Towards Dynamic and Safe Configuration Tuning for Cloud Databases

Xinyi Zhang†‡
Peking University & Alibaba Group
zhang_xinyi@pku.edu.cn

Hong Wu†
Alibaba Group
hong.wu@alibaba-inc.com

Yang Li∗
Peking University
liyang.cs@pku.edu.cn

Jian Tan†
Alibaba Group
j.tan@alibaba-inc.com

Feifei Li†
Alibaba Group
lifeifei@alibaba-inc.com

Bin Cui∗‡§
Peking University
bin.cui@pku.edu.cn

ABSTRACT
Configuration knobs of database systems are essential to achieve high throughput and low latency. Recently, automatic tuning systems using machine learning methods (ML) have shown to find better configurations compared to experienced database administrators (DBAs). However, there are still gaps to apply the existing systems in production environments, especially in the cloud. First, they conduct tuning for a given workload within a limited time window and ignore the dynamicity of workloads and data. Second, they rely on a copied instance and do not consider the availability of the database when sampling configurations, making the tuning expensive, delayed, and unsafe. To fill these gaps, we propose OnlineTune, which tunes the online databases safely in changing cloud environments. To accommodate the dynamicity, OnlineTune embeds the environmental factors as context feature and adopts contextual Bayesian Optimization with context space partition to optimize the database adaptively and scalably. To pursue safety during tuning, we leverage the black-box and the white-box knowledge to evaluate the safety of configurations and propose a safe exploration strategy via subspace adaptation. We conduct evaluations on dynamic workloads from benchmarks and real-world workloads. Compared with the state-of-the-art methods, OnlineTune achieves 14.4%–165.3% improvement on cumulative performance while reducing 91.0%–99.5% unsafe configuration recommendations.

CCS CONCEPTS
• Information systems → Autonomous database administration; • Computing methodologies → Machine learning.

KEYWORDS
online tuning; cloud database; availability

1 INTRODUCTION
Modern database management systems (DBMS) have dozens of configurable knobs that control the runtime behaviors [15] and impact database performance such as throughput and latency. Finding an optimal configuration for a given workload is proven to be an NP-hard problem [56] and it is impossible to enumerate and evaluate all the combinations. To find promising configurations, database administrators (DBAs) put considerable effort into tuning the configurations based on their experience. In addition, the workloads from real-world applications are dynamic, which means the properties of the workloads vary over time. Figure 1 (a) shows a dynamic workload trace from a real-world application. Similarly, the underlying data size and distribution constantly change due to the data modification operations. In Figure 1 (b), the size of underlying data increases almost three times after running TPC-C workload for 33 hours. The dynamicity of the environment makes configuration tuning more difficult.

Many recent studies focus on automatic configuration tuning for DBMSs using Machine Learning (ML) techniques, including Gaussian Process Regression from OtterTune [6], and Deep Deterministic Policy Gradient from CDBTune [72] and QTune [40]. These ML-based tuning systems can efficiently sample more promising configurations than DBAs; that is, they could find near-to-optimal configurations with fewer trials. They utilize ML models to suggest promising configurations and update the models by evaluating the configurations iteratively. To achieve this, they first need to copy the target DBMS instance to either replay or forward the workload. In each iteration, the tuner suggests a configuration and applies it to the copied DBMS. Then, the workload is executed to evaluate the performance metrics of the suggested configuration, and the model is updated based on the evaluation. This process repeats until the models converge or stop conditions are met (e.g., budget limits).

Unfortunately, there are several major problems when deploying these systems in production, especially in the cloud. All the existing
In this paper, we propose OnlineTune, an online tuning system that tunes the databases safely and adaptively in the constantly changing cloud environment. For dynamicity, we formulate the online tuning problem into a contextual bandit problem. OnlineTune features the context and optimizes the database performance over a context-configuration joint space. We propose a clustering and model section strategy to scale up OnlineTune with accumulated observations. OnlineTune clusters the observations, fits multiple models, and selects the appropriate one for a given context. For safety, OnlineTune safely explores the configuration space. It evaluates the safety of configurations by leveraging both the black-box knowledge (i.e., posterior estimate from the model) and white-box knowledge (i.e., heuristics-based rules from the domain experience). Since satisfying the safety constraint in the continuous and high-dimension space is non-trivial, we transform the high-dimension optimization problem into a sequence of subspace problems that can be solved efficiently. Each model maintains a configuration subspace centered around the best configuration found so far. OnlineTune starts from configurations similar to those known to be safe and expands the subspace to facilitate further explorations. Specifically, we make the following contributions:

- To address the challenges in real DBMS scenarios with the dynamic workload, we define the online tuning problem and solve it as a contextual bandit problem with safety constraints. To the best of our knowledge, OnlineTune is the first online configuration tuning system for DBMS with safety consideration.
- We propose a context featurization model that extracts features of workloads and underlying data. Using this, OnlineTune adopts the contextual Bayesian optimization technique to optimize the database adaptively with constantly changing environments.
- To enhance the scalability of OnlineTune with extensive data in the cloud, we propose a clustering and model selection strategy that significantly decreases the computation complexity.

We propose OnlineTune, an online tuning system that tunes the databases safely and adaptively in the constantly changing cloud environment. For dynamicity, we formulate the online tuning problem into a contextual bandit problem. OnlineTune features the context and optimizes the database performance over a context-configuration joint space. We propose a clustering and model selection strategy to scale up OnlineTune with accumulated observations. OnlineTune clusters the observations, fits multiple models, and selects the appropriate one for a given context. For safety, OnlineTune safely explores the configuration space. It evaluates the safety of configurations by leveraging both the black-box knowledge (i.e., posterior estimate from the model) and white-box knowledge (i.e., heuristics-based rules from the domain experience). Since satisfying the safety constraint in the continuous and high-dimension space is non-trivial, we transform the high-dimension optimization problem into a sequence of subspace problems that can be solved efficiently. Each model maintains a configuration subspace centered around the best configuration found so far. OnlineTune starts from configurations similar to those known to be safe and expands the subspace to facilitate further explorations. Specifically, we make the following contributions:

- To address the challenges in real DBMS scenarios with the dynamic workload, we define the online tuning problem and solve it as a contextual bandit problem with safety constraints. To the best of our knowledge, OnlineTune is the first online configuration tuning system for DBMS with safety consideration.
- We propose a context featurization model that extracts features of workloads and underlying data. Using this, OnlineTune adopts the contextual Bayesian optimization technique to optimize the database adaptively with constantly changing environments.
- To enhance the scalability of OnlineTune with extensive data in the cloud, we propose a clustering and model selection strategy that significantly decreases the computation complexity.
• To solve the safety issue, we combine the black-box and the white-box knowledge to evaluate the safety of configurations and propose a safe exploration strategy via subspace adaptation, largely reducing the risks of applying harmful configurations.

• We implement the proposed method and evaluate on dynamic workloads from benchmarks and the real-world application. Compared with the state-of-the-art techniques, ONLINE-TUNE achieves 14.4%–165.3% improvement on cumulative improvement while decreasing 91.0%–99.5% unsafe recommendations.

The remainder of the paper is organized as follows. We review the related works in Section 2 and formally define the online tuning problem in Section 3. A system overview of ONLINE-TUNE is presented in Section 4, followed by a description of our techniques for contextual performance modeling in Section 5, safe configuration recommendation in Section 6, Section 7 presents our experimental evaluation. Finally, we conclude in Section 8.

2 RELATED WORK

Configuration Tuning. There has been an active area of research on tuning configurations of DBMS, which can be summarized as:

• Rule-based. Rule-based methods recommend configurations using heuristic rules. Database vendors develop tuning tools to provide DBA with knobs recommendations through identifying database’s bottlenecks due to misconfigurations [12] or asking the DBA questions about their application [66]. Wei et al. propose to generate fuzzy rules for database tuning [65]. BestConfig [75] searches configurations based on several heuristics. Rule-based methods strongly depend on the assumptions of their heuristics and fail to utilize knowledge gained from previous tuning efforts.

• Learning-based. Tuned [15], Otterture [6] and ResTune [73] use Bayesian Optimization (BO) based method, modeling the tuning as a black-box optimization problem. Reinforcement Learning (RL) is adopted in [40, 72] to tune DBMS by learning a neural network between the internal metrics and the configurations. In the field of data analytics systems, ReM [36] studies the problem of tuning the memory allocation and develops an empirically-driven algorithm. And Tuneful [18, 19] combines incremental Sensitivity Analysis and BO to prune configuration space. Although RL methods can adapt to workloads by fine-tuning its neural network, additional time and evaluation samples are needed. All the above methods train a machine learning model to learn the offline tuning policy and cannot promptly respond to the dynamic environment. Besides, they do not consider the safety constraints when interacting with the database, which restricts them from being deployed in production.

Query Featurizing. Query featurizing aims at translating plain SQLs into their vectorized representations. We summarize it into SQL text parsing and logical plan parsing based on the input. SQL text parsing directly processes the query texts. TF-IDF is used to represent a query as a collection of the weighted frequencies of its individual word tokens [45, 73]. However, its unbounded vocabulary makes generalization across workloads difficult. Therefore ResTune restricts its calculation to reserved SQL keywords [73]. To support larger vocabulary, [29, 76] resort to representation learning which is frequently used in NLP. Representation learning produces dense vectors capturing nuanced relationships of unstructured data. They use deep models, e.g., LSTM or CNN, to learn the distributional embeddings. Logical plan parsing parses a query plan to aggregate key features, including the cost or categories of operators, scanned tables, and predicates. It has been adopted by many state-of-the-art works in the field of query-performance prediction or cardinality estimation [4, 5, 23, 32, 37, 47, 51, 57]. QTune encodes query type, involved tables, query operators and the corresponding costs to predict the internal metrics of a database [40].

Bayesian Optimization. Our algorithm falls under the general umbrella of Bayesian optimization (BO). It learns and optimizes a black-box function over configuration space [53]. BO works iteratively: (1) updating the surrogate model that describes the relationship between configurations and their performances and (2) choosing the next configuration to evaluate by computing the acquisition function value. The acquisition function measures the utility of candidate points for the next evaluation by trading off the exploration of uncertain areas and exploiting promising regions.

BO has been extensively used in many scenarios, including hyperparameter tuning [9, 14, 30, 41, 42, 67], experimental design [22] and controller tuning [11, 20, 21, 46]. Contextual BO considers the environmental conditions by augmenting the GP kernel with extra context variables and uses CGP – UCB to select promising action [34]. DBA Bandit [52] chooses a set of indices from finite and discrete configuration space based on the context of indexed columns and derived statistics from database optimizer. It achieves an $O(\sqrt{n})$ regret bound after playing $n$ rounds as a safety guarantee, implying that the per-step average cumulative regret approaches zero after sufficiently many steps. However, guaranteeing the safety of every step is still challenging but vital for mission-critical applications. Recently, Constrained BO is proposed to optimize a black-box function with unknown constraint [24, 25, 39]. But the constraint is not considered safety-critical, and the algorithm is allowed to evaluate unsafe parameters. The main instance of safe optimization is SAFEOPT algorithm [55]. However, its formulation relies on a discretization of configuration space, which hinders high-dimensional applications. Meanwhile, scaling up BO with high dimensions is another active area. Recent works propose local modeling [16, 33, 63] and space partitioning [49, 64], which achieve strong empirical results in high dimensional problems. Motivated by the advances, ONLINE-TUNE refines a configuration subspace that can be discretized efficiently.

3 PROBLEM STATEMENT

Consider a database system with a continuous configuration space $\Theta = \Theta_1 \times \Theta_2 \times \ldots \times \Theta_m$ and a context space $C$. The context $c \in C$ is uncontrollable environmental conditions, e.g., dynamic workloads. We denote the database performance metrics as $f$, which can be any chosen metric to be optimized, such as throughput, 99%th percentile latency, etc. Given a configuration $\theta$ and context $c$, the corresponding performance $f(\theta, c)$ can be observed only after evaluation.

As we discussed, the online tuner should satisfy two requirements: dynamicaly and safety. The dynamicaly requirement means the tuner should consider changing environmental conditions (context) when recommending the configurations. We formalize it as a contextual bandit problem [34], where at each iteration $t$, the tuner receives context $c_t$ and outputs a configuration $\theta_t$ to
maximize the payoff (i.e., database performance $f$). The safety requirement indicates that we additionally need to ensure that, for each tuning iteration $t$, $f_t \geq \tau$ holds, where $f_t = f(\theta_t, c_t)$ and $\tau \in \mathbb{R}$ is a specific safety threshold. We define the configurations that satisfy the above condition to be safe. The safety threshold indicates the degree of risk end-users can tolerate when adopting the tuning approach. As the database is a mission-critical system, the service level agreement must be guaranteed by the cloud providers under the vendor default configuration [59, 61]. Intuitively, the safety threshold is set to the database performance under the default configuration (denoted as default performance). In case the default performance fluctuates with workload changes, we assume that the default performance for any given workload can be acquired. Note in practice, the default performance can be easily obtained or predicted when a historical knowledge base is available (such as in [6, 18, 73]). For example, the user could train a regression model that inputs the context and outputs the default performance. Even without any previous knowledge, we can take some time to observe the default performance. At the very beginning, without any assumptions about $f$, searching the safe configurations is a nonsensical task. Hence, we also assume that, before the optimization, we are given an initial safety set that contains at least one safe configuration. Assuming that the objective is a maximization problem, the online tuning problem to solve for $c_t \in C$ is:

$$\arg\max_{\theta_t \in \Theta} f(\theta_t, c_t),$$

subject to $f(\theta_t, c_t) \geq \tau$. \hfill (1)

4 OVERVIEW OF ONLINETUNE

Workflow. To conduct online tuning, ONLINETUNE first queries the default configuration and its performance to build an initial safety set. The safety threshold is set to the default performance, as discussed in Section 3. Then, ONLINETUNE functions iteratively. It adapts to the dynamic environment by featurizing the context, recommending a promising configuration $\theta_t$ at the beginning of an iteration, and evaluating its performance during the iteration to update the model. The interval size (i.e., time for one iteration) controls the granularity of ONLINETUNE’s adaptation to the dynamicity. Given a tuning time, ONLINETUNE can collect more observations and make more fine-grained suggestions with a small interval size. We use a three-minute interval by default and conduct sensitivity analysis in our experiments. Within one iteration, the workflow of ONLINETUNE forms an iterative cycle, as presented by ①-⑦ in Figure 2. The workflow consists of two stages: contextual performance modeling and safe configuration recommendation.

Contextual performance modeling aims at obtaining a surrogate model that predicts the performance of given configurations in dynamic environments. The surrogate is a contextual Gaussian Process (GP) model fitted on historical observations (or the initial safety set, if no historical observations exist). ① ONLINETUNE first captures the dynamic factors (e.g., workload and its underlying data) through context featurization, obtaining a context (see Section 5.1 for details). ② It then selects a contextual GP model fitted on the observations from the cluster with similar contexts. The clusters are periodically re-clustered in an offline manner (Section 5.3). ⑦ The construction of contextual GP is introduced in Section 5.2.

Safe configuration recommendation aims at selecting a safe and promising configuration from the configuration space. To avoid the aggressive exploration of BO, especially the over-exploration of boundaries, ONLINETUNE reduces the optimization over the whole configuration space into a sequence of subspace optimization. All operations in this stage are restricted in the subspace where the safe optimization problem can be discretized and solved efficiently. In essence, the subspace is centered around the best configuration estimated so far, gradually moving towards the optimal. This stage inputs the selected surrogate model and its corresponding subspace. ③ For a newly fitted model, ONLINETUNE initializes a subspace centered around the best-estimated configuration. Otherwise, ONLINETUNE adapts the subspace according to the tuning history, e.g., expands the subspace when making consecutive successes (Section 6.1). ④ The adapted subspace is discretized to build a candidate set. ONLINETUNE assesses the safety of the candidates based on the model’s lower bound estimate, forming a safety set. It also consults the white box to dismiss unsafe configurations. In case the heuristic white box excludes the optimal configurations from the safety set, ONLINETUNE relaxes inappropriate rules (Section 6.2.2). ⑤ ONLINETUNE then selects a configuration from the safety set by either maximizing the acquisition function or exploring the safe boundaries of subspace. ⑥ Finally, ONLINETUNE applies the configuration to the online database and evaluates its performance.

Architecture. The main parts of ONLINETUNE are deployed in a backend tuning cluster (ONLINETUNE sever), while the context
workloads. We choose LSTM, which have been used successfully for the percentage of rows filtered by table conditions in queries, the final hidden state on the encoder network provides a dense DBMS optimizer: (1) estimate of rows to be examined by queries, related to the tuning policy, we use the following features from the database data is a non-trivial task [26, 69, 70]. Based on the observation that only the data changes affecting the workload queries are linked to the tuning policy, we use the following features from the DBMS optimizer: (1) estimate of rows to be examined by queries, (2) the percentage of rows filtered by table conditions in queries, whether an index is used. The first two features are queries’ cardinality estimation, capturing the effective changes in data size and distribution. The last feature indicates the index building/dropping operation. We average the three features of queries, obtaining the underlying data feature of a workload.

5 CONTEXTUAL PERFORMANCE MODELING

OnlineTune responds to dynamic environments when tuning online DBMSs. It augments the GP kernel with context variables to learn the relationship between (context, configuration), and database performance. We first discuss how to featurize context and construct a contextual GP model. Then, we present a clustering and model selection strategy to enhance the model’s scalability.

5.1 Context Featurization

Context featurization aims to capture uncontrollable dynamic factors, which affects the relationship between configurations and database performances. When tuning the database, the workload and underlying data are constantly and continuously changing due to the upstream applications and the DML statements (e.g., insert, delete and update). OnlineTune featurizes the two factors to adapt to the dynamicity.

5.1.1 Workload Featurization. We now illustrate how to featurize the changing workload. There are two dynamic aspects of workloads [43]: (1) query arrival rate: the number of arriving queries per second can fluctuate. (2) query composition: the types of queries may change, and the ratio of queries composition may vary.

Query arrival rate can be encoded by one dimension. For query composition, we need to translate the plain queries into vectorized representations. We adopt representation learning techniques [8] to extract informative encoding of queries and generalize across workloads. We choose LSTM, which have been used successfully for SQL query analysis [28, 29, 76]. We use a standard LSTM encoder-decoder network [58] to ease the burden of collecting labeling data. The final hidden state on the encoder network provides a dense encoding for the query. Lastly, we average the query encoding, obtaining the queries composition feature of a workload.

5.1.2 Underlying Data Featurization. Learning the distribution of database data is a non-trivial task [26, 69, 70]. Based on the observation that only the data changes affecting the workload queries are related to the tuning policy, we use the following features from the DBMS optimizer: (1) estimate of rows to be examined by queries, (2) the percentage of rows filtered by table conditions in queries,

\[ \mu_t(\theta, c) = k^T \left( K + \sigma^2 I \right)^{-1} y_{1:t}, \]

\[ \sigma_t^2(\theta) = k(\theta, c), (\theta, c) - k^T \left( K + \sigma^2 I \right)^{-1} k, \] 

(2)
where $k = [k((\theta_1, c_1), (\theta, c)), \ldots, k((\theta_l, c_l), (\theta, c))]^T$ and $K$ is a covariance matrix whose $(i, j)$th entry is $K_{i,j} = k((\theta_i, c_i), (\theta_j, c_j))$. The kernel $k((\theta, c), (\theta', c'))$ should model the distances between points of configuration and context. Concretely, we construct an additive kernel $k_{\theta}(\theta, \theta') + k_c(c, c')$. We use a linear kernel $k_c(c, c')$ to model the dependence on contexts and a Martin kernel $k_{\theta}(\theta, \theta')$ to model the nonlinear performance on configurations. Intuitively, such design could model overall trends according to the context, and the configuration-specific deviation from this trend [34].

The composite kernel implies that function values are correlated when configurations and contexts are similar [10]. We expect the same configuration across correlated contexts to have similar performance predictions. The correlations between contexts can significantly speed up the tuning. Figure 3 shows a simple scenario with a one-dimension context feature and a one-dimension configuration. Even though the algorithm has only explored the configuration space at the first context ($z = 0$, left figure), the correlation between the functions generalizes information to the unobserved context ($z = 0.1$, central figure). The knowledge transfer improves data efficiency and reduces the number of required evaluations.

5.3 Clustering and Model Selection
The contextual GP can model the database performance in dynamic environments. However, it has $O(n^3)$ complexity with $n$ observations. The cubical computation complexity limits the applicability with increasing observations in the cloud. To tackle this problem, we propose a clustering and model selection strategy based on the similarity of context features. The observations are clustered, and the number of observations in each cluster can be bounded under a constant number $P$. OnlineTune fits multiple contextual GPs based on the clusters and learns a decision boundary for model selection. Therefore the complexity is bounded by $O(P^3)$, which makes OnlineTune can scale with increasing observations.

Algorithm 1 presents the procedure. We first perform the DBSCAN clustering algorithm [17] based on context features \{$c_i\}_l$, obtaining a cluster label $l_i$ for each $c_i$ (Line 2), as shown in Figure 4 (a). For each cluster, OnlineTune fits a contextual GP model using its observations (Line 3). To select a model for unseen contexts, OnlineTune uses SVM to learn a non-linear decision boundary (Line 4, as shown in Figure 4 (b)). We choose SVM for its simplicity, ease of training, and the need for fewer samples to generalize well in practice [62]. Besides improving scalability, such clustering excludes the observations with distant contexts from the training set for the GP model, preventing the “negative transfer” [35].

Augmented observations are classified into a cluster based on their context and the learned boundary. However, the distribution of context features may shift as more observations are collected, and the previous clustering and boundary need to be re-learned periodically. OnlineTune maintains the existing clustering and a simulated new clustering to determine whether to re-learn or not (Line 1). The difference between the two kinds of clusterings indicates context shifts. We use a mutual information-based score (MI) to quantify the difference. MIs close to zero indicate two vastly dissimilar clusterings, while MIs close to one indicates the opposite. When the MI score is smaller than a threshold (we set 0.5 in the experiments), the re-clustering is triggered, and the boundary and models are updated based on the revised clusters (Line 2–4).

6 SAFE CONFIGURATION RECOMMENDATION
OnlineTune aims to optimize the database performance while ensuring safety throughout the tuning process. Instead of optimizing the performance function over the global configuration space, OnlineTune restricts its optimization in a subspace and gradually expands it towards the optimum. The subspace restriction enables a fine discretization to generalize safety and mitigates the over-exploiting nature of BO. This section introduces how OnlineTune adapts the subspace, forms a safety set, and selects configurations.

6.1 Subspace Adaptation
To expand towards the global optimum, the configuration subspace is adjusted iteratively and alternated between hypercube and line regions, as shown in Algorithm 2.

The hypercube region $\{\theta \mid \|\theta - \theta_{\text{best}}\|_2 \leq R_h\} \cap \Theta$ is defined with a region center $\theta_{\text{best}}$ and a radius $R_h$. The region center $\theta_{\text{best}}$ is set to the best configuration observed so far. The radius $R_h$ controls the optimization space. If $R_h$ is large enough for the hypercube region to contain the whole configuration space, this will be equivalent to running standard global BO, while a small $R_h$ may slow down the exploration. At initialization, $R_h$ is set to a base value (e.g., 5% ranges of each dimension). It is typical behavior to shrink $R_h$ after consecutive “failures” and expand it after consecutive “successes” [50]. We define a “success” as recommending a configuration better than the previous one and a “failure” as a recommendation that does not. The details about adjusting $R_h$ are illustrated in Line 4–7.

The line region is a one-dimensional affine subspace $\{\theta_{\text{best}} + ad : a \in \mathbb{R}\} \cap \Theta$, defined with the offset $\theta_{\text{best}}$ and a direction $d \in \mathbb{R}^m$. It is proved that the optimization with line regions can converge globally [33]. The direction of the line region determines the trace of optimization. We implement two strategies to generate the directions: random direction (increasing the exploration) and important direction (aligned with the important knob, increasing exploitation). Our technical report [74] contains the details.
The subspace is first initialized as a hypercube region to restrict the optimization near the initial safety set (Lines 1–2). The hypercube region encourages optimization densely in the interior but may lead to an over-exploitation, even getting trapped in a local optimum [60]. OnlineTune switches to line region to control the trade-off between exploitation and safe exploration (Lines 3–13). The alternation is triggered by a switching rule (Lines 8 and 12): when no unevaluated safe configuration exits in $\Theta_{n}$ or a certain number of consecutive failures to recommend better configurations, OnlineTune switches to another type of subspace. The update of $\theta_{\text{best}}$ moves the subspace towards the optimum.

6.2 Safety Assessment

OnlineTune discretizes the adapted subspace to build a candidate set. Then the safety of each candidate is assessed based on the confidence bounds of the contextual GP (black-box knowledge) and the existing domain knowledge (white-box knowledge).

6.2.1 Black-Box Knowledge. Given context $c$, OnlineTune utilizes the confidence bounds of the selected contextual GP model $m^{n}$ to access the safety of $\theta$:

\[
\begin{align*}
l_{n}(\theta, c) &= \mu_{n}(\theta, c) - \beta \sigma_{n}(\theta, c), \\
u_{n}(\theta, c) &= \mu_{n}(\theta, c) + \beta \sigma_{n}(\theta, c),
\end{align*}
\]

where $l_{n}$ and $u_{n}$ are the lower and upper bound predictions, with $\mu_{n}$ and $\sigma_{n}$ from Equation 2. The parameter $\beta$ controls the tightness of the confidence bounds, and we set its value following the study of Srinivas et al. [54]. The true function value $f$ falls into the confidence interval $[l_{n}(\theta, c), u_{n}(\theta, c)]$ with a high probability. We can determine safe configurations with $l_{n}(\theta, c) > r$; that is, configurations with worst-case performance still above the safety threshold $r$. We restrict the performance modeling in a local subspace containing previously evaluated configurations, and the local modeling can be trusted more than global modeling. This is also the rationale behind the trust-region methods from stochastic optimization [71].

6.2.2 White-Box Knowledge. Although the relationship between configurations and database performances is complex. Domain knowledge does exist to dismiss bad configurations. For example, the total buffer size cannot exceed the physical memory capacity of the deployed machine. Experienced DBA tunes the databases based on domain knowledge, and some database tuning tools also use heuristics to give tuning suggestions [3, 12, 65, 66]. Such white-box tuning provides intuitive suggestions and could serve as a warm starting and space pruning component for ML-based tuning. When forming the final safety set, OnlineTune consults the white box and dismisses the unsafe or unpromising configurations deviated from the white box’s suggestions. We implement OnlineTune’s white-box assistant using MysqlTuner [3]. MysqlTuner examines the DBMS metrics and uses static heuristics to suggest setting ranges for configurations. The ranges are generated based on rules (e.g., setting key buffer size larger than the total MyISAM indexes size or increasing the join buffer size if #joins without indexes per day is larger than 250). OnlineTune removes the configurations not satisfying the white box’s suggestions from its safety set. If the white box suggests a specific configuration instead of the ranges, OnlineTune filters the configurations far away from the suggested one. Other white-box tuning tools or hand-crafted rules can coexist with OnlineTune if they provide suggestions for OnlineTune’s tuning knobs. Instead of directly applying, their suggestions can be used as white-box assistants for OnlineTune.

However, the white box does not evolve according to the feedback, causing the trap in local optimum. The rules could even be inappropriate, excluding the optimal configurations from $S_{0}$. This happens only when the white-box rule rejects a configuration while the black-box algorithm recommends this configuration (i.e., decision conflict). To prevent this, OnlineTune uses a relaxation strategy to relax inappropriate rules. OnlineTune maintains a conflict counter and a conflict-safe counter for each rule. When the decision conflict happens several times, reaching a threshold, OnlineTune will ignore the rule and recommend the controversial configuration. Note that only one rule can be ignored in the recommendation to control the interdependence between different rules. After evaluation, if the configuration is safe, the conflict-safe counter will increase by one. When a rule’s conflict-safe counter reaches a threshold, the rule will be relaxed (e.g., the configuration range given by the rule is enlarged). The thresholds for each white-box rule can be set differently according to its credibility. A larger threshold leads to more trust in the white-box rule.

6.3 Candidate Selection

After the safety set is generated, OnlineTune selects a configuration from the candidates in the safety set. Like all the black-box optimization, the selection should trade-off between two objectives: (1) exploitation, trying to localize the high-performance regions within the current safety set, (2) exploration, acquiring new knowledge, and trying to expand the current estimate of the safety set. We adopt Upper Confidence Bound (UCB) [54] constrained to the
safety set as a sampling criterion, shown in Equation 4:

$$\theta_{\text{max}} = \arg \max_{\theta \in S_n} \mu_{\theta}(c) + \beta \sigma_{\theta}(c).$$

(4)

UCB selects the configuration at locations where the upper bound of its confidence interval is maximal. Repeatedly evaluating the system performance at configurations given by UCB improves the mean estimate of the underlying function and decreases the uncertainty at candidate configurations for the maximum. The global maximum is provably found eventually [54].

To expand the safe subspace explicitly, OnlineTune also selects the safe configurations at the boundary of the safety set with the highest uncertainty since they are promising candidates for expanding the safety set. To unify the two sampling criteria, we adopt the epsilon-greedy policy [48], which selects the maximal UCB configuration with a probability of $1 - \epsilon$ and the boundary point with a possibility of $\epsilon$.

7 EVALUATION

Outline: We first compare OnlineTune with the state-of-the-art database configuration tuning methods on dynamic environments in Section 7.1. The comparison is conducted under three settings: (1) dynamic queries composition, where the queries composition of the workload is constantly changing, (2) transactional-analytical, simulating a daily transactional-analytical workload cycle by alternating the execution of OLTP and OLAP workloads, and (3) real-world workload. In Section 7.1.4, we analyze the overhead of the compared methods. In Section 7.2, a case study is presented. In Section 7.3, we conduct ablation studies to evaluate the effectiveness of OnlineTune’s design, including the context space design and the safe exploration strategy. We also validate the robustness of OnlineTune on different initial safety sets and interval sizes. In addition, we evaluate the baselines under static workload setting to analyze the search efficiency in Section 7.4.

Baselines: The baselines are explained below:

- **DBA Default** is the configuration provided by experienced DBAs.
- **OnlineTune** is our safe and contextual tuner. We implement OnlineTune’s prediction model using GPy library [1].
- **BO** is a Bayesian Optimization approach, widely used in database configuration tuning [6, 7, 15, 31]. We use similar design with OtterTune [15]: Gaussian process as surrogate model and EI as acquisition function. We also implemented BO via GPy library.
- **DDPG** is a reinforcement learning agent which is used to tune the database configuration [72]. The agent inputs internal metrics of the DBMS and outputs proper configurations. The DDPG algorithm is implemented using PyTorch library [2] with its neural network architecture borrowed from CDBTune [72].
- **QTune** is a query-aware tuner that supports three tuning granularities [40]. We adopt its workload-level tuning. It embeds workload features to predict internal metrics via a pre-trained model, while CDBTune uses the measured internal metrics.
- **ResTune** adopts constrained Bayesian Optimization to minimize resource utilization with SLA constraints [73]. It uses an ensemble framework (i.e., RGPE) to transfer historical knowledge from observations of source workloads. We modify ResTune to maximize database performance with the same safety constraints as OnlineTune. To adopt RGPE in online tuning, we cluster every 25 observations as one source workload.
- **MysqlTuner** is a MySQL tuning tool that examines DBMS metrics and uses static heuristics to suggest configurations [3]. It is also the white-box assistant that OnlineTune consults.

Workloads: We use three workloads with different characteristics from well-known benchmarks and a real-world workload. The three workloads are TPC-C, Twitter from OLTP-Bench [13] and JOB [38]. TPC-C is a traditional OLTP benchmark characterized by write-heavy transactions with complex relations. Twitter is extracted from web-based applications, characterized by heavily skewed many-to-many relationships and non-uniform access. JOB is an analytical workload with 113 multi-join queries, characterized by realistic and complex joins [38]. We load about 29 GB data for Twitter, 18 GB for TPC-C, 9 GB for JOB and use unlimited arrival rates for OLTP workloads to fully evaluate the benefits from tuning, as [6, 15, 31, 40, 72] did. The real-world workload comes from a database monitoring service. In our experiments, we use the workload from 10:00 to 16:00 on September 2nd, 2021. And the read-write ratio per minute varies from 3:1 ~ 74:1 in this period.

Setup: We use version 5.7 of RDS MySQL. We tune 40 dynamic configuration knobs without restarting the database since the restart is not acceptable for online databases. The 40 configuration knobs are chosen based on their importance by DBAs. The experiments run on a cloud instance with 8 vCPU and 16GB RAM. The interval size is set to 3 minutes. We use the DBA default configuration as the initial safety set and its performance as the safety threshold, which are also added to the training set of other baselines for fairness.

Metrics: We evaluate the baselines from two perspectives: safety and cumulative performance during tuning. For safety, we count the number of unsafe configuration recommendations (#Unsafe) and the number of system failures (#Failure) within the tuning period. For the cumulative performance, it is measured by the number of transactions (#txn) processed by the database during tuning for OLTP workload and the sum of execution time for OLAP workload.

7.1 Evaluation on Dynamic Workflow

We evaluate the baselines tuning online database with constructed dynamic workloads and real-workload workload with dynamic query arrival rates. We intend to answer two questions: (1) Can they recommend the configuration adaptively with the dynamic workload? (2) Can they reliably respect the safety requirement?

7.1.1 Evaluation on workloads with dynamic query compositions. To simulate the dynamicity, we construct workload with dynamic query compositions. For TPC-C and Twitter, we vary transaction weights via OLTP-Bench [13]. The weights are sampled from a normal distribution with a sine function of iterations as mean and a 10% standard deviation. For JOB, we execute ten queries per iteration, and five out of them are re-sampled. If the execution time exceeds the interval size, we kill the queries. Since TPC-C is a write-heavy workload, its underlying data is also changing (e.g., its data size change from 18 GB to 48 GB during tuning).

**OnlineTune finds the workload-specific configuration.** As shown in Figure 5, OnlineTune achieves 54.3% - 93.8% improvement on cumulative performance than the MySQL default, and
16.2% ~21.9% improvement than the DBA default. The DBA default is expected to have a relatively robust performance across workloads. OnlineTune applies better configuration, illustrating its adaptability in dynamic environments. In addition, OnlineTune achieves 14.4%~165.3% improvement on cumulative performance than existing offline approaches. The reasons are two-fold. First, the offline methods struggle to handle dynamic environments. BO uses observations \( \{ \theta, y_i \} \) to fit the GP model, ignoring dynamic environmental factors. When workload drifts, BO fails to learn suitable configurations. ResTune adopts an ensemble GP that assigns different weights to its base models but its base models still ignore the dynamic factors. DDPG and QTune fine-tune the models when environments change, but fine-tuning a neural network needs lots of training samples, which is inefficient for online tuning. Second, the offline methods over-explore configuration spaces. Evaluating unsafe configurations is a part of learning for offline methods. Although ResTune aims at finding the configurations satisfying the constraints, it still needs to evaluate and learn the unsafe area. Compared with the white-box method (MysqlTuner), OnlineTune achieves 10.1%~19.7% improvement. Although MysqlTuner does not have the over-exploration issue, it relies on heuristic rules and traps in local optimum.

OnlineTune reliably respects the safety requirement when tuning the online database. Recommending safe configurations is non-trivial when tuning online databases since the workload changes may cause the shifting of safe configurations. As shown in Figure 5, none of the system failures occur when applying OnlineTune, while offline tuning methods cause several system failures. The offline methods have 22.2%~97.8% unsafe configuration recommendations within the 400 tuning intervals. Compared to them, OnlineTune reduces 91.0%~99.5% unsafe recommendations. This is contributed to the safe exploration strategy of OnlineTune, which we analyze in detail with an ablation study in Section 7.3.2.

7.1.2 Evaluation on Transactional-Analytical Cycle. We simulate a daily transactional-analytical workload cycle by alternating dynamic TPC-C and JOB workloads. We use 99% latency as the optimization objective. The workload characterization of TPC-C (OLTP, write-heavy) and JOB (OLAP, complex joins) are significantly different. Respecting safety constraints is rather tricky in such a scenario. Figure 6 (a) shows the performance of OnlineTune and the DBA default. The workload starts as TPC-C and repeatedly alternates with JOB every 100 iterations. OnlineTune gradually finds configurations better than the DBA default, as shown in the zoom-in plot. When switching to JOB, OnlineTune takes some iterations to explore the configuration space. Meanwhile, several unsafe configurations with latency slightly larger than the safety threshold occur. Then, OnlineTune finds suitable configuration adaptively (e.g., larger sort buffer size for JOB workload). When switching from TPC-C to JOB again, OnlineTune selects the surrogate model fitted with previous observations for JOB and recommends suitable configurations more quickly. OnlineTune achieves performance than the DBA default while respecting the safety requirement, while other approaches fail, as shown in Figure 7 (a).

7.1.3 Evaluation on Real-World Workload. We evaluate on a workload from the real-workload application. Figure 6 (b) shows the tuning process of OnlineTune and Figure 7 (b) presents the performances of all the baselines. We observe that OnlineTune achieves 39.4% improvement on cumulative improvement compared to the DBA default and 25.5%~64.2% improvement compared to offline methods. Although OnlineTune applies several unsafe configurations at the beginning, their performances are within 10% deviation of the default performance as shown in Figure 6 (b).

7.1.4 Algorithm Overhead. Figure 8 shows the computation time when tuning JOB workload. For BO, DDPG and ResTune, the computation time consists of (1) statistics collection, (2) model fitting, and (3) model probe. For OnlineTune and QTune, the time also includes featurization. OnlineTune’s computation time is slightly larger than BO at the beginning. However, BO’s overhead increases exponentially over tuning iteration, as GP suffers from cubic complexity on sample number. Instead, OnlineTune’s computation time is within 3.79 seconds due to its clustering strategy. In addition to the computation time, we also analyze the impact of OnlineTune’s tuning overhead. OnlineTune’s featurization module is deployed in the database instance, while the other parts are deployed in the backend. Only the overhead does not influence the database instance. Therefore, we focus on the impact of featurization. This module takes about 57.7 ms per iteration on average. To measure its impact, we keep the configuration unchanged and compare the resource usages and database performance. We observe that the average CPU usage with featurization is 77.28%, and the one without featurization is 77.27%. The increase is negligible since the featurization time only accounts for a slight proportion of a tuning interval (180 s). Its impact on database performance
can also be overlooked due to marginal resource consumption. For evaluation, we keep configurations unchanged to observe the impact. In practice, the database performance will be improved when running OnlineTune, as evaluated above.

7.2 Case Study
To further investigate OnlineTune’s tuning performance, we conduct a case study tuning five knobs. We construct a workload trace using YCSB with different read/write transaction compositions, as shown in Figure 9. The joint context-configuration space is smaller than the other experiments in the paper. Therefore, we can use extensive evaluations to explore the space and obtain the best configurations (denoted as the Best) and the unsafe areas for each workload composition. As shown in Figure 10, we observe that there are interactions between knobs. And, regular patterns exist among workload compositions, e.g., when buffer pool size (k1) is large and heap table size (k2) is small, the throughput is relatively low. However, the overall effects of knobs are quite different among the workloads, e.g., distinct optimal positions. Thus, it is necessary to consider the workload dynamicity when tuning online.

Figure 11 presents the tuning result, which aligns with the evaluation in Section 7.1. As shown in the iterative plot, the distances between OnlineTune and the Best gradually decrease as OnlineTune safely finds configurations near the optimum. The iterative performances of the offline baselines have large fluctuations due to their trials and errors. In Figure 11, we focus on the configuration values of the top-2 important knobs. The unsafe region is approximated by excluding the knob’s values from all the safe configurations. Note that if a configuration is assigned with a knob value in the unsafe region, the configuration is unsafe, but not vice versa. This is due to knobs interaction may cause unsafe. We observe that the optimal configurations (the Best) are not portable across workloads. OnlineTune applies configurations adaptive to the dynamic workload, which safely proceeds towards the optimum. The other baselines explore the space aggressively in the first 50 iterations and always have a chance to explore the unsafe region.

Figure 13 visualizes the working process of OnlineTune’s modules. From the left figure, we make two observations. First, OnlineTune can select the corresponding model for the observed context. Second, the subspace is initially centered around the default configuration and moves towards the optimum. The configuration subspaces maintained by each model gradually move far from the default, safely exploring the configuration space. From the right figure, we observe that the size of safety set is updated with the augmented observations. If OnlineTune evaluates unsafe configuration, its safety estimation will be immediately tightened, and OnlineTune will recommend conservative configurations near the evaluated-best ones, avoiding successive regression.

7.3 Analysis of OnlineTune
We carefully design OnlineTune with contextual modeling and safe exploration strategy to explore configurations safely in dynamic environments. We evaluate the corresponding designs via ablation study and validate the robustness of OnlineTune.
Figure 9: Pattern of YCSB workload.

Figure 10: Throughput as a function of configurations: k1 denotes sort_buffer_pool_size and k2 denotes max_heap_table_size.

Figure 11: Results on YCSB workload: The left figure shows the cumulative result and the right shows the iterative performance. For clarity of presentation, we only show the iterative performances of the top 3 baselines (OnlineTune, ResTune and BO).

Figure 12: Configurations applied by OnlineTune, ResTune and BO on YCSB workload: We present the top 2 important knobs.

Figure 13: Visualization of OnlineTune. In the left figure, different colors denote different models selected over iterations. The lines denote the distance between the recommended configuration and the default, and the dash denotes the distance between subspace centers and the default. The right figure presents the size of safety set estimated by OnlineTune in its restricted subspace. For reference, we also plot the performance improvement over iterations.

7.3.1 Ablation Study on Contextual Modeling. We remove certain components of contextual modeling (featurization of workload and data changes, clustering, and model selection) in OnlineTune to understand their contribution to the overall system. We compare (1) OnlineTune, (2) OnlineTune-w/o-workload, removing workload feature from context, (3) OnlineTune-w/o-data, removing optimizer statistics (underlying data feature) from context, (4) OnlineTune-w/o-clustering, removing clustering and model selection
design. The experiments are conducted in dynamic TPC-C and JOB with the same setting as in Section 7.1.1. Figure 14 shows the improvement in cumulative performance against the DBA default and safety statistic of those baselines. In the following evaluation, instead of cumulative performance, we use cumulative improvement that shows the benefit of online tuning more clearly. For the read-only workload JOB where no data changes occur, OnlineTune-w/o-data exceeds OnlineTune slightly because the dimension of context feature decreases with no information loss, which means that modeling over contexts becomes easier. However, when data change occurs as in TPC-C workload, OnlineTune outperforms the other baselines since its context feature gives a comprehensive abstraction of the dynamic environment. Since the clustering and model selection strategy can prevent "negative transfer", OnlineTune outperforms OnlineTune-w/o-clustering in all the cases.

7.3.2 Ablation Study on Safe Exploration. To safely optimize the online database, OnlineTune utilizes contextual GP (black-box knowledge) and domain knowledge (white-box knowledge) and gradually expands the optimization subspace. To analyze their functionality, we compare: (1) OnlineTune, (2) OnlineTune-w/o-white, removing white-box safety assessment (3) OnlineTune-w/o-black, removing black-box safety assessment, (4) OnlineTune-w/o-subspace, optimizing in the whole configuration space, (5) OnlineTune-w/o-safe, removing all the safety strategy, i.e., vanilla contextual BO. As shown in Figure 15, OnlineTune beats other baselines in both performance and safety degree. We make the following observations: (1) The black-box knowledge largely reduces the unsafe recommendations. The limited domain rules only cover a small subset of unsafe cases and fail to capture the complex and high-dimensional relationship between configuration, environment, and database performance. Therefore, OnlineTune-w/o-black’s unsafe recommendations are much more than OnlineTune-w/o-white’s. (2) The white-box knowledge assists in filtering the unsafe configurations. We find that the unsafe recommendation in OnlineTune-w/o-white is mainly caused by the knobs without intrinsic ordering, e.g., thread_concurrency. Thread_concurrency defines the maximum number of threads permitted inside InnoDB. But a value of 0 (the default value) is interpreted as infinite concurrency (no limit). The GP model depends on the natural ordering, e.g., thread_concurrency. (3) Optimizing over promising subspace instead of the whole configuration space can enhance safety and localize good configurations. OnlineTune-w/o-subspace recommends more unsafe configurations than OnlineTune. As discussed, it is hard to generalize the safety of configurations in the whole space, and the prediction of GP is more trustworthy in a small trust region. Besides, the global optimization approaches (i.e., OnlineTune-w/o-subspace, OnlineTune-w/o-safe) they over-explore boundaries of configuration spaces. (4) Without any safety designs, OnlineTune-w/o-safe has the worst performance.

7.3.3 Varying Interval Sizes. We run OnlineTune under different interval sizes: 5 seconds, 1 minute, 3 minutes, 6 minutes, and 12
Table 1: Statistics on static workloads: For each workload, bold face indicates the best value. Search Step denotes the iteration needed to find a configuration within 10% of the estimated optimum, \( \backslash \) denotes such configuration is not found.

| Workload | ONLINE TUNE | BO | DDPG | ResTune | QTune | MysqlTuner |
|----------|-------------|----|------|---------|-------|------------|
|          | Max Improv. | Search Step | Max Improv. | Search Step | Max Improv. | Search Step | Max Improv. | Search Step | Max Improv. | Search Step | Max Improv. | Search Step |
| TPC-C    | 17.03%      | 176 | 19.99% | 74       | 16.66% | 76       | 12.03%      | \( \backslash \) | 12.02%      | \( \backslash \) | 13.44%      | \( \backslash \) |
| Twitter  | 48.18%      | 129 | 43.43% | 158      | 33.79% | \( \backslash \) | 46.99%      | 10         | 8.06%       | \( \backslash \) | 13.07%      | \( \backslash \) |
| JOB      | 11.67%      | 141 | 7.77%  | \( \backslash \) | 7.46%  | \( \backslash \) | 11.84%      | 155        | 11.24%      | 158         | 7.23%       | \( \backslash \) |

Figure 18: Iterative performance on static workloads: minimal throughput (0) and maximal latency (200) denote system failure.

As shown in Figure 17, the MySQL default’s performance is worse than DDPG, QTune in all the cases. The existing approaches work well to search for optimal configurations suitable for dynamic workloads. Second, ONLINE TUNE could collect more observations in a given time, leading to better tuning policies. However, the interval size also determines the time for evaluating the database performance. It cannot be too small to avoid the instability caused by performance fluctuation [7]. As shown, tuning with a 5-second interval performs worse than the 1-minute one with more unsafe recommendations. And we have observed significant performance variance for 5-second intervals on a fixed configuration.

7.3.4 Varying Initial Safety Set and Safety Threshold. In the above evaluation, we use the DBA default as the initial safety set and its performance as a safety threshold. However, there exists a question: Can ONLINE TUNE recommend suitable configurations with an inferior starting point? Therefore, we use the MySQL default configuration as the initial safety set and its performance as the safety threshold. As shown in Figure 17, the MySQL default’s performance is worse than the DBA default’s. One main difference is that the MySQL default sets the buffer pool size 128 MB while the DBA default sets the buffer pool size 13 GB. When starting from the MySQL default, ONLINE TUNE applies safe configurations better than the MySQL default. And it achieves a comparable performance to tuning with the DBA default as the starting point after about 150 iterations.

7.4 Evaluation on Static Workload

The existing approaches work well to search for optimal configurations on static workloads. This evaluation aims to assess the search efficiency of ONLINE TUNE with the safety constraints. Figure 18 presents the performance with statistics shown in Table 1.

ONLINE TUNE reduces the unsafe recommendations significantly with search efficiency comparable to the state-of-the-art offline tuning methods. Offline methods are designed to search for the optimum without safety considerations. In terms of efficiency, ONLINE TUNE is comparable to BO, ResTune, and better than DDPG, QTune in all the cases. ONLINE TUNE is very unlikely to apply unsafe configurations, while the offline methods violate safety constraints considerably. Although ONLINE TUNE’s safety consideration may make the exploration slower, adaptively restricting the search space could localize good solutions, which improves the convergence, especially in JOB’s case.

8 CONCLUSION

We introduce ONLINE TUNE, an online tuning system that is aware of the dynamic environments and optimizes the database safely. ONLINE TUNE features the dynamic environmental factors as context feature and leverages Contextual Bayesian Optimization to optimize the context-configuration joint space. We propose a safe exploration strategy, greatly enhancing the safety of online tuning. As future extensions, we plan to investigate combining ONLINE TUNE with offline tuning. The offline process could explore more configurations on replicas of the target DBMS, replaying the historical workloads to collect observations for online tuning. Therefore, ONLINE TUNE could exploit the promising configuration space in the online phase, thus responding to the dynamic environment more quickly. In addition, ONLINE TUNE could pause online configuring after applying suitable configurations. This could be achieved by a stopping and triggering mechanism. We could keep ONLINE TUNE’s workflow at each iteration (including context featureization and acquisition value calculation for candidate points) but not change the database configurations until more promising candidates appear. For example, we can measure whether more promising candidates exist by calculating the Expected Improvement (EI) value against the applied configuration. The configuring is triggered when candidates with EI values larger than a threshold exist, indicating that the context changes lead to the need for re-configuring.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC)(No. 61832001), Alibaba Group through Alibaba Innovative Research Program and National Key Research, and the Beijing Academy of Artificial Intelligence. Bin Cui is the corresponding author.
