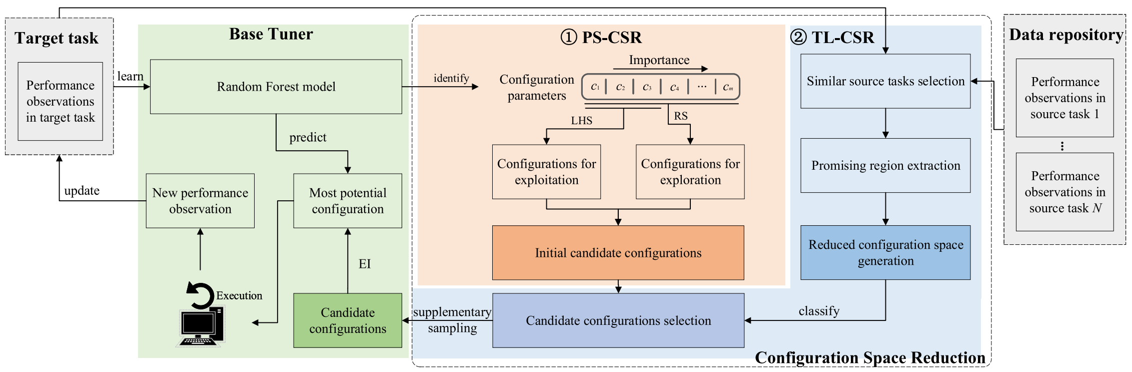
Target task
Performance observations in target task
learn
Base Tuner
Random Forest model
predict
update
New performance observation
Most potential configuration
EI
Execution
Candidate configurations
identify
① PS-CSR
Importance
Configuration parameters
c_1 | c_2 | c_3 | c_4 | ⋯ | c_m
LHS
RS
Configurations for exploitation
Configurations for exploration
Initial candidate configurations
supplementary sampling
Candidate configurations selection
② TL-CSR
Similar source tasks selection
Promising region extraction
Reduced configuration space generation
classify
Configuration Space Reduction
Data repository
Performance observations in source task 1
Performance observations in source task N

Fig. 1. Overview of Our ETune Approach.

Algorithm 1 BO-based base tuner

```
Input: N: the number of performance evaluations; C: the configuration
    space; M: the surrogate model RF; a: the acquisition function; C_in:
    the initially sampled configurations.
Output: c*: the optimal configuration.
 1: for $c \in C_{in}$ do
 2:     evaluate the performance $f(c)$ of $c$;
 3:     update the observations $H = H \cup (c, f(c))$;
 4: end for
 5: for $i = |C_{in}| + 1$ to $N$ do
 6:     fit a RF model $\mathcal{M}$ based on observations $H$;
 7:     configuration space reduction: $\hat{C} = reduce(C)$;
 8:     select the configuration to evaluate next: $c_i = \arg\max_{c \in \hat{C}} a(c; \mathcal{M})$;
 9:     evaluate the performance $f(c_i)$ of $c_i$;
10:     update the observations $H = H \cup (c_i, f(c_i))$;
11: end for
12: return $c^*$ with the best performance in $H$;
```

predictions for unseen configurations. The BO framework of base tuner is described in Algorithm 1, where CSR (line 7) is what we will subsequently describe in detail. Our goal is to improve the efficiency of configuration tuning by tuning over a reduced configuration space, while avoiding compromising the tuning effectiveness.

3.3. Parameter selection based configuration space reduction

By eliminating the less important parameters in tuning process, the configuration space can be reduced significantly (Cao et al., 2020), further speed up the search for a near-optimal configuration (Kanellis et al., 2020; Li et al., 2018b). The most widely used methods for identifying important parameters of a software system are Lasso (Van Aken et al., 2017; Li et al., 2018b) and random forest regression analysis (Kanellis et al., 2020; Fekry et al., 2020). Existing studies show that the latter performs better than the former (Li et al., 2018b; Zhang et al., 2021a). This is because that RFs are more suitable for modeling non-linear relationships and interaction effects than Lasso regression, and Gini importance has been successfully applied to high-dimensional feature selection (Menze et al., 2009). This is one of the important reasons why we choose RF as the surrogate model of base tuner.

Existing tuning frameworks perform parameter selection prior to configuration tuning, and only a subset of parameters are tuned subsequently. The first problem is that it takes up valuable tuning time, while the second is the requirement for a high degree of accuracy, which is often not feasible in the initial iterations. To solve the above problems, we put the parameter selection in the process of configuration tuning. RFs have the advantage of automatic feature selection allowing the base tuner to obtain parameter importance rankings after updating the surrogate model in each iteration. This removes the preceding parameter selection phase and allows updating the important parameters during the iterations, avoiding the omission of impactful parameters and consequent wasted potential for configuration tuning.

The base tuner needs to select one of the most promising configurations out of the remaining configuration space in each iteration. However, due to the huge configuration space, finding the most promising configuration by using the RF model to predict all configurations in the remaining configuration space is inefficient and often infeasible. Thus, we select a subset of configurations that contains configurations with a higher probability of being the most promising. In the selection of candidate configurations, we need to fully exploit the important parameters, which helps to find the most promising configuration. It is often not feasible to predict the full set of combinations of important parameters due to the presence of numerical parameters. We employ Latin Hypercube Sampling (LHS) to the important parameters since LHS is able to generate samples that thoroughly and uniformly cover the specified configuration space (Helton and Davis, 2003). Besides, other parameters are considered to be of low importance and are kept as default values. It is very challenging to accurately determine the important parameters at the early stage of the tuning process (Chen et al., 2021). This requires us to perform *exploitation* to the important parameters while maintaining *exploration* of the entire configuration space. We apply Random Sampling (RS) to the entire configuration space (i.e., all parameters), and these randomly sampled configurations are also put into the candidate configurations for exploration, which at the same time helps to avoid the configuration tuning algorithm from falling into a local optimum.

The exploration of unimportant parameters will reduce the tuning efficiency. To alleviate this problem, we gradually increase the number of LHS for important parameters and continuously decrease the number of RS for the entire configuration space in the iterations. This is because that the selection of important parameters becomes progressively more accurate as the number of performance observations increases. The number of random samples in tth iteration is determined by a piecewise decay function:

$$n_t^r = \begin{cases} n_i^r & \text{if } t < T^r, \\ max(n_i^r - s^r \times t, 0) & \text{if } t \geq T^r, \end{cases} \tag{3}$$

where n_i^r denotes the initial number of random samples, s^r is the step of decay. Moreover, the number of LHS is increased by a fixed value in each iteration:

$$n_t^l = n_i^l + s^l \times t. \tag{4}$$

3.4. Transfer learning based configuration space reduction

Inspired by the search space design method for hyperparameter tuning (Li et al., 2022), TL-CSR consists of four steps: (1) selecting similar source tasks via task similarities, (2) extracting promising regions in configuration space from source tasks, (3) generating reduced configuration space for target task, and (4) selecting candidate configurations for base tuner.

Similar Source Tasks Selection. The selection of similar source tasks is based on the task similarities between different sources and target. We adopt a widely used rank correlation metric to evaluate the similarity $S(\mathcal{M}^i; H_t^T)$ between the ith source task and the target task (Li et al., 2022; Zhang et al., 2021b; Wistuba et al., 2015), which is computed as:

$$S(\mathcal{M}^i; H_t^T) = \frac{\sum_{k=1}^{t}\sum_{l=k+1}^{t} \mathbb{I}\left((\mathcal{M}^i(\boldsymbol{c}_k) > \mathcal{M}^i(\boldsymbol{c}_l)) \odot (f(\boldsymbol{c}_k) > f(\boldsymbol{c}_l))\right)}{t(t-1)/2}, \quad (5)$$

where $\mathcal{M}^i$ denotes the RF model trained by performance observations H^i collected in the ith source task, $\mathcal{M}^i(\boldsymbol{c}_k)$ is the performance prediction of configuration $\boldsymbol{c}_k$ via $\mathcal{M}^i$, H_t^T refers to the performance observations collected in target task. Moreover, $\odot$ is the exclusive-nor operation, the above equation calculates the proportion of order-preserving configuration pairs in the source and target.

To increase the exploration of configuration space and make use of multiple source tasks, we select the similar source tasks by sampling based on the similarities between different sources and target. The probability of the source task H^i being selected is:

$$p(i) = S(\mathcal{M}^i; H_t^T) / \sum_{i=1}^{N} S(\mathcal{M}^i; H_t^T). \quad (6)$$

The n similar source tasks are selected via sampling from the N source tasks without replacement.

Promising Region Extraction. After several similar source tasks are selected, promising regions need to be extracted from each source task. The principle followed here is that when the similarity is high, we are confident of obtaining compact promising regions, while when the similarity is relatively low, we need to keep larger promising regions to ensure that we do not miss some interesting regions for target task.

Considering that promising regions in big-data systems are usually non-convex and discontinuous, the promising region extraction problem is modeled as a classification problem, i.e., predicting whether a configuration is within the promising regions of the configuration space. A Support Vector Machine (SVM) classifier is employed to learn the promising regions for each source task. The SVM classifier is trained with performance observations $H_t^i = \{\boldsymbol{c}_j^i, f(\boldsymbol{c}_j^i)\}_{j=1}^{t}$ collected in the ith source task. The training samples $\{\boldsymbol{c}_j^i, l_j^i\}_{j=1}^{t}$ need to be re-labeled, and the rule for labeling is as follows:

$$l_j^i = \begin{cases} 1 & \text{if } f(\boldsymbol{c}_j^i) < f_+^i, \\ 0 & \text{if } f(\boldsymbol{c}_j^i) \geq f_+^i, \end{cases} \quad (7)$$

where f_+^i is determined by a certain quantile α^i of the performance values in H_t^i, i.e., the Cumulative Distribution Function (CDF) $P(f(\boldsymbol{c}_j^i) < f_+^i) = \alpha^i$. To control the size of the promising regions based on task similarity, the quantile α^i must be related to the task similarity, i.e.,

$$\alpha^i = \alpha_{min} + (1 - 2 \cdot max(S(\mathcal{M}^i; H_t^T) - 0.5, 0)) \cdot (\alpha_{max} - \alpha_{min}), \quad (8)$$

where α_{min} and α_{max} control the range of α^i. When the task similarity is higher, α^i is closer to α_{min} so that compact promising regions can be extracted. Conversely, when the task similarity is lower, α^i is closer to α_{max}. In extreme cases, promising regions should approximate the entire configuration space. Therefore, α_{min} and α_{max} are usually set to constants close to 0 and 1, respectively. This ensures that when the task similarity is relatively extreme, the promising regions can still be extracted according to the above principle.

Reduced Configuration Space Generation. For n SVM classifiers trained for different source tasks, an ensemble approach is used to determine the promising regions of the target task. For a given configuration, if more than half of the classifiers predict that the configuration is within the promising regions, we consider that the configuration is also within the promising regions of the target task. The ensemble model SVM^T is represented as follows:

$$SVM^T(\boldsymbol{c}_j) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} SVM^{s_i}(\boldsymbol{c}_j) \geq \lceil \frac{n}{2} \rceil, \\ 0 & \text{otherwise}. \end{cases} \quad (9)$$

Candidate Configurations Selection. We place TL-CSR after PS-CSR, i.e., using the ensemble model to determine whether the initial candidates selected by PS-CSR are in the promising regions. This runs the risk of having too few filtered candidates to enable effective exploration of the reduced configuration space. We set a threshold value for the number of final candidates. If the remaining candidates can reach this threshold, they are output directly; otherwise, a supplementary sampling will be performed until the number of candidates is satisfied.

Actually, the promising configurations are close to each other and possibly locate at a few promising regions (Zhu and Liu, 2019). Therefore, we prioritize sampling around remaining candidates in the supplementary sampling. Considering r ($r < T$) candidates remaining in a certain iteration and a final threshold of T. We use an iterative method to complete the candidates. First, $(T-r)/10$ configurations are randomly selected among the remaining candidates, and then 10 configurations are randomly sampled in a certain area around each selected candidate. The boundary is set to the minimum value of the distance between any two configurations among the remaining candidates. If the $T - r$ configurations sampled in the first round are all predicted to be within the promising regions, the supplementary sampling is finished. Otherwise, the above sampling process continues until the required number is reached.

3.5. Overall algorithm of configuration space reduction

Algorithm 2 demonstrates the CSR process in Algorithm 1. This CSR process performs in each iteration since the important parameters and task similarities are constantly changing during the iterations. Lines 1–7 conduct PS-CSR, while lines 8–26 perform TL-CSR.

Specifically, the importance of each parameter is calculated based on Gini importance (lines 1–3), and the important parameters are selected by taking the top-k parameters in order of Gini importance (line 4). LHS is adopted to exploit the knowledge about important parameters (line 5), while RS is applied to explore the whole configuration space to avoid adverse effects caused by the wrong choice of important parameters (line 6). The numbers of sampled configurations for RS and LHS are determined by Eqs. (3) and (4), and vary during the iterations. The configurations obtained from LHS and RS are all initial candidate configurations (line 7).

For TL-CSR, lines 8–10 calculate the similarity between each source and target based on Eq. (5), and line 11 samples n similar sources based on Eq. (6). Then, the promising region extractions of selected sources are conducted in lines 12–14. The promising regions for each source are determined by an SVM classifier for which training samples are prepared according to Eq. (7). An ensemble model with voting through several classifiers is used to represent the promising regions of the target task (line 15).

To select candidates for base tuner, we first determine whether the initial candidates generated in line 7 are in the promising regions of the target by the predictions of the ensemble model generated in line 15 (line 16). If the number of remaining configurations reaches a preset threshold T, these configurations are output directly; otherwise, a supplementary sampling is performed until the threshold is satisfied (lines 17–26).

4. Evaluation

We implement ETune and other algorithms, and conduct extensive experiments to evaluate the superiority of our approach. The source code and the data can be found in the online repository: https://doi.org/10.5281/zenodo.8161973.

Algorithm 2 Configuration space reduction

```
Input: $\mathcal{M}^i$: the RF model of $i$-th source task, $i = 1, \cdots, N$; $\mathcal{M}^T$: the RF model of target task; $H^T$: the performance observations in target task; $C$: the configuration space defined by all $m$ parameters; $k$: the number of important parameters; $n^l$: the number of LHS; $n^r$: the number of RS; $T$: the threshold of candidate configurations.
Output: $candidates$: the candidate configurations in $\hat{C}$.
1: for $i = 1$ to $m$ do
2:     $importance_i \leftarrow Gini\_importance(C, \mathcal{M}^T)$;
3: end for
4: $importance\_parms \leftarrow top{-}\mathrm{k}(C, importance_i, k)$;
5: $confs_{exploit} \leftarrow LHS(importance\_parms, n^l)$;
6: $confs_{explore} \leftarrow RS(C, n^r)$;
7: $candidates_I \leftarrow add(confs_{exploit}, confs_{explore})$;
8: for $i = 1$ to $N$ do
9:     $task\_similarity_i \leftarrow S(\mathcal{M}^i; H^T)$;
10: end for
11: sample $n$ similar source tasks $s_1, \cdots, s_n$ based on Equation (6);
12: for $i = 1$ to $n$ do
13:     train an SVM classifier $SVM^{s_i}$ for $s_i$ based on the training samples prepared by Equation (7);
14: end for
15: generate the ensemble model $SVM^T$ for target task based on Equation (9);
16: $candidates \leftarrow \{c_i | c_i \in candidates_I, SVM^T(c_i) = 1\}$;
17: while $|candidates| < T$ do
18:     $n_m \leftarrow T - |candidates|$;
19:     $bound \leftarrow min\_distance(candidates)$;
20:     $sampled\_confs \leftarrow RS(candidates, n_m/10)$;
21:     for $i = 1$ to $|sampled\_confs|$ do
22:         $confs_a \leftarrow RS(sampled\_confs_i, bound)$;
23:         $candidates \leftarrow add(candidates, confs_a)$;
24:     end for
25:     $candidates \leftarrow \{c_i | c_i \in candidates, SVM^T(c_i) = 1\}$;
26: end while
27: return candidates;
```

4.1. Research questions

Our experiment investigates the following research questions (RQs).

RQ1: How effective and efficient is ETune compared with the state-of-the-art baselines?

RQ2: Do the two CSR methods help the base tuner improve tuning efficiency?

RQ3: Can the CSR methods be combined with other transfer learning based tuners to further improve the tuning efficiency?

We study **RQ1** to evaluate how efficiently ETune could recommend better or equally good configurations than other advanced approaches. We investigate **RQ2** to verify whether PS-CSR and TL-CSR methods can help ETune efficiently find high-performance configurations in target task by identifying promising regions. We use **RQ3** to demonstrate the interoperability of the two CSR methods, and further verify its effectiveness.

4.2. Experimental methodology

Study Subject, Cluster, Parameters, and Performance Metrics. We choose Spark (version 2.2.1) as the study subject because it has been used in a wide range of domains including machine learning, streaming computing, database management, etc (Yu et al., 2018). Our experiments are conducted on a cluster of three cloud servers, one serves as the master node and the other two serve as slave nodes. Each server is equipped with four 4-core Intel(R) Xeon(R) CPU E5-2682 v4 @2.50 GHz processors, 32 GB RAM, and a 100 GB disk.

Table 2
Experimented workloads in this study.

Application	Abbr.	Input data size (DS)
Sort	ST	10, 15, 20 (GB)
Wordcount	WC	30, 50 (GB)
Terasort	TS	3, 5, 7 (GB)
Bayes	BS	0.1, 0.5 (million pages)
Pagerank	PR	0.7, 1 (million pages)

The 30 configuration parameters for tuning in this work are consistent with those selected by Tuneful (Fekry et al., 2020) since those parameters represent a superset of parameters used in related work (Yu et al., 2018; Zhu et al., 2017) and cover almost all internal aspects of Spark. In addition, we use execution time as the performance metric for big-data systems, due to the fact that execution time directly affects the user experience.

Baseline Algorithms. To evaluate the performance of ETune, we compare it with two state-of-the-art tuners, namely, ResTune (Zhang et al., 2021b) and Tuneful (Fekry et al., 2020). Both the two tuners can leverage the tuning experience from the previous tuning tasks and transfer knowledge to speed up the tuning process of target task. ResTune selects several similar source tasks and combines the surrogate models of selected source tasks into a weighted ensemble model, which reduces the tuning time by a meta-learning based approach. Tuneful first identifies the important parameters by a multi-round sensitivity analysis and subsequently tunes only the important parameters. In the tuning phase, a similar source task is selected and multitask GP is employed to share the tuning samples across similar workloads. It is worth mentioning that workload characterization is out of the scope of this paper. For a fair comparison, we remove the workload characterization part from the baseline algorithms and measure the workload similarity by rank correlation, consistent with this paper.

Workload. In big-data software systems, a workload is an application deployed on the cloud and its input (Hsu et al., 2018b). We choose five applications with different characteristics to evaluate the performance of ETune. The applications are chosen from the well known big data benchmarks Hibench (Huang et al., 2010): (1) Sort: an application that sorts its text input data. (2) Wordcount: a text analysis application that counts word occurrences. (3) Terasort: a numeric data sorter. (4) Bayes: an application that builds a Bayesian classification model. (5) Pagerank: a graph analytics workload that ranks the influence of graph vertices. Moreover, we employ two or three different sizes of input datasets for each application. The workloads are shown in Table 2.

Evaluation Metrics. The performance (i.e., execution time) and tuning cost are used to evaluate the tuning effectiveness and efficiency, respectively.

Tuning effectiveness is evaluated by the execution time of the tuned configurations by ETune and the baselines. Our target is to obtain tuned configurations similar to what state-of-the-art tuners achieve. In order to minimize the measurement noise of the performance in our experiments, the servers are dedicated to perform the tuning experiments for spark. The tuning experiments for different workloads are performed sequentially to avoid the fluctuation of system performance in measurements (Zhang et al., 2021a). In addition, we repeat each performance measurement three times and return the average performance obtained from the three measurements to the tuners, thus providing reliable performance data (Dorn et al., 2020).

The performance improvement (PI) of an algorithm over the baseline in comparison is defined as:

$$PI_{baseline} = \frac{P_{baseline} - P}{P_{baseline}} \cdot 100, \tag{10}$$

where $P_{baseline}$ is the performance of the baseline, and P is that of the algorithm being evaluated. The objective here is to evaluate the tuning effectiveness that different tuning approaches can achieve.

Tuning cost aims to evaluate the tuning efficiency, that is, a tuning approach that can obtain close-to-optimal configurations significantly faster than the others is better. Previous work validates the fact that the time for performance evaluation dominates the tuning time, thus it is reasonable to focus on the number of iterations N_I required for each tuner to obtain good configurations (Zhang et al., 2021b). To further measure the efficiency of different tuners, we also compare tuning time, which is the sum of the workload execution time T needed by the tuner. It is noteworthy that the tuning time T does not only depend on the number of iterations N_I, but also on the recommended configurations in tuning process, as evaluating a bad configuration results in a slow execution of the workload (Fekry et al., 2020). To fairly compare the tuning efficiency of different algorithms, we compare the tuning cost to find the optimal configuration $\boldsymbol{c}^*$ in baseline $TC_{baseline}$ and the tuning cost to find a configuration better or comparable to $\boldsymbol{c}^*$ in the tuning algorithm being evaluated TC. The cost reduction (CR) of an algorithm over the baseline in comparison is calculated in the same way as PI.

Experiment Outline. To support the evaluation of tuners that leverage past tuning experience, a data repository needs to be constructed before experiments begin. We run base tuner for each workload to collect the performance observations in data repository. To answer **RQ1**, the execution time of the tuned configuration and tuning cost of ETune are compared with those of state-of-the-art tuners (i.e., ResTune and Tuneful). Next, we design and run experiments to answer **RQ2**. To verify the validity of the two CSR methods, we compare the tuning effectiveness and efficiency of three tuning approaches: (1) base tuner without CSR, (2) base tuner with PS-CSR, (3) base tuner with PS-CSR and TL-CSR. For **RQ3**, we combine existing advanced tuners with our CSR methods and compare the experimental results with the original tuners to verify the interoperability of our methods.

In our experiments, we select any one of the 12 workloads as the target task and the remaining 11 workloads as the source tasks. The tuning budget of each tuning experiment is 150 performance observations, among which 50 performance observations are used as initial samples. We use the same initial samples in all algorithms for fair comparisons. Moreover, we run ten times of each experiment and demonstrate the average result. To statistically compare the performance of different approaches under the same test case, we first conduct normality tests using the Shapiro–Wilk test and homogeneity of variance tests using Levene's test for each set of data that needs to be compared. If the data follows a normal distribution and the variances are equal, we use a Student's t-test; if the data follows a normal distribution but the variances are not equal, we use a Welch's t-test; if the data does not follow a normal distribution, we use a Mann–Whitney U test. The significance level is set at 0.05 for all tests.

4.3. Results (RQ1): Comparison with advanced baselines

We evaluate the tuning effectiveness and efficiency of ETune in various workloads. Fig. 2 shows the normalized execution time of current best configuration recommended by different tuning approaches during the iterations. The y-axis represents the execution time normalized to the corresponding execution time of default configuration for each workload.

In addition, the normalized average execution time and tuning costs of Tuneful, ResTune and ETune over different workloads are listed in Table 3. We statistically compare the performance and tuning time of the three methods by means of significance tests. For each workload, the best algorithm is selected by performing a significance test on the results of ten repeated runs of the experiment, and the corresponding results are marked in gray. If there is no gray marking in a workload, it means that the algorithm with the best average performance in that test case cannot statistically outperform all the other algorithms.

In terms of tuning effectiveness, we observe that ETune is able to find better or comparable configurations compared to existing tuning algorithms in different workloads. For example, the PIs of ETune compared to ResTune and Tuneful reach to 10.79% and 13.17% under TS-DS2 workload respectively.

Tuning efficiency can be measured by the reduction of tuning cost. There are two ways of considering tuning cost, one is the number of iterations required to find the optimal configuration, and the other is the accumulated workload execution time required in this process. We show both two tuning costs in Table 3. When calculating the tuning costs we exclude the cases where the optimal configuration found by an algorithm in a particular test case is in the initial samples. This is due to the fact that all algorithms in the same test case use the same initial samples, it is meaningless to compare tuning time in this case.

The experimental results show that ETune is able to find the optimal configuration with fewer iterations in most test cases (9/12). Meanwhile, ETune achieves the lowest tuning time among the three algorithms in all test cases. We observe that in some test cases (e.g., ST-DS2, BS-DS2, and PR-DS1), ETune tends to have lower tuning time, even though the baseline algorithms are able to find comparable configurations with fewer iterations. This indicates that ETune recommends more "safe" configurations during the tuning process.

We denote the cost reduction in tuning time as CR^t. Compared with Tuneful and ResTune, the average CR^t of Etune are 52.14% and 46.21% under different workloads, respectively. When calculating the average cost reduction, we remove the test cases in which the baseline algorithm ultimately recommends a configuration in the initial samples. For example, ETune finds a configuration that is 27.07 s faster than the one found by ResTune for TS-DS2, while the tuning time is reduced by 11,090.65 s. Most notably, the tuning time of Tuneful is 82.4% longer than that of ETune under ST-DS2 workload, although Tuneful finds the optimal configuration 16 iterations faster than ETune. This means that although the number of iterations for ETune to find a better configuration is larger than Tuneful, the cumulative execution time for ETune to find that configuration is less. This also validates our conclusion above that ETune is able to perform safer exploration in configuration space by generating promising regions during the search process and avoid exploring configurations with poor performance. Considering that the definition of CR^t is closer to the actual application scenario, we subsequently discuss the tuning cost considering only the tuning time.

To summarize, Tuneful can only leverage the tuning experience of one similar workload, so Tuneful has a greater risk of negative transfer than ETune and ResTune, and is slightly less effective and efficient at tuning. In contrast, both ETune and ResTune can extract knowledge from several similar workloads to improve tuning efficiency, in fact in our experiments ETune and ResTune select the same similar workloads. ResTune trains surrogate models using performance observations from similar workloads and builds a meta-learner to improve tuning efficiency by ensemble these surrogate models, the weight of each surrogate model is determined by the task similarity. ETune, on the other hand, trains classifiers using performance observations of similar workloads and corresponding task similarities, and uses these classifiers vote to identify promising regions of the target task to accelerate configuration tuning by reducing the configuration space. Due to the expensive cost of performance evaluations, the number of performance observations for each source task in our experiments is relatively limited (i.e., 150), and training surrogate models has higher requirements on the number and quality of samples than training classifiers. Therefore, the surrogate model trained in ResTune may not be accurate enough compared to the binary classifier trained in ETune, which will lead to a slightly worse tuning efficiency of ResTune than ETune.

4.4. Results (RQ2): Validity of configuration space reduction

The above results demonstrate that ETune is able to recommend better or comparable configurations compared to two state-of-the-art baselines, while reducing tuning time by 46.21%–52.14% on average.

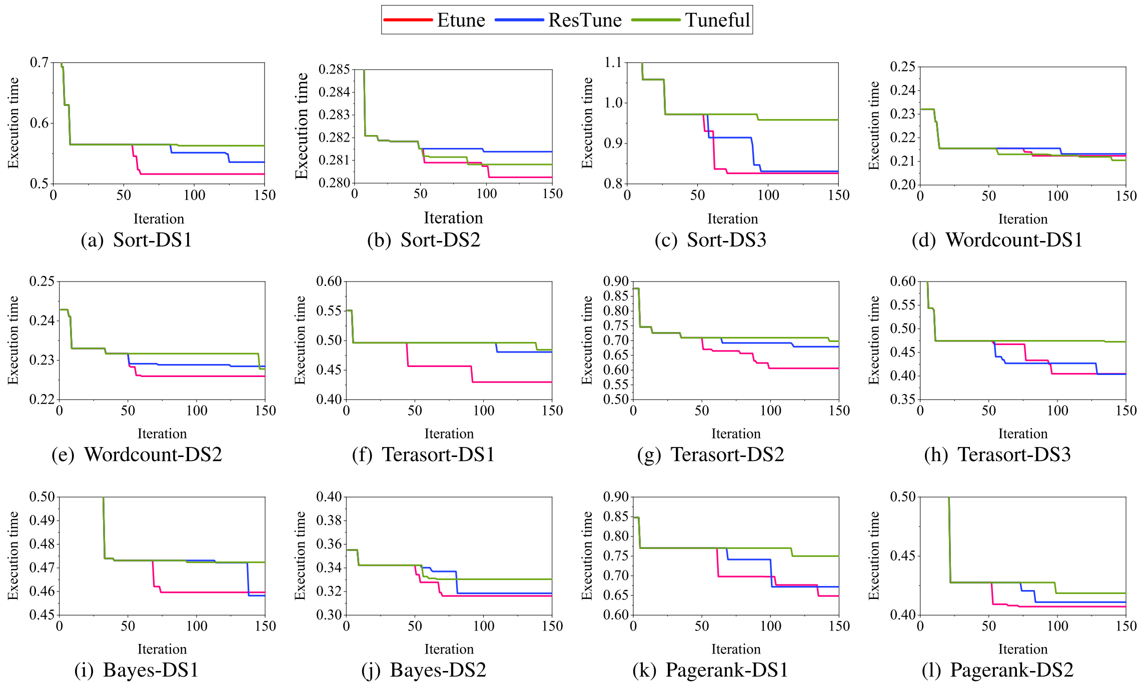

Fig. 2. Tuning evaluation — normalized execution time of current best configuration recommended by different tuners during the iterations for different workloads (normalized to the execution time of default configuration, lower is better).

Table 3
Normalized execution time and tuning costs of Tuneful, ResTune and ETune over different workloads. Execution time and tuning time are normalized to the corresponding execution time of Tuneful and tuning time of ETune, respectively.

Workload	Normalized execution time			# of iteration			Normalized tuning time		
	Tuneful	ResTune	Etune	Tuneful	ResTune	Etune	Tuneful	ResTune	Etune
ST-DS1	1	0.952	**0.917**	89	125	**67**	2.374	2.611	**1**
ST-DS2	1	1.002	**0.998**	**86**	98	102	1.824	2.144	**1**
ST-DS3	1	0.867	**0.862**	93	95	**72**	1.955	1.304	**1**
WC-DS1	**1**	1.013	1.009	140	103	**82**	1.766	1.820	**1**
WC-DS2	1	1.003	**0.992**	146	125	**60**	2.831	2.015	**1**
TS-DS1	1	0.992	**0.887**	139	110	**92**	2.237	1.931	**1**
TS-DS2	1	0.973	**0.868**	143	117	**99**	2.049	2.326	**1**
TS-DS3	1	**0.855**	0.857	135	129	**96**	1.927	1.783	**1**
BS-DS1	1	**0.970**	0.973	93	138	**74**	1.850	2.312	**1**
BS-DS2	1	0.964	**0.957**	**67**	81	70	2.129	1.620	**1**
PR-DS1	1	0.896	**0.865**	116	**101**	135	1.946	1.468	**1**
PR-DS2	1	0.982	**0.973**	99	84	**73**	2.728	1.797	**1**

However, the magnitude of the effect of PS-CSR and TL-CSR on the performance of ETune is still unclear. To gain a deeper insight, we compare the tuning effectiveness and efficiency of the following three tuning approaches: (1) base tuner (denoted as Base), (2) base tuner with PS-CSR (denoted as Base+PS), (3) ETune that combines the base tuner with PS-CSR and TL-CSR. If we turn off the TL-CSR in ETune, we will get the Base+PS algorithm. Moreover, a base tuner is used in this experiment, it replaces the candidate configuration selection of PS-CSR, which trades off exploitation and exploration based on parameter importance, with random sampling.

A summary of tuning effectiveness and efficiency of Base, Base+PS and ETune is presented in Table 4. We use significance tests with the significance level 0.05 to compare the performance and tuning time of the three methods, the best result in each workload is marked in gray. These results show that these three tuners recommend configurations with comparable performance, with ETune finding slightly better configurations than the Base and Base+PS algorithms in some test cases.

If the Base algorithm searches for a configuration with better performance compared to Base+PS, the tuning time of the Base+PS algorithm cannot be computed in a strict sense. Considering that the difference in performance achieved between the two algorithms is very small, for

Table 4
Normalized execution time and tuning time of Base, Base+PS and ETune over different workloads. All execution time and tuning time are normalized to the corresponding execution time and tuning time of Base algorithm.

Workload	Normalized execution time			Normalized tuning time		
	Base	Base+PS	ETune	Base	Base+PS	ETune
ST-DS1	1	0.987	**0.968**	1	1.054	**0.841**
ST-DS2	1	0.991	**0.981**	1	1.012	**0.776**
ST-DS3	1	0.953	**0.935**	1	1.038	**0.937**
WC-DS1	1	1.019	**0.955**	1	**0.742**	0.945
WC-DS2	1	1.011	**0.967**	1	0.473	**0.458**
TS-DS1	1	1.003	**0.935**	1	0.893	**0.692**
TS-DS2	1	0.989	**0.913**	1	0.515	**0.323**
TS-DS3	1	1.002	**0.987**	1	0.973	**0.824**
BS-DS1	1	0.981	**0.947**	1	0.711	**0.685**
BS-DS2	1	1.015	**0.980**	1	0.876	**0.541**
PR-DS1	1	1.010	**0.982**	**1**	1.167	1.072
PR-DS2	1	0.973	**0.958**	1	**0.497**	0.605

Table 5
Normalized execution time and tuning time of ResTune and ResTune+CSR over different workloads.

Workload	Normalized execution time		Normalized tuning time	
	ResTune	ResTune+CSR	ResTune	ResTune+CSR
ST-DS1	**1**	1.002	1	**0.372**
ST-DS2	**1**	1.001	1	**0.621**
ST-DS3	**1**	1.004	1	**0.863**
WC-DS1	1	**0.992**	1	**0.783**
WC-DS2	1	**0.998**	1	**0.684**
TS-DS1	1	**0.970**	1	**0.674**
TS-DS2	1	**0.946**	1	**0.653**
TS-DS3	1	**0.997**	1	**0.669**
BS-DS1	1	**0.996**	1	**0.488**
BS-DS2	1	**0.978**	1	**0.723**
PR-DS1	1	**0.957**	1	**0.873**
PR-DS2	1	**0.952**	1	**0.860**

Table 6
Normalized execution time and tuning time of Tuneful and Tuneful+CSR over different workloads.

Workload	Normalized execution time		Normalized tuning time	
	Tuneful	Tuneful+CSR	Tuneful	Tuneful+CSR
ST-DS1	**1**	1.001	1	**0.983**
ST-DS2	1	**0.994**	1	**0.979**
ST-DS3	1	**0.986**	1	**0.872**
WC-DS1	1	**0.982**	1	**0.545**
WC-DS2	1	**0.993**	1	**0.570**
TS-DS1	1	1.003	1	**0.824**
TS-DS2	1	**0.956**	1	**0.748**
TS-DS3	1	**0.943**	1	**0.890**
BS-DS1	**1**	1.010	1	**0.895**
BS-DS2	1	**0.982**	1	**0.866**
PR-DS1	1	**0.995**	1	**0.683**
PR-DS2	1	1.005	1	**0.924**

these test cases we adjust the tuning time to the cumulative workload execution time for the algorithm to find the optimal configuration. The above adjustment to tuning time for specific test cases helps measure the tuning time required for different algorithms to find comparable configurations.

In terms of tuning efficiency, ETune performs the best among the three tuning approaches in six out of twelve test cases, and Base+PS performs the best in two test cases. On average, ETune outperforms the other two approaches. It reduces the average tuning time by 27.50% and 11.05% compared with Base and Base+PS, respectively. Moreover, it is worth mentioning that although the tuned performance values of Base and Base+PS algorithms are basically equal, Base+PS algorithm is slightly better in terms of tuning efficiency. The above observations indicate that TL-CSR plays a significant role in ETune, and PS-CSR as a supplement to TL-CSR further improves the tuning efficiency and robustness of ETune.

4.5. Results (RQ3): Interoperability

To validate the interoperability of our CSR methods, we integrate our methods to the state-of-the-art tuning approaches (i.e., ResTune and Tuneful) and compare the tuning effectiveness and efficiency between the updated tuners and original tuners. The normalized execution time and tuning time of ResTune and ResTune+CSR under different workloads are shown in Table 5, while the comparison of normalized execution time and tuning time of Tuneful and Tuneful+CSR under different workloads are shown in Table 6. The execution time and tuning time are normalized to the corresponding execution time and tuning time of original algorithms. The best algorithm is selected by performing a significance test on the results of ten repeated runs for each workload with significance level 0.05, and the corresponding results are marked in gray.

Experimental results show that the combination of our CSR methods with other tuners can find comparable configurations more quickly than the original tuners. Specifically, adding the CSR methods results in a 31.12% and 18.57% reduction in tuning time for ResTune and Tuneful, respectively. The above results not only demonstrate the interoperability of our CSR methods, but also its effectiveness in improving tuning efficiency.

4.6. Discussion

(1) ETune is designed to work efficiently both initially (when zero/little tuning data is available) as well as later (as more tuning data is acquired). Initially, we use the base tuner to tune the big-data systems and utilize PS-CSR to shrink the configuration space when starting from scratch. Once the sufficient workload changes have been seen, TL-CSR is added to reduce the configuration space further by transferring knowledge across past tuning tasks, leading to higher tuning efficiency.

Table 7
Top 5 important parameters of Spark with three different workloads.

Workload	BS-DS1	BS-DS2	PR-DS2
Params	executor.cores	executor.instances	default.parallelism
	shuffle.compress	executor.memory	broadcast.compress
	executor.memory	broadcast.blockSize	executor.cores
	serializer	executor.cores	speculation.quantile
	io.compression.codec	kryoserializer.buffer	executor.memory

(2) In TL-CSR, we carefully design how to perform promising region extraction. When a source task is quite similar to the target task, we obtain compact promising regions. In the worst case, when the source task is dissimilar to the target task, we keep larger promising regions (i.e., almost the entire configuration space) to ensure that we do not miss some interesting regions for target task. This guarantees that ETune can still work without suitable available knowledge, avoiding the risk of negative transfer. However, in this case, ETune will fall back to Base+PS, which limits the tuning efficiency. This suggests that if the cloud providers have access to a sufficient amount of tuning data, they will be in the ideal position to offer workload-specific configuration tuning services in the most efficient manner, thereby minimizing costs.

(3) Different workloads (including the applications and input data sizes) are sensitive to different subsets of parameters and respond to each in different ways. We demonstrate this with three workloads: Pagerank and Bayes with two input sizes. The five most important parameters for the three workloads are shown in Table 7.

For each workload, the set of important parameters is different. Even for the same application, different input data sizes have different important parameters. Regarding the sensitivity of important parameters to different workloads, similar findings are found in existing studies (Mühlbauer et al., 2023; Lesoil et al., 2023). This suggests that the parameters' significance needs to be considered in workload-specific configuration tuning. We carefully design PS-CSR to balance the exploitation of important parameters and the exploration of the entire configuration space in the tuning of new workloads, ensuring that important parameters are accurately identified across different workloads and improving tuning efficiency.

(4) Parameter selection for high-dimensional configuration spaces can improve tuning efficiency, but wrong parameter selection will have serious negative impact on the subsequent tuning results. Meanwhile, the limited performance evaluations make it difficult to achieve accurate parameter selection before tuning. Therefore, it is a feasible solution to exploit the important parameters while maintaining a certain degree of exploration of other parameters at the beginning of tuning process.

(5) Tuning approach that utilizes only one source task for transfer learning has a higher risk of negative transfer than tuning approach that utilizes knowledge transferred from multiple source tasks. Therefore, defining a suitable similar tasks selection mechanism and using

multiple source tasks for transfer learning tend to achieve better tuning performance. This makes the cloud provider a suitable player to provide workload-specific configuration tuning, as the cloud provider has easy access to many historical tuning tasks.

(6) Our ETune approach finds comparable configurations faster than other state-of-the-art baseline algorithms. ETune's improvement in tuning efficiency is achieved by limiting the search process to the promising regions of the target task through the CSR methods, avoiding wasting limited performance evaluations in areas of poor performance. This will also have the benefit that "safer" configurations tend to be recommended during tuning process, which gives ETune the potential to be used for online tuning.

(7) Most existing transfer learning based configuration tuning follows the same idea, i.e., learning a surrogate model of the system performance by leveraging the knowledge from past tuning tasks and using that model to guide the search of the configuration space. Our CSR methods in ETune are orthogonal to the previously proposed approaches, using past tuning experience to identify the promising regions and thus improve tuning efficiency. Therefore, these two methods can be easily combined to further accelerate the configuration tuning.

4.7. Threats to validity

Internal validity: Threats to internal validity are factors that may influence our results. To increase the internal validity, we performed controlled experiments by executing each test case ten times and reporting the average of these ten runs. To statistically compare the performance and tuning time of different approaches under the same test case, we first perform normality tests using the Shapiro–Wilk test and homogeneity of variance tests using Levene's test. If the data follows a normal distribution and the variances are equal, we use a Student's t-test; if the data follows a normal distribution but the variances are not equal, we use a Welch's t-test; if the data does not follow a normal distribution, we use a Mann–Whitney U test. The significance level is set at 0.05 for all tests. For each workload, the best algorithm is selected by performing a significance test on the results of ten repeated runs of the experiment. Such a method can avoid the misleading effects of specifically selected test cases and ensure the stability of the result. Nonetheless, more data points will further enhance the internal validity. Moreover, we set the same tuning cost constraint suggested by users (i.e., 150 iterations) for all algorithms in each test case, and each algorithm is forced to stop when the constraint is met. All tuning algorithms in the same test case use the same initial samples for a fair comparison. These results are considered to be fair and reliable. Finally, we have tried multiple hyperparameter values for ETune in our experiments and observed that these good hyperparameter settings are almost the same in each test case and lead to better experimental results. The hyperparameters of baseline algorithms are set as close as possible to the original settings.

External validity: Threats to external validity concern factors that may influence the generalizability of our results. We increased the external validity by choosing five representative applications, including Sort, Wordcount, Terasort, Bayes, and Pagerank. Further, we employ different sizes of input data for each application, an application-input size pair is considered as a workload in this work. The subject system is deployed and used in the real world. Finally, our ETune is a black-box approach and is independent to the study subject, the results of our evaluations are transferable to other high-dimensional software systems.

5. Related work

Configuration Tuning for Big-Data Software Systems. Existing configuration tuning approaches are divided into four categories, namely, rule-based, model-based, search-based, and learning-based. Rule-based approaches assist users in tuning parameters based on the experience of human experts, online tutorials (HadoopTuning, 2015; HadoopTutorial, 2018; Apache Spark Tuning Guide, 2019; Apache Storm Performance Tuning, 2019), or tuning instructions (White, 2012; Nabi et al., 2014; ClouderaSparkTuning, 2018; ClouderaYarnTuning, 2018). Model-based approaches (Herodotou and Babu, 2011; Herodotou et al., 2011; Wang et al., 2012; Shi et al., 2014; Liu et al., 2015; Verma et al., 2011a,b; Khan et al., 2015; Chen et al., 2013; Wang and Khan, 2015; Venkataraman et al., 2016; Zacheilas et al., 2017; Singhal and Singh, 2017) construct analytical models in the early stage of system development. These two approaches are both white-box approaches, which are based on a deep understanding of system internals. In contrast, search-based and learning-based approaches treat the system as a black box.

Search-based methods take various search strategies to explore the configuration space directly. Random search (Bergstra and Bengio, 2012) explores each dimension of parameters uniformly at random. The main idea in Tang (2017) is to search near-optimal configurations with heuristic searching algorithms, such as genetic algorithm (GA) and Markov Chain Monte Carlo (MCMC) strategy. BestConfig (Zhu et al., 2017) employs the divide-and-diverge sampling method and the recursive bound-and-search algorithm to tune system configurations. Different from search-based methods, learning-based approaches employ a variety of machine learning techniques to construct performance models, such as classification and regression trees (CART) (Guo et al., 2013, 2018; Nair et al., 2018a; Sarkar et al., 2015; Valov et al., 2015; Nair et al., 2017; Wang et al., 2016), RF (Bei et al., 2015; Tang, 2017; Bao et al., 2018a), neural networks (Mahgoub et al., 2017; Ha and Zhang, 2019), and GP (Van Aken et al., 2017). Furthermore, several search strategies adopt various performance models to find optimal configurations, including GA (Bei et al., 2015; Yu et al., 2018) and hill climbing algorithm (Zacheilas et al., 2018). In addition, BO with a GP has emerged as a powerful black-box optimization framework. There are several BO-based approaches using this adaptive sampling process to solve the configuration tuning problem (Thummala and Babu, 2010; Van Aken et al., 2017; Cereda et al., 2021; Fekry et al., 2020; Li et al., 2018a; Dalibard et al., 2017; Jamshidi and Casale, 2016; Nair et al., 2018b; Trotter et al., 2017; Fischer et al., 2015).

Configuration Space Reduction for Configuration Tuning. One important way of CSR is parameter selection. Most of existing tuners reduce the configuration space using domain knowledge (Krishna et al., 2020; Tang, 2017; Nair et al., 2018b; Bei et al., 2015; Bao et al., 2018; Zhu et al., 2017; Bao et al., 2019b; Thummala and Babu, 2010; Cereda et al., 2021), whereas several efforts attempt to automatically select important parameters (Van Aken et al., 2017; Fekry et al., 2020; Mahgoub et al., 2017; Yeh et al., 2016; Bindal et al., 2020; Debnath et al., 2008; Cao et al., 2020; Kanellis et al., 2020; Li et al., 2018b; Lima et al., 2018; Liu et al., 2020). For example, OtterTune (Van Aken et al., 2017) applies Lasso to select important knobs for databases. Tuneful (Fekry et al., 2020) identifies workload-specific influential parameters using a multi-round sensitivity analysis method.

Moreover, another way of CSR is to combine a black-box tuner (usually BO) and a white-box model. Structured Bayesian optimization (SBO) (Dalibard et al., 2017) allows developers to provide contextual information in the form of a probabilistic model. Following in with the same philosophy, Guided Bayesian Optimization (GBO) (Kunjir, 2020) supplements a BO with an approximate white-box model. Specifically, GBO recommends a configuration to evaluate next which both maximizes the acquisition function and is expected to perform well according to the white-box model as well. The introduction of white-box models accelerates the black-box optimizer, but requires a deep understanding of the system. Therefore this approach tends to tune only a small number of parameters and may require reconstructing the white-box model as the workload changes.

Similarity-Aware Configuration Tuning. Most approaches perform configuration tuning for a specific workload such that the tuning has to start from the scratch once the workload changes, which is expensive and inefficient (Jamshidi et al., 2017). Instead, similarity-aware configuration tuning across workloads has become a hot area of research in recent years. Tuneful (Fekry et al., 2020) and SimTune (Fekry et al., 2020a) combines workload characterization and Multitask BO to accelerate finding near-optimal configurations. Similarly, OtterTune (Van Aken et al., 2017; Zhang et al., 2018) starts the configuration tuning by reusing the data from the similar previous workload to train a GP model.

The above three approaches identify one most similar workload from its repository. In contrast, ResTune (Zhang et al., 2021b) uses a meta-learning approach designed to leverage the experience of multiple previous tuning tasks, thus accelerate the tuning process of the new tasks. The knowledge extracted from multiple historical tuning tasks is represented by multiple base models, and an ensemble model motivated by RGPE (Feurer et al., 2018) is employed to combine the knowledge.

Most studies on similarity-aware configuration tuning follow the same idea, i.e., learning an surrogate model of the system by leveraging the knowledge from past tuning tasks and using that model to guide the search of the configuration space. Our CSR methods are orthogonal to the previously proposed approaches, using past tuning experience to effectively highlight the promising configuration space and thus improve tuning efficiency.

6. Conclusion

In this paper, we proposed an efficient tuner, ETune, which accelerates an BO-based tuner with two configuration space reduction techniques to find close-to-optimal configurations faster under workload change scenarios for big-data software systems. Rather than learning a multi-task surrogate model, ETune aims to automatically design a compact but promising configuration space with the aid of previous tuning tasks. To demonstrate the efficacy of ETune, we conducted comprehensive experiments in diverse workloads. Experimental results show that ETune outperforms two state-of-the-art baselines by 46.21%–52.14% in terms of tuning time on average. Furthermore, we conducted experiments to demonstrate the performance superiority of ETune in terms of tuning time over the base tuner, and base tuner with PS-CSR. Finally, we verified the interoperability of our CSR methods, the experimental results demonstrate that our methods significantly speed up the existing tuning approaches by a 18.57%–31.12% reduction in tuning time on average.

CRediT authorship contribution statement

Rong Cao: Conceptualization, Methodology, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Software, Visualization. **Liang Bao:** Methodology, Writing – review & editing, Funding acquisition, Resources, Supervision, Project administration. **Kaibi Zhao:** Methodology, Formal analysis, Investigation, Software, Validation. **Panpan Zhangsun:** Software, Validation, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the National Natural Science Foundation of China [Grant No. 62172316]; the Ministry of Education Humanities and Social Science Project of China [Grant No. 17YJA790047]; the Soft Science Research Plans of Shaanxi Province, China [Grant No. 2020KRZ018]; the Research Project on Major Theoretical and Practical Problems of Philosophy and Social Sciences in Shaanxi Province, China [Grant No. 20JZ-25]; the Key R and D Program of Shaanxi, China [Grant No. 2019ZDLGY13-03-02]; the Natural Science Foundation of Shaanxi Province, China, China [Grant No. 2019JM-368]; and the Key R and D Program of Hebei, China [Grant No. 20310102D].

References

Apache Spark Tuning Guide, 2019. Apache spark tuning guide. https://spark.apache.org/docs/latest/tuning.html.

Apache Storm Performance Tuning, 2019. Apache storm performance tuning. https://storm.apache.org/releases/current/Performance.html.

Bao, L., Liu, X., Chen, W., 2018a. Learning-based automatic parameter tuning for big data analytics frameworks. In: 2018 IEEE International Conference on Big Data. Big Data, IEEE, pp. 181–190.

Bao, L., Liu, X., Wang, F., Fang, B., 2019b. Actgan: Automatic configuration tuning for software systems with generative adversarial networks. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 465–476.

Bao, L., Liu, X., Xu, Z., Fang, B., 2018. Autoconfig: Automatic configuration tuning for distributed message systems. In: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 29–40.

Bei, Z., Yu, Z., Zhang, H., Xiong, W., Xu, C., Eeckhout, L., Feng, S., 2015. RFHOC: A random-forest approach to auto-tuning hadoop's configuration. IEEE Trans. Parallel Distrib. Syst. 27 (5), 1470–1483.

Bergstra, J., Bengio, Y., 2012. Random search for hyper-parameter optimization.. J. Mach. Learn. Res. 13 (2).

Bindal, P.B., Singhal, D., Subramanyam, A., Kumar, V., et al., 2020. OneStopTuner: An end to end architecture for JVM tuning of spark applications. arXiv preprint arXiv:2009.06374.

Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32.

Calzarossa, M.C., Massari, L., Tessera, D., 2016. Workload characterization: A survey revisited. ACM Comput. Surv. 48 (3), 1–43.

Cao, Z., Kuenning, G., Zadok, E., 2020. Carver: Finding important parameters for storage system tuning. In: 18th $USENIX$ Conference on File and Storage Technologies ($FAST$ 20). pp. 43–57.

Cereda, S., Valladares, S., Cremonesi, P., Doni, S., 2021. Cgptuner: a contextual Gaussian process bandit approach for the automatic tuning of IT configurations under varying workload conditions. Proc. VLDB Endow. 14 (8), 1401–1413.

Chen, K., Powers, J., Guo, S., Tian, F., 2013. CRESP: Towards optimal resource provisioning for MapReduce computing in public clouds. IEEE Trans. Parallel Distrib. Syst. 25 (6), 1403–1412.

Chen, J., Xu, N., Chen, P., Zhang, H., 2021. Efficient compiler autotuning via bayesian optimization. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, pp. 1198–1209.

ClouderaSparkTuning, 2018. Cloudera performance management - tuning spark applications. https://www.cloudera.com/documentation/enterprise/5-9-x/topics/admin_spark_tuning.html.

ClouderaYarnTuning, 2018. Cloudera performance management - tuning YARN. https://www.cloudera.com/documentation/enterprise/5-8-x/topics/cdh_ig_yarn_tuning.html.

Dalibard, V., Schaarschmidt, M., Yoneki, E., 2017. BOAT: Building auto-tuners with structured Bayesian optimization. In: Proceedings of the 26th International Conference on World Wide Web. pp. 479–488.

Debnath, B.K., Lilja, D.J., Mokbel, M.F., 2008. SARD: A statistical approach for ranking database tuning parameters. In: 2008 IEEE 24th International Conference on Data Engineering Workshop. IEEE, pp. 11–18.

Dorn, J., Apel, S., Siegmund, N., 2020. Mastering uncertainty in performance estimations of configurable software systems. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. pp. 684–696.

Duan, S., Thummala, V., Babu, S., 2009. Tuning database configuration parameters with ituned. Proc. VLDB Endow. 2 (1), 1246–1257.

Fekry, A., Carata, L., Pasquier, T., Rice, A., 2020a. Accelerating the configuration tuning of big data analytics with similarity-aware multitask Bayesian optimization. In: 2020 IEEE International Conference on Big Data. Big Data, IEEE, pp. 266–275.

Fekry, A., Carata, L., Pasquier, T., Rice, A., Hopper, A., 2020. To tune or not to tune? In search of optimal configurations for data analytics. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2494–2504.

Feurer, M., Letham, B., Bakshy, E., 2018. Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles. In: AutoML Workshop At ICML, Vol. 7.

Feurer, M., Springenberg, J.T., Hutter, F., 2014. Using meta-learning to initialize Bayesian optimization of hyperparameters. In: MetaSel@ ECAI. Citeseer, pp. 3–10.

Fischer, L., Gao, S., Bernstein, A., 2015. Machines tuning machines: Configuring distributed stream processors with bayesian optimization. In: 2015 IEEE International Conference on Cluster Computing. IEEE, pp. 22–31.

Guo, J., Czarnecki, K., Apel, S., Siegmund, N., Wasowski, A., 2013. Variability-aware performance prediction: A statistical learning approach. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 301–311.

Guo, J., Yang, D., Siegmund, N., Apel, S., Sarkar, A., Valov, P., Czarnecki, K., Wasowski, A., Yu, H., 2018. Data-efficient performance learning for configurable systems. Empir. Softw. Eng. 23 (3), 1826–1867.

Ha, H., Zhang, H., 2019. Deepperf: performance prediction for configurable software with deep sparse neural network. In: 2019 IEEE/ACM 41st International Conference on Software Engineering. ICSE, IEEE, pp. 1095–1106.

HadoopTuning, 2015. Hadoop performance tuning tutorial. http://hadooptutorial.info/hadoop-performance-tuning/.

HadoopTutorial, 2018. Hadoop MapReduce tutorial. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

Helton, J.C., Davis, F.J., 2003. Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. Reliab. Eng. Syst. Saf. 81 (1), 23–69.

Hennig, P., Schuler, C.J., 2012. Entropy search for information-efficient global optimization.. J. Mach. Learn. Res. 13 (6).

Herodotou, H., Babu, S., 2011. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. Proc. VLDB Endow. 4 (11), 1111–1122.

Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F.B., Babu, S., 2011. Starfish: A self-tuning system for big data analytics.. In: Cidr, Vol. 11, No. 2011. pp. 261–272.

Hoffman, M., Brochu, E., De Freitas, N., et al., 2011. Portfolio allocation for Bayesian optimization. In: UAI. pp. 327–336.

Hsu, C.-J., Nair, V., Freeh, V.W., Menzies, T., 2018a. Arrow: Low-level augmented bayesian optimization for finding the best cloud vm. In: 2018 IEEE 38th International Conference on Distributed Computing Systems. ICDCS, IEEE, pp. 660–670.

Hsu, C.-J., Nair, V., Menzies, T., Freeh, V.W., 2018b. Scout: An experienced guide to find the best cloud configuration. arXiv preprint arXiv:1803.01296.

Huang, S., Huang, J., Dai, J., Xie, T., Huang, B., 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops. ICDEW 2010, IEEE, pp. 41–51.

Jamshidi, P., Casale, G., 2016. An uncertainty-aware approach to optimal configuration of stream processing systems. In: 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. MASCOTS, IEEE, pp. 39–48.

Jamshidi, P., Siegmund, N., Velez, M., Kästner, C., Patel, A., Agarwal, Y., 2017. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 497–508.

Jones, D.R., Schonlau, M., Welch, W.J., 1998. Efficient global optimization of expensive black-box functions. J. Global Optim. 13 (4), 455–492.

Kanellis, K., Alagappan, R., Venkataraman, S., 2020. Too many knobs to tune? towards faster database tuning by pre-selecting important knobs. In: 12th *USENIX* Workshop on Hot Topics in Storage and File Systems. HotStorage 20.

Khan, M., Jin, Y., Li, M., Xiang, Y., Jiang, C., 2015. Hadoop performance modeling for job estimation and resource provisioning. IEEE Trans. Parallel Distrib. Syst. 27 (2), 441–454.

Krishna, R., Tang, C., Sullivan, K., Ray, B., 2020. ConEx: Efficient exploration of big-data system configurations for better performance. IEEE Trans. Softw. Eng. 48 (3), 893–909.

Kunjir, M., 2020. Speeding up AutoTuning of the memory management options in data analytics. Distrib. Parallel Databases 38 (4), 841–863.

Lesoil, L., Acher, M., Blouin, A., Jézéquel, J.-M., 2023. Input sensitivity on the performance of configurable systems an empirical study. J. Syst. Softw. 201, 111671.

Li, Z.L., Liang, C.-J.M., He, W., Zhu, L., Dai, W., Jiang, J., Sun, G., 2018a. Metis: Robustly tuning tail latencies of cloud systems. In: 2018 *USENIX* Annual Technical Conference (*USENIX ATC* 18). pp. 981–992.

Li, Y., Shen, Y., Jiang, H., Bai, T., Zhang, W., Zhang, C., Cui, B., 2022. Transfer learning based search space design for hyperparameter tuning. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. KDD '22, Association for Computing Machinery, New York, NY, USA, pp. 967–977. http://dx.doi.org/10.1145/3534678.3539369.

Li, T., Shi, S., Luo, J., Wang, H., 2018b. A method to identify spark important parameters based on machine learning. In: International Conference of Pioneering Computer Scientists, Engineers and Educators. Springer, pp. 525–538.

Lima, M.I.V., de Farias, V.A., Praciano, F.D., Machado, J.C., 2018. Workload-aware parameter selection and performance prediction for in-memory databases. In: SBBD. pp. 169–180.

Liu, J., Tang, S., Xu, G., Ma, C., Lin, M., 2020. A novel configuration tuning method based on feature selection for hadoop MapReduce. IEEE Access 8, 63862–63871.

Liu, C., Zeng, D., Yao, H., Hu, C., Yan, X., Fan, Y., 2015. Mr-cof: a genetic mapreduce configuration optimization framework. In: International Conference on Algorithms and Architectures for Parallel Processing. Springer, pp. 344–357.

Mahgoub, A., Wood, P., Ganesh, S., Mitra, S., Gerlach, W., Harrison, T., Meyer, F., Grama, A., Bagchi, S., Chaterji, S., 2017. Rafiki: A middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference. pp. 28–40.

Menze, B.H., Kelm, B.M., Masuch, R., Himmelreich, U., Bachert, P., Petrich, W., Hamprecht, F.A., 2009. A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data. BMC Bioinformatics 10 (1), 1–16.

Mühlbauer, S., Sattler, F., Kaltenecker, C., Dorn, J., Apel, S., Siegmund, N., 2023. Analyzing the impact of workloads on modeling the performance of configurable software systems. In: Proceedings of the International Conference on Software Engineering. ICSE, IEEE.

Nabi, Z., Bouillet, E., Bainbridge, A., Thomas, C., 2014. Of streams and storms. IBM White Paper, Citeseer, pp. 1–31.

Nair, V., Menzies, T., Siegmund, N., Apel, S., 2017. Using bad learners to find good configurations. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 257–267.

Nair, V., Menzies, T., Siegmund, N., Apel, S., 2018a. Faster discovery of faster system configurations with spectral learning. Autom. Softw. Eng. 25 (2), 247–277.

Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S., 2018b. Finding faster configurations using flash. IEEE Trans. Softw. Eng. 46 (7), 794–811.

Rasmussen, C.E., 2003. Gaussian processes in machine learning. In: Summer School on Machine Learning. Springer, pp. 63–71.

Sampaio, A.R., Beschastnikh, I., Maier, D., Bourne, D., Sundaresen, V., 2023. Auto-tuning elastic applications in production. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice. ICSE-SEIP, pp. 355–367.

Sarkar, A., Guo, J., Siegmund, N., Apel, S., Czarnecki, K., 2015. Cost-efficient sampling for performance prediction of configurable systems (t). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 342–352.

Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N., 2015. Taking the human out of the loop: A review of Bayesian optimization. Proc. IEEE 104 (1), 148–175.

Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., Wang, C., 2014. Mrtuner: a toolkit to enable holistic optimization for mapreduce jobs. Proc. VLDB Endow. 7 (13), 1319–1330.

Singhal, R., Singh, P., 2017. Performance assurance model for applications on SPARK platform. In: Technology Conference on Performance Evaluation and Benchmarking. Springer, pp. 131–146.

Srinivas, N., Krause, A., Kakade, S.M., Seeger, M., 2010. Gaussian process optimization in the bandit setting: No regret and experimental design. In: Proceedings of the 27th International Conference on Machine Learning. ICML-10, June 21-24, 2010, Haifa, Israel.

Sullivan, D.G., Seltzer, M.I., Pfeffer, A., 2004. Using probabilistic reasoning to automate software tuning. ACM SIGMETRICS Perform. Eval. Rev. 32 (1), 404–405.

Tang, C., 2017. System performance optimization via design and configuration space exploration. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 1046–1049.

Thummala, V., Babu, S., 2010. iTuned: a tool for configuring and visualizing database parameters. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. pp. 1231–1234.

Trotter, M., Liu, G., Wood, T., 2017. Into the storm: Descrying optimal configurations using genetic algorithms and bayesian optimization. In: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems. FAS* W, IEEE, pp. 175–180.

Valov, P., Guo, J., Czarnecki, K., 2015. Empirical comparison of regression methods for variability-aware performance prediction. In: Proceedings of the 19th International Conference on Software Product Line. ACM, pp. 186–190.

Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B., 2017. Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 1009–1024.

Venkataraman, S., Yang, Z., Franklin, M., Recht, B., Stoica, I., 2016. Ernest: Efficient performance prediction for large-scale advanced analytics. In: 13th *USENIX* Symposium on Networked Systems Design and Implementation (*NSDI* 16). pp. 363–378.

Verma, A., Cherkasova, L., Campbell, R.H., 2011a. Aria: automatic resource inference and allocation for mapreduce environments. In: Proceedings of the 8th ACM International Conference on Autonomic Computing. pp. 235–244.

Verma, A., Cherkasova, L., Campbell, R.H., 2011b. Resource provisioning framework for mapreduce jobs with performance goals. In: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing. Springer, pp. 165–186.

Wang, K., Khan, M.M.H., 2015. Performance prediction for apache spark platform. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems. IEEE, pp. 166–173.

Wang, K., Lin, X., Tang, W., 2012. Predator—An experience guided configuration optimizer for hadoop MapReduce. In: 4Th IEEE International Conference on Cloud Computing Technology and Science Proceedings. IEEE, pp. 419–426.
Wang, G., Xu, J., He, B., 2016. A novel method for tuning configuration parameters of spark based on machine learning. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications. IEEE, pp. 586–593.
White, T., 2012. Hadoop: The Definitive Guide. O'Reilly Media, Inc..
Wistuba, M., Schilling, N., Schmidt-Thieme, L., 2015. Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, pp. 104–119.
Yeh, C.-C., Zhou, J., Chang, S.-A., Lin, X.-Y., Sun, Y., Huang, S.-K., 2016. Bigexplorer: A configuration recommendation system for big data platform. In: 2016 Conference on Technologies and Applications of Artificial Intelligence. TAAI, IEEE, pp. 228–234.
Yu, Z., Bei, Z., Qian, X., 2018. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 564–577.
Zacheilas, N., Maroulis, S., Kalogeraki, V., 2017. Dione: Profiling spark applications exploiting graph similarity. In: 2017 IEEE International Conference on Big Data. Big Data, IEEE, pp. 389–394.
Zacheilas, N., Maroulis, S., Priovolos, T., Kalogeraki, V., Gunopulos, D., 2018. Dione: A framework for automatic profiling and tuning big data applications. In: 2018 IEEE 34th International Conference on Data Engineering. ICDE, IEEE, pp. 1637–1640.
Zaouk, K., Song, F., Lyu, C., Sinha, A., Diao, Y., Shenoy, P., 2019. Udao: A next-generation unified data analytics optimizer. Proc. VLDB Endow. (PVLDB) 12 (12).
Zhang, X., Chang, Z., Li, Y., Wu, H., Tan, J., Li, F., Cui, B., 2021a. Facilitating database tuning with hyper-parameter optimization: a comprehensive experimental evaluation. arXiv preprint arXiv:2110.12654.
Zhang, B., Van Aken, D., Wang, J., Dai, T., Jiang, S., Lao, J., Sheng, S., Pavlo, A., Gordon, G.J., 2018. A demonstration of the ottertune automatic database management system tuning service. Proc. VLDB Endow. 11 (12), 1910–1913.
Zhang, X., Wu, H., Chang, Z., Jin, S., Tan, J., Li, F., Zhang, T., Cui, B., 2021b. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In: Proceedings of the 2021 International Conference on Management of Data. pp. 2102–2114.
Zhu, Y., Liu, J., 2019. Classytune: A performance auto-tuner for systems in the cloud. IEEE Trans. Cloud Comput..
Zhu, Y., Liu, J., Guo, M., Bao, Y., Ma, W., Liu, Z., Song, K., Yang, Y., 2017. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 338–350.

Rong Cao is currently pursuing the Ph.D. degree in software engineering with the School of Computer Science and Technology, Xidian University, China. Her main research interests include performance optimization of software systems and machine learning.

Liang Bao received the Ph.D. degree in computer science from Xidian University, P.R. China, in 2010. He is currently a professor with the School of Computer Science and Technology, Xidian University. His research interests include software architecture, cloud computing and big data.

Kaibo Zhao is currently a PhD. candidate in College of computer science and technology at Xidian University, China, under the supervision of Prof Bao Liang. He received his bachelor's degree from Xi'an University of Architecture and Technology in 2019. His research interests include image detection, image segmentation, and image generation.

Panpan Zhangsun received the master degree in college of Computer Science and Technology at Xidian University, china in 2023. Her research interests include software system performance prediction, software system parameter optimization, etc.