DBAugur: An Adversarial-based Trend Forecasting System for Diversified Workloads

Yuanning Gao[†], Xiuqi Huang[†], Xuanhe Zhou[‡], Xiaofeng Gao[*†], Guoliang Li[‡], and Guihai Chen[†]
[†]MoE Key Lab of Artificial Intelligence, Department of Computer Science and Engineering
[†] Shanghai Jiao Tong University, Shanghai, China
[‡]Department of Computer Science, TsingHua University, Beijing, China
Email: {gyuanning, huangxiuqi, gaoxiaofeng, chen-gh}@sjtu.edu.cn, {zhouxuan19@mails., liguoliang@}tsinghua.edu.cn

***Abstract*—Trend forecasting is vital to optimize the workload performance. It becomes even more urgent with an increasing number of applications and database configurations. However, DBAs mainly target at historical workloads and may give sub-optimal configuration advice when the workload trends have changed. Although there are some studies on trend forecasting, they have several limitations. First, they mainly predict the changes of query numbers, which do not combine other critical factors (*e.g.,* disk utilization) and cannot fully reflect the future workload trends. Besides, there are numerous queries in the workloads and exact clustering algorithms like K-means cannot effectively merge similar queries which contain noises like time shifts. Second, basic machine learning models like RNN may have relatively low prediction accuracy on complex workloads (*e.g.,* no cycles but random bursts). Third, real-world workloads may have diverse patterns, while previous models cannot efficiently and reliably predict for all the different workload patterns.**

To address these challenges, we propose a trend forecasting system (`DBAugur`) that utilizes adversarial neural networks to predict the trends of different workloads. First, `DBAugur` collects the important features (*e.g.,* queries, resource metrics) to characterize workloads, and reduces the number of involved queries by separately merging similar queries based on the SQL semantics and trend patterns. Second, `DBAugur` utilizes Generative Adversarial Networks (GANs) to capture the latent patterns, correlations between different metrics, and occasional bursts within the complicated and time-varying workloads. Moreover, we further propose a time-sensitive ensemble algorithm that takes advantage of various machine learning models (*e.g.,* generative models, convolutional models, feed-forward models) to accommodate the various workload patterns. The experimental results show that `DBAugur` outperformed state-of-the-art methods on various real-world workloads.

***Index Terms*—Workload Forecasting, GAN, DBMS**

I. Introduction

With the increase in data scales, applications, as well as various system configurations, the database management system (DBMS) has become more sophisticated and demanding to manage. Many database administrators (DBAs) devote roughly 25% of their time tuning the database system to improve performance [1], which is time-consuming and expensive. To relief the human burden and enhance tuning efficiency, various database optimization methods are proposed to automatically optimize the system configurations in different aspects (*e.g.,* knobs [2]–[4], indexes [5], [6], materialized views [7], [8]), which have become a hotspot of researches.

[*]Xiaofeng Gao is the corresponding author.

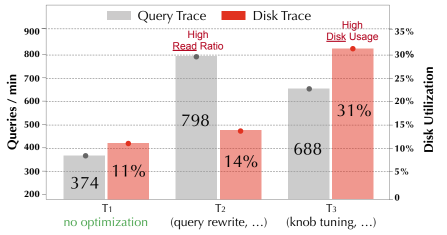


Fig. 1: Trend Forecasting Example. We trace the trends of two workload metrics, *i.e.,* query number and disk utilization, which can benefit database optimization.

The first and most essential step towards optimizing the performance is to accurately anticipate workload trends, *i.e.,* the characteristics (*e.g.,* queries, resource consumption) of future workloads [1]. Traditional methods typically rely on DBAs to collect historical workloads and optimize database configurations by the empirical experience [9]. This hand-crafted solution works effectively when the workload is stable and does not fluctuate too much. However, in real-world scenarios, workloads are diversified and tend to dynamically change over time. For example, in Figure 1, the workloads at T_2 and T_3 require different optimization solutions (*e.g.,* the disk utilization ratio is high at T_3 and we require to increase the threshold of concurrent disk I/O operations) and the configuration for workload at T_1 is no longer optimal. In this case, the manual method may take great time to analyze the new workload patterns or pick suboptimal configurations based on out-of-date historical workloads.

Limitations of Existing Methods. There are some machine learning based (ML) methods that aim to predict the workload changes and proactively adjust the configurations [1], [10]. They predict the occurrence frequency (i.e., query arrival rates at each timestamp) of templated queries with classic machine learning models like RNN. However, there are several limitations. First, the query occurrence frequency cannot represent the workload trends well. Other important factors like *changes in resource usages* are also vital to predict, which cannot be directly inferred by the query frequency features. In a distributed DBMS, for instance, if some SQL queries involving lots of disk-I/Os (e.g., range scans) hit the same server, that server will become a hotspot. In this case, we require both query frequency features and resource utilization ratios to perform load balancing, because such queries may have low arrival rates. Besides, they cluster queries (by the patterns of occurrence frequencies) with exact algorithms (like K-means) to reduce the forecasting overhead. However, even similar query traces may contain noises in real scenarios, which should not direct them into different clusters. For example, in planetarium, users always look up the number of left tickets and the ticket prices together, whose queries should be clustered together even if they have slight time difference. Third, basic ML models mainly capture short-term relations within workload traces. Whereas some workloads are very complex and require to forecast based on a long-term historical workload traces, for which basic ML models may have low forecasting accuracy. Fourth, they utilize single ML model like RNN to forecast for various workloads, which are of different complexity and the RNN-based forecasting model may perform bad on some workloads. Thus, we require multiple models for comprehensive decision-making.

Challenges. To efficiently and precisely forecast for real-world workloads. There are three main challenges. ❶ *How to combine similar workloads so as to reduce the forecasting overhead* (**C1**). There can be numerous workload traces and forecasting for each trace may cause great resource waste. Meanwhile, the noises and occurring time difference make it more challenging to identify similar workloads. ❷ *How to learn from complex workload traces and forecast future traces* (**C2**). It is usually impossible to construct a "one-size-fits-all" forecasting model because each model has its advantages. On one hand, traditional statistic methods (*e.g.,* Linear Regression (LR) and ARIMA [11]) are explicit and simple to implement. However, they are insufficient to learn complex workload patterns. On the other, machine learning (ML) approaches (*e.g.,* Multilayer Perceptron (MLP) [12], Long Short Term Memory (LSTM) [13], Kernel Regression (KR) [14] and Temporal Convolutional Networks (TCN) [15]) are good at learning workload distribution across time series. However, current ML-based algorithms suffer from the overfitting problem and exist high-variance in prediction. Furthermore, when dealing with more complex workload forecast scenarios, they are also insufficient because of the lack of fitting ability. ❸ *How to generalize the forecast model to diversified workloads* (**C3**). Recap that there can be various workloads in real scenarios. Existing methods only rely on single ML models to forecast for each workload, which may cause bad prediction on some workloads (*e.g.,* sudden bursts and long-term patterns). Since there are numerous available ML models, it is challenging to choose a suite of effective models for trend forecasting (*e.g.,* learning the complex query/resource patterns, learning the long-term pattern changes) and ensemble their learned knowledge to give the final decisions. As a result, we expect to update (train) the forecasting model in a dynamic manner.

Our Proposed Methods. To address the challenges, we propose a robust learning-based workload forecasting system (`DBAugur`) using generative adversarial networks. ❶ We first collect the workload queries from the system logs, obtain the runtime statistics (*e.g.,* resource usages), and use both the queries and resource usage to represent the workloads. Here we adopt two strategies to reduce the forecasting overhead: (1) We map the queries into query templates based on the semantic features; (2) We cluster the workloads with similar trace patterns together, where we propose dynamic time warping to identify the similar patterns between any two traces (for **C1**). ❷ With the featurized workloads, we develop `WFGAN`, which exploits the strong generative potential of Generative Adversarial Networks (GAN) [16] to capture complicated and time-varying workload patterns. With adversarial learning, we can learn a tailored loss function, which helps learn the latent workload distribution and capture the correlation and burstiness across time. Specifically, we leverage multi-task learning [17] to jointly train WFGAN on query and resource utilization workloads so that the knowledge learned in one forecasting task can benefit the other forecasting task, improving the learning efficiency and prediction accuracy (for **C2**). ❸ Moreover, we further propose a time-sensitive ensemble algorithm that takes advantage of both GAN (the generative capability), TCN (the global view for complex workloads), and MLP (the local view for simple workloads) to accommodate the diversified workload patterns (for **C3**).

Contributions. We make the following contributions:

(1) We develop a novel learning-based workload forecasting system `DBAugur`, which can provide accurate and efficient workload prediction for databases. To the best of our knowledge, this is the first adversarial based database workload forecasting system.

(2) To deal with the growing volume of workload data and boost training efficiency, we propose an online workload clustering method, where *Dynamic Time Warping* (DTW) [18] is introduced to replace the traditional distance computation method. It can handle distorted workload patterns and detect underlying patterns with varying lengths and speeds. Besides, Ball-Tree [19] is integrated in this clustering method to accelerate the nearest neighbor search.

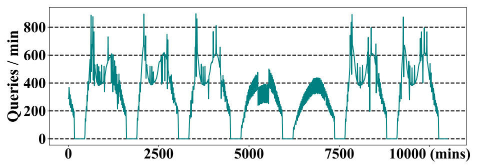
(3) To accurately forecast complex workload, we design a forecasting model that utilizes adversarial neural networks to learn the internal distribution of these workload patterns.

(4) To adapt to diversified workloads, we present a dynamic ensemble algorithm combining the advantages of `WFGAN`, `TCN`, and `MLP` to improve the accuracy of forecasting various workload traces. Specifically, we design a time-sensitive weight calculation method to integrate the above three models and result in improved forecasting performance.

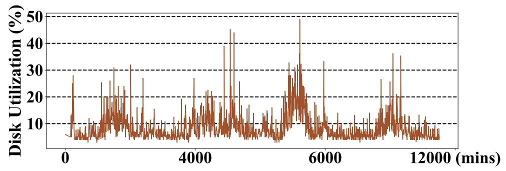
(5) We compare `DBAugur` with other state-of-the-art forecasting models on real-world workloads, which demonstrates the superiority of `DBAugur` in capturing diversified workload patterns. Besides, we implemented `DBAugur` on PostgreSQL to test its ability to optimize real-world database applications.

II. Preliminaries

We first introduce database workloads and showcase typical workload patterns (Section II-A), and then we formalize the problem of *workload forecasting* (Section II-B).



(a) BusTracker Workload Trace



(b) Alibaba Cluster Workload Trace

Fig. 2: Workload Patterns – Examples of two real-world workload patterns.

A. Database Workloads

Database workload traces are a collection of queries that access the same database instance during a certain time period (*e.g.,* queries submitted in the past hour). From the view of system burdens, we characterize database workloads in two aspects, *i.e.,* query traces (*e.g.,* the numbers of submitted queries) and resource utilization traces (*e.g.,* memory utilization ratios).

Query Trace. Query trace $W(Q)$ refers to the changes of query statements issued by the user's applications over time, which is typically measured by the occurrence frequencies.

Example 1: With the guidance of predicted query traces, we can benefit downstream optimization tasks. For example, for index management, we can create the corresponding index on the predicted "hot" columns to accelerate the query execution. Furthermore, the system configurations, such as buffer size, number of thread connections, number of locks, and so on, can also be optimized based on the forecasting knowledge.

Resource Utilization Trace. Resource trace $W(R)$ refers to the resource utilization, *e.g.,* consumed memory (*e.g.,* the memory use of the database over a given period of time), CPU (*e.g.,* the number of instructions being executed by the processor during the period), and storage space (*e.g.,* a statistic on the number of inputs and outputs gathered by the database).

Example 2: By forecasting the changes in required system resources, we can judiciously release or allocate system resources so as to meet the performance requirements and improve the resource utilization for the incoming workloads.

Database Workload. Based on above observations, we need both query and resource utilization traces to reflect actual workload characters. Otherwise, it may lead to inaccurate results by only forecasting either queries or resource usage.

Definition 1 (Database Workload): A database workload is denoted as $W = (Q, R)$, where Q and R separately denote the query and resource utilization traces. Given a set of queries Q in the workload, the queries are concurrently executed, which consume some system resources R to obtain the desired data.

Example 3: As the demand for large distributed database systems increases, different types of workloads need to be considered coordinately. OLTP has concurrent queries and is io-intensive; while OLAP has complex queries and is memory-intensive. Therefore, it is necessary and critical to consider both queries and resources to fully reflect future workloads.

With the historical workloads, it is still challenging to accurately forecast the incoming workload traces since workload patterns present on a database instance can be highly dynamic and time-varying [20]. First, the workload is non-static. From the global view, the workload exhibits periodic patterns. However, the period of these patterns can be very long and less obvious, making it difficult to capture them. Second, there are sudden bursts in workload, which require a robust ability to detect bursts ahead of time and adapt rapidly to these dynamic scenarios. Furthermore, because workload patterns tend to vary over time, the forecasting model should be able to adapt to the new patterns.

B. Workload Forecasting

Forecasting Horizon. To decide the time distance of future workloads from current workloads, we define the forecasting horizon.

Definition 2 (Forecasting Horizon): Forecasting Horizon refers to how far the model predicts the future. Given current workloads $(x_1, x_2, \ldots, x_T) \in \boldsymbol{R}^T$, of which x_i, $1 \leq i \leq T$, means the workload at time i, the forecasting horizon H means that the model can predict the future workload x_{T+H} at time $T + H$.

Example 4: Given a database workload trace, the forecasting horizon equals 1 hour means that we want to predict what the workload looks like after 1 hour later. Increasing the forecasting horizon will decrease the forecasting accuracy.

Forecasting Interval. Next to decide the granularity of workload forecasting, we define the forecasting interval.

Definition 3 (Forecasting Interval): Forecasting Interval represents the time interval between two adjacent workload values x_{i-1} and x_i. The forecasting interval I equals to the difference between time $i - 1$ and time i.

Example 5: When pre-processing the workload trace, if the forecasting interval is set to 10 minutes, we will aggregate the workloads by 10 minutes. If the forecasting interval is small, the workload trace can contain more detailed information, improving the prediction accuracy. The small interval, on the other hand, will increase the volume of workload, making the training process costly.

Workload Forecasting. Based on the forecasting horizon and interval, database workload forecasting aims to predict the workload trend changes (*e.g.,* query and resource utilization traces) during a certain period of future time.

Definition 4 (Single-Trace Forecasting): For any workload trace $\boldsymbol{X} = (x_1, x_2, \ldots, x_T)$ with the forecasting horizon H and forecasting interval I, where x denotes one type of workload metrics (*e.g.,* query arrival rate, resource usage), workload forecasting is to predict the value of workload metric x at time $T+H$, *i.e.,* $\hat{x}_{T+H} = F(x_1, x_2, \ldots, x_T)$, where $F(\cdot)$ denotes any forecasting model.

Definition 5 (Multi-Trace Forecasting): Given a set of workload traces $\{\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \cdots, \boldsymbol{X}^{(n)}\}$, where $\boldsymbol{X}^{(j)}$ denotes one

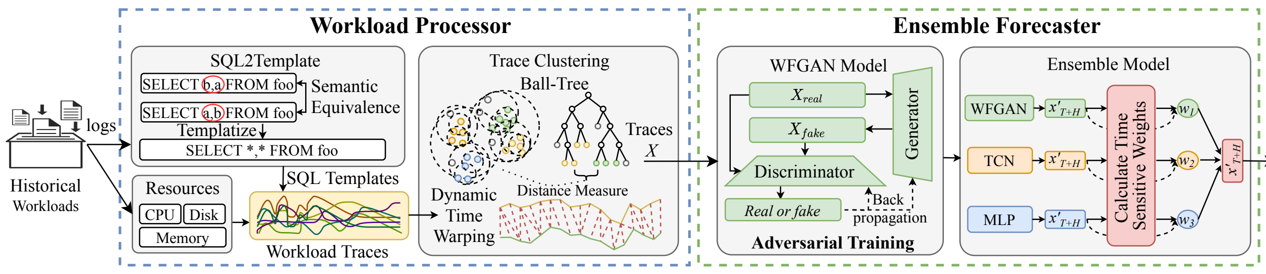


Fig. 3: System Framework.

workload metric trace and n denotes the number of workload metrics, for any future time T, workload forecasting is to infer the workload trend at time $T+H$ by predicting the metric values at time $T+H$, *i.e.,* $\hat{W}_{T+H} = \{\hat{x}^{(1)}_{T+H}, \hat{x}^{(2)}_{T+H}, \cdots, \hat{x}^{(n)}_{T+H}\}$.

Example 6: As shown in Figure 2, real-world workloads have various access patterns. *BusTracker* workload[1] tracks the public transit bus system in real time and records the number of SQL queries executed every minute by the BusTracker application. As shown in Figure 2(a), although this workload roughly follows a one-day cyclic pattern, there are various sudden crests and troughs as time going on. Figure 2(b) presents *Alibaba Cluster Workload Trace*[2], a collection of resource utilization traces that are gathered from real world production. As can be seen, the periodic pattern in this dataset a is longer and less obvious. Moreover, there are many bursts caused by complex queries. In these scenarios, both current ML-based algorithms and QB5000 [1] are insufficient to capture long-term patterns and sudden bursts.

III. System Overview

DBAugur is an end-to-end workload forecasting framework that can be utilized as an external auxiliary tool for DBAs or as an in-kernel DBMS module for database optimization. In this section, we present the system overview. As shown in Figure 3, DBAugur consists of three main parts, including *Workload Processor*, *Ensemble Forecaster* and *Adversarial Training*.

Workflow. Given a new workload being executed in the database, after a period of time, the *Workload Processor* first extracts the historical workload traces from database logs, where queries with similar syntax formats are merged into a small number of query templates, clusters similar traces with dynamic time warping, and utilizes the processed traces to represent historical workloads. With those workload traces, next the *Ensemble Forecaster* trains a learned ensemble forecasting model to predict the future trend of the workload. The ensemble model is composed of three parts, where *MLP* is a full-connected neural network that captures the short-term trace changes, *TCN* is a convolutional neural network that captures the long-term trace patterns, and *WFGAN* is a generative adversarial network that adopts the adversarial training mechanism and can capture the bursts in workload traces. The ensemble model predicts the workload trend at any future time based on the forecasting results of the three inference models.

[1]http://www.cs.cmu.edu/ malin199/data/tiramisu-sample
[2]https://github.com/alibaba/clusterdata

Workload Processor. To begin with, since there are numerous database logs containing vital workload traces (*e.g.,* queries with timestamps) and the textual data in the logs cannot be directly fed into the forecasting model, *SQL2Template* introduces the template-based equivalence checking technique, which converts raw query logs into various SQL templates. Next, with the SQL templates and other resource workload traces, *Trace Clustering* aims to efficiently group similar workload traces into clusters to further reduce the forecasting overhead and handle the continuously new query data. To achieve that, we propose an online workload clustering algorithm, which takes workload traces as input and output clustered workloads. To efficiently cluster the data, we propose a dynamic time warping (DTW) approach, which allows comparisons of two distorted workload patterns. Ball-Tree is also built in *Trace Clustering* as a storage structure for efficient nearest neighbor search.

Ensemble Forecaster. Forecasting model is the core component of DBAugur. It uses clustered workloads as training data and provides predictions for workload characters at any future time (*i.e.,* different workload horizons and intervals). To achieve that, we first propose a adversarial-based model called WFGAN. In WFGAN, the generator learns the latent distribution of workload patterns, while the discriminator helps to capture the correlation and burstiness in time-varying forecasting. We then put forward a time-sensitive ensemble algorithm that fuses multiple forecasting models (*i.e.,* WFGAN, TCN and MLP) for more accurate prediction by dynamically assigning different weighting factors to the models. Note that the three models can be trained in parallel.

Adversarial Training. To well train the forecasting model on complex workload traces (*e.g.,* random bursts), we train the generative forecasting model (WFGAN) with adversarial training. *Adversarial training* aims to ensure that the forecasting model simulates all the important characters of the workload traces so as to "cheat" the discriminator and gain relatively high accuracy. Each time we sample a batch of workload traces and rely on the generator-discriminator mechanism to learn the internal distribution of those workload traces (*e.g.,* typical patterns like cycles and bursts).

IV. DTW-based Workload Processor

Due to the large number of workload traces, building forecasting models for each of these traces are impractical. Intuitively, we can cluster similar queries to reduce the forecasting overhead. However, existing methods mainly rely on exact distance computation strategy (*e.g.,* cosine distance), which cannot effectively merge similar queries with noises like time shifts [1], [10]. Hence, in this section, we first extract the workload information from database logs (Section IV-A); and then we propose a robust distance computation method to measure the similarity between any two workload traces (Section IV-B), and cluster similar workload traces based on the computed distances (Section IV-C).

A. Workload Trace Extraction

The gathered database logs are usually of the string type and have messy formats and cannot be used as direct input to the learning model. As a result, we should first extract the numerical workload characteristics from the original logs, including the query traces and resource utilization traces. For query traces, they are extracted from the query logs, and there are two observations.

(1) The SQL statements are ordered by execution timestamps, based on which we can generate the workload traces for all the query statements, *i.e.,* $(x_1, x_2, \cdots, x_T)$, where x_i represents the workload value at time i (i.e., the arrival rate of corresponding query), and T denotes the entire length.

(2) Many SQL statements have similar syntax format and we can combine these statements as single query templates so as to reduce forecasting overhead. To accomplish this, we first format the statements by normalizing the statement format (*e.g.,* the same usage of spacing, case, bracket placement). Next, we replace the variables in the SQL queries with placeholders. For instance, the SQL statement "`SELECT * FROM Stu WHERE id=`**`5`**` and age>`**`21`**` and height<`**`180`**" $\rightarrow$ query template "`SELECT * FROM Stu WHERE id=`**`$`**` and age>`**`&`**` and height<`**`#`**", whose trace equals to the sum of all the combined query traces. Moreover, to further decrease the number of query templates, *semantic equivalence checking* is also performed to assort similar SQL templates into the same type. For example, the statement "`SELECT a,b FROM foo`" is equivalent to "`SELECT b,a FROM foo`", and "`SELECT * FROM A JOIN B on A.id=B.id`" is equivalent to "`SELECT * FROM B JOIN A on B.id=A.id`".

For resource utilization traces, they are extracted from runtime statistics. For example, there are usually thousands of virtual machines and containers running the database service in a cloud environment. For each database instance, we can obtain the corresponding workload series $(x_1, x_2, \cdots, x_T)$ for a specific resource (*e.g.,* CPU, I/O, memory), where x_i represents the resource utilization ratio at time i.

After the above processing, the origin query and resource utilization information are finally converted into much fewer workload traces ordered by the timestamp attribute.

B. Similarity Measure of Workload Traces

To forecast the trend of database workloads (*e.g.,* query workloads or resource workloads), we should train a corresponding forecasting model for each workload trace. For queries, there are large number of SQL templates within an application and each SQL template corresponds to a workload trace. For utilized resources, there are numerous servers running the database service on the cloud and each server also records corresponding resource traces (*e.g.,* I/O or CPU utilization ratios on each server). As a result, the number of traces is still large, training a model for each workload trace is unfeasible. Before building forecasting models, the number of workload traces should be further reduced. To this end, we can group similar workload traces into clusters and train the forecasting model for representative clustered workloads. This strategy can be used without compromising the forecasting accuracy. First, a small number of clusters can cover most database workloads. Thus, building forecasting models on the top-K clusters is fully capable of capturing the common workload patterns. Second, many workload traces have similar workload patterns and can be grouped together. For example, a set of statements in a distributed transaction will generate similar query and resource traces.

When clustering the workloads, a key problem is how to efficiently measure the similarity of any two workload traces. Traditional methods usually adopt Euclidean distance or Cosine distance [1] to compute the distance between two sequences. However, due to the possibility of temporal drift, the above two methods are unable to precisely match two warped workload traces, causing poor clustering results. In this paper, we solve this problem by introducing a more powerful distance computation technique called Dynamic Time Warping (DTW) [18], which is widely used in the field of pattern recognition, as illustrated in the second part of Figure 3. DTW is a robust similarity measurement that allows comparisons of two traces with varied lengths and speeds, which is superior to the lock-step measures such as Euclidean distance [21]. Therefore, it distinguishes underlying patterns more efficiently than searching for perfect matches in the raw traces, naturally increasing the resistance of `DBAugur` to data drifting such as amplitude shifting/scaling, linear increase/decrease, etc. The DTW algorithm is shown in Algorithm 1. The primary idea is to utilize dynamic programming to discover the optimum alignment between two traces by warping the time dimension with certain constraints, as shown in lines $6-9$ in Algorithm 1. The output in line 10 is the distance between $\boldsymbol{X}^{(1)}$ and $\boldsymbol{X}^{(2)}$ with the best alignment. To improve the matching efficiency, we add a window w in DTW to limit its search space. Moreover, we adopt the LB_keogh algorithm [22] to further decrease the time complexity of DTW to linear time $O(T)$.

C. Workload Trace Clustering

To efficiently cluster workload traces, based on DBSCAN [23], we propose an online workload clustering algorithm called Descender (**D**ensity bas**E**d **S**patial **C**lust**E**ri**N**g with **D**ynamic tim**E** wa**R**ping), which is a density based method and is roust to arbitrary shaped clusters and outliers.

The Descender algorithm is described as follows. Given a set of workload traces W, a similarity threshold ρ and a similarity measurement function, we intend to group the traces with similar patterns together. The main idea is that a trace

Algorithm 1: DTW Based Distance function

```
Input: Workload traces $\boldsymbol{X}^{(1)}$ and $\boldsymbol{X}^{(2)}$
Input: Historical length $T$ and Window size $w$
Output: Distance between two workload traces
1  Initial $DTW$ as a $T \times T$ matrix;
2  foreach $i := -1$ to $T$ do
3      foreach $j := -1$ to $T$ do
4          $DTW[i,j] = infinity$;
5  $DTW[-1,-1] := 0$;
6  foreach $i := 0$ to $T$ do
7      foreach $j := \max(0, i-w)$ to $\min(T, i+w)$ do
8          $dist = (\boldsymbol{X}^{(1)}[i] - \boldsymbol{X}^{(2)}[j])^2$;
9          $DTW[i,j] = dist + \min(DTW[i-1,j], DTW[i,j-1], DTW[i-1,j-1])$;
10 Out: $Sqrt(DTW[T-1,T-1])$;
```

belongs to a cluster if it is close to many other traces from that cluster. Descender first builds a Ball-Tree [19] on the current workload traces, which partitions traces into a nested set of hyperspheres known as "balls" to speed up discovery of neighborhood workload traces. Compared with KD-Tree, Ball-Tree is more efficient especially in situations when the number of dimensions is very large.

Workflow. When the clustering process begins, Descender will iterate over each trace $\boldsymbol{X}$ and uses Ball-Tree to quickly find its neighbors under radius ρ. If there are at least $MinSize$ traces in its neighbors (i.e., this trace and its neighbors may form a cluster), the trace $\boldsymbol{X}$ will be identified as a *core point*. Otherwise, it is marked as an outlier. Descender then generates a new cluster $C_{\boldsymbol{X}}$ based on the new trace $\boldsymbol{X}$. In this case, all the traces in the neighbors $N_{\boldsymbol{X}}$ also become a part of that cluster. In the same way, if these neighbors are also *core points*, the traces in their immediate vicinity are also added to the same cluster $C_{\boldsymbol{X}}$. In this way, the Descender algorithm is capable of grouping the workload traces into different clusters.

Online Clustering. Descender also supports online clustering. For a new trace, Descender will update the environment, merge or split the clusters based on the current clustering density. If the new trace fails to become a *core point*, we will create a new cluster with that trace as its sole member.

After the clustering processing, all the workload traces have been grouped into separate clusters. The majority of the database workload can be characterized by a a small number of clusters, leaving the other clusters to have little impact on the forecasting task. Therefore, we only select the top-K representative clusters (i.e., clusters with the largest workload volumes) and build a forecasting model for each cluster, for which we use average workload of traces within each cluster as the training data. During the clustering, we also track each trace and its proportion in the corresponding cluster to which it belongs. Accordingly, we can infer the trend for each trace once we forecast the cluster's future workload tendency.

V. Forecaster

In order to improve forecasting performance, we first propose **W**orkload **F**orecasting **GAN** (WFGAN), the first adversarial based model that exploits the ability of GAN to generate real workload values from latent space (Section V-A, V-B). The reasons are as follows. ❶ Unlike the previous models, which only optimize one single objective [24] such as the likelihood loss, the hinge loss or other losses, WFGAN uses an adversarial training procedure to directly model the latent distribution of the workload traces, which can detect the underlying patterns of random workload traces and eliminate the error accumulation problem. ❷ Under the supervision and criticism of the discriminator, WFGAN is capable of building a more sufficient non-linear mapping through neural networks, which can capture the correlation and burstiness across long-term workload patterns. ❸ During training and inference phases, the discriminator performs like a tailored loss function, which sufficiently exploits the workload information to regularize and guide the generator for adaptive and precise forecasting. Next, we further present a time-sensitive ensemble algorithm to generalize the forecasting model to diversified workloads. The ensemble algorithm employs multiple models to ensure that forecasting results are accurate and robust in a variety of scenarios (Section V-C).

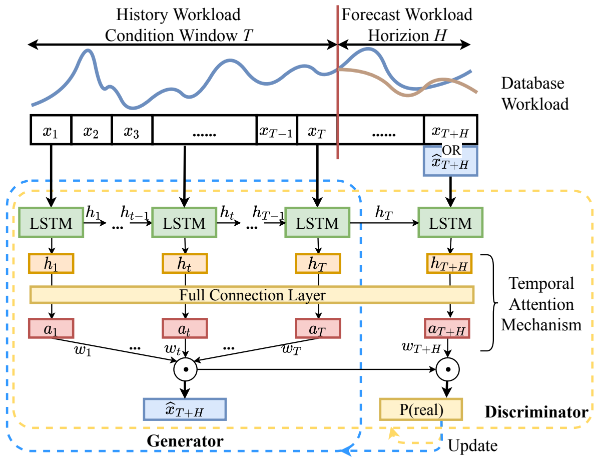


Fig. 4: Model Architecture (WFGAN).

A. *WFGAN for Workload Forecasting*

A generative adversarial network (GAN) [16] is a type of machine learning framework that is designed for modeling the latent data distribution given a set of samples from the true data distribution. GAN consists of two neural network components: a **Generator** G and a **Discriminator** D. To be specific, the generator G generates fake data from the noise space z and passes it to the discriminator. The purpose of the generator is to deceive the discriminator, even though they are fake. On the other hand, D has been trained to determine whether or not a given data sample is authentic. Finally, as denoted in Eqn. (1), GAN engages a min-max game between the generator G and the discriminator D based on a cost function. When using GAN to solve the workload forecasting problem, G is in charge of generating future workloads and trying to "cheat" D, while D maximizes the ability to distinguish between real and forecasted workloads. Based on the basic framework of GAN, we propose WFGAN. The framework of WFGAN is shown in Figure 4, and we will introduce these components in the follows.

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})]+\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))] \tag{1}$$

As discussed before, to accurately characterize the database workloads, we should take advantage of both query and resource utilization traces. To this end, we first merge query traces $W(Q)$ and resource traces $W(R)$ into a single workload collection W, where W contains a set of workload traces $\{\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \cdots, \boldsymbol{X}^{(n)}\}$. We then propose to employ multi-task learning (MTL) [17] to jointly learn two tasks. In MTL, multiple learning tasks are solved at the same time by sharing some network layers, which exploits commonalities and differences across tasks and thus improve learning efficiency and prediction accuracy. Specifically, the input layer receives training samples from query and resource workloads and the output layer has two output units with one for each task. The shallow network parameters in the hidden layer will be shared by both forecasting models, while their deep network parameters will be optimized separately. By this way, we can learn more robust and universal representations for two workload forecasting tasks, which can leverage useful information contained in these tasks to help improve the generalization performance of all the tasks.

Generator. In the standard GAN, the generator samples data from the noise space z using a normal or uniform distribution, which reflects the latent features of the generated data. This procedure, however, will result in an uncontrolled output, i.e., the generator will not be able to output the target workload that we require. To address this issue, we leverage a modified GAN called **C**onditional GAN (CGAN) [25], which uses external data as the condition window to improve the model's performance. By this mechanism, instead of sampling data from the noise space, we directly feed a workload trace $\boldsymbol{X} = (x_1, x_2, \cdots, x_T)$ as a condition window into the generator. In this case, the output $\hat{x}_{T+H}$ of the generator represents the predicted workload at time $T + H$. Figure 4 depicts the WFGAN's structure.

Initially, the generator will generate random data in early stages. However, during the continuous adversarial training of G and D, D will guide G to generate a more accurate workload $\hat{x}_{T+H}$. For example, if G makes inaccurate predictions for future workload trends, D will catch this error and return a large loss value to further optimize the network parameters of G. In this situation, the condition window (i.e., the historical workload trace X) offers WFGAN an explicit head start in terms of the learning target.

Discriminator. For the discriminator, real and forecasted workloads are respectively input into D. As shown in Figure 4, we separately concatenate the real workload x_{T+H} and the forecasted workload $\hat{x}_{T+H}$ to $\boldsymbol{X}$ to form a new trace $\boldsymbol{X}_{real} = \boldsymbol{X} \circ x_{T+H}$ and $\boldsymbol{X}_{fake} = \boldsymbol{X} \circ \hat{x}_{T+H}$ with length $T+1$, which represents the real data and the fake data respectively. The goal of the discriminator is to try its best to determine if the input workload is real or not. In this case, D will output a probability value indicating the likelihood that the data is real, i.e., $P(real)$. Then we can train the discriminator to be like a two-class classifier. If the input is a real workload trace, the $D(\boldsymbol{X}_{real})$ should equal 1. Otherwise, it should be 0.

As discussed in the above section, the generator tends to produce fake data in order to trick the discriminator as much as possible, i.e., $D(G(\boldsymbol{X})) = 1$. As a result, we can use the probability output by D to train the generator through the back-propagation algorithm. Finally, the discriminator D can only identify the tiny difference between the real and the generated workloads, i.e., the generator G predicts the future workload that the discriminator cannot tell the difference. The WFGAN model eventually converges and can accurately forecast the workload x_{T+H}.

Internal Structure. The internal structure of G and D adopts a similar RNN-based neural network, as illustrated in Figure 4. Given a workload trace $\boldsymbol{X} = (x_1, x_2, \cdots, x_T)$, the RNN network can learn a non-linear mapping from x_t to h_t, *i.e.,* $h_t = f(h_{t-1}, x_t)$, where h_t represents the hidden state of x_t (i.e., the output of RNN at time t). To better capture workload patterns and improve forecasting efficiency, WFGAN adopts a modified RNN called LSTM (Long Short Term Memory) [13]. LSTM is made up of a number of memory units that can selectively cache the historical information for current prediction.

We can regard the last hidden state h_T of LSTM as the predicted workload. However, relying only on the last output may lose information. To fully exploit the historical knowledge, we introduce the temporal attention mechanism [26], which can pick elementary stimulus features in the early stages of workload trends. For the final output, we add an attention-layer as shown in Figure 4. For the generator, we summarize all the hidden states from h_1 to h_T to produce the predicted workload $\hat{x}_{T+H}$, as denoted in Equation (2), where $f_{attention}$ is a weighting function to integrate the hidden states:

$$\hat{x}_{T+H} = f^{G}_{attention}(h_1, h_2, \cdots, h_T) \tag{2}$$

For the discriminator, we concatenate h_{T+H} to the first T hidden states. Then the neural network will output the probability to denote whether the input sample is real or not, as shown in Equation (3).

$$P(real) = f^{D}_{attention}(h_1, h_2, \cdots, h_T, h_{T+H}) \tag{3}$$

B. *Adversarial Training for WFGAN*

To efficiently train the model, traditional neural networks require a hand-crafted loss function. As a result, an incorrect loss function will severely damage the performance of the model. For WFGAN, we adopt adversarial training to optimize the model, in which the discriminator will perform as an tailored loss function to drive the training of the generator. The discriminator (*i.e.,* the loss function) can learn from the current workload patterns, which is robust and flexible compared to a hand-crafted loss function.

Loss Function. As previously stated, the duty of the discriminator is to distinguish between the real and predicted workloads. $\boldsymbol{X}_{fake}$ is the concatenation of the workload trace $\boldsymbol{X}$ and the predict workload $G(X)$ by the generator G. Here WFGAN measures the discriminator's loss using cross-entropy, as denoted in Equation (4), which means that D seeks to maximize the probability of distinguishing between the real and the predicted workloads:

$$\max_{D} \mathbb{E}_{\boldsymbol{X}\sim p_{\boldsymbol{X}}}[\log D(\boldsymbol{X}_{real})] + \mathbb{E}_{\boldsymbol{X}\sim p_{\boldsymbol{X}}}[\log(1 - D(\boldsymbol{X}_{fake})] \tag{4}$$

On the other hand, the generator tries to deceive the discriminator with the highest probability value, as denoted in Equation (5):

$$\min_{G} \mathbb{E}_{\boldsymbol{X}\sim p_{\boldsymbol{X}}}[\log(1 - D(\boldsymbol{X}_{fake}))] \tag{5}$$

Finally, the cost objective function of WFGAN is shown in Equation (6), where D attempts to maximize the value while G tries to minimize it:

$$\min_{G}\max_{D} \mathbb{E}_{\boldsymbol{X}\sim p_{\boldsymbol{X}}}[\log D(\boldsymbol{X}_{real})] + \mathbb{E}_{\boldsymbol{X}\sim p_{\boldsymbol{X}}}[\log(1 - D(\boldsymbol{X}_{fake}))] \tag{6}$$

Algorithm Outlines. Once the objective function is defined, the network parameters of G and D can be learned jointly by the alternating gradient descent. WFGAN first fixes the parameters of the generator and perform gradient descent by D-steps on D using the real and the predicted workloads. Then we switch sides. Fix the discriminator and train the generator by G-steps. Two neural networks are trained in alternating steps until the generator can forecast the workload accurately. The algorithm 2 shows the back-propagation algorithm used for the gradients in WFGAN training.

```
Algorithm 2: Adversarial Training for WFGAN
Input: Learning rate $\mu^D$ for $D$ and $\mu^G$ for $G$
1  Initialize gradients $\theta_D$ and $\phi_G$ ;
2  foreach number of train iterations do
3      Sample minibatch $m$ traces
       $\{\boldsymbol{X}_{real}^{(1)}, \boldsymbol{X}_{real}^{(2)}, \cdots, \boldsymbol{X}_{real}^{(m)}\}$ from training
       workloads;
4      Compute $\boldsymbol{X}_{fake}^{(i)}$ by the generator $G$:
       $\boldsymbol{X}_{fake}^{(i)} = \boldsymbol{X}_{real}^{(i)}[1:T] \circ G(\boldsymbol{X}_{real}^{(i)}[1:T])$;
5      foreach D-steps do
6          Update the discriminator by ascending its
           gradient:
7          $\theta_D = \theta_D + \mu^D \nabla_{\theta_D} \frac{1}{m}\sum_{i=1}^{m}[\log D(\boldsymbol{X}_{real}^{(i)}) +$
           $\log(1 - D(\boldsymbol{X}_{fake}^{(i)}))]$
8      foreach G-steps do
9          Update the generator by descending its
           gradient:
10         $\phi_G =$
           $\phi_G - \mu^G \nabla_{\phi_G} \frac{1}{m}\sum_{i=1}^{m}[\log(1 - D(\boldsymbol{X}_{fake}^{(i)}))]$;
```

C. Forecasting for Diversified Workloads

As described earlier, when dealing with a variety of workload scenarios, we need to build a versatile forecasting model. As discussed in [27], a model should take account of different granularities of workload data, such as short-term and long-term trend patterns, as well as dynamic and complex patterns.

- Short-term prediction requires the model to capture local and linear features. However, There may still be some nonlinear features that interfere with the prediction.
- Long-term prediction requires the model with a global view to capture long-term features in the trace. Meanwhile, we need to alleviate the problem of gradient explosion during model training.
- Complex patterns require the model to have strong nonlinear fitting capabilities to deal with unexpected situations (e.g., the bursts in the trace).

In order to meet the above requirements, we further develop an ensemble algorithm that integrates MLP, TCN, and WFGAN to form the *Forecaster* module. Next, we will elaborate on the rationality of model selection and the design of the ensemble algorithm.

TABLE I: Comparison of Forecasting Models

	Model Structure	Feature	Advantage
MLP	fully connection	local	short term
TCN	dilated convolution	global	long term
WFGAN	adversarial structure	potential	complex and burst

1) Comparison of Models: As shown in Table I, the selected models correspond to the requirements listed above.

Multilayer Perceptron (MLP) [12]. MLP is a fully connected feedforward neural network. MLP is simple and good at capturing local trend patterns, which is suitable for short-term forecasting. Compared with other linear models such as LR and ARIMA, the activation function in MLP enables it to capture the local nonlinear patterns in short-term prediction. Moreover, it can be trained quickly.

Temporal convolutional network (TCN) [15]. TCN is a variation on convolutional neural networks (CNN) for sequence modeling tasks, in which the output at time t is convolved solely with elements from time t and earlier in the previous layer. TCN employs dilated convolutions that helps cover the longer workload information from the output obtained with every convolution operation. Compared with general CNN, TCN offers a wider field of view at the same computational cost. Moreover, it can avoid the gradient explosion problem that exists in RNN. As a result, TCN has a good global view and can forecast long-term patterns effectively.

Recap WFGAN is good at modeling the bursts and learning complex workload patterns. However, it works poor in scenarios like longer-term prediction. In this case, if we make a potent brew of WFGAN, TCN, and MLP to take advantage of each model, the *Forecaster* module will be able to handle diversified workloads and gain higher forecasting ability. Note that we only consider the above three models for ensemble because they are enough to catch patterns. Adding more models to the ensemble will increase the training cost with tiny performance improvement.

2) Ensembled Workload Forecasting: In this section , we propose a time-sensitive ensemble algorithm based on fuzzy and statistics theories [28], [29] to generate the final prediction of DBAugur.

Let $\hat{x}_t(i)$ be the predicted workload value at time t, with $i = 1, 2, 3$ indicating that this forecasting value is produced using WFGAN, TCN, or MLP. Correspondingly, for the i-th forecasting model, we use $e_t(i) = (x_t - \hat{x}_t(i))^2$ to represent its prediction error at time t. A small $e_t(i)$ means that the prediction accuracy of the i-th model is high at time t. Therefore, when fusing the models at time $t + H$, we are supposed to provide a higher ensemble weight to models with a small $e_t(i)$. However, a static $e_t(i)$ is insufficient to be highly sensitive to time, because it cannot fully account for a model's future forecasting ability.

In this paper, we introduce a time-varying **forecasting distance** with an attenuation factor δ to ensure that models with strong predictive abilities have higher ensemble weights at the forecasting moment. For the i-th model, let $\mathbf{e}(\mathbf{i}) = (e_1(i), e_2(i), \cdots, e_t(i))$ denote the prediction error vector from time 1 to t, which is a dynamically changing and rising vector as time goes on. The forecasting distance is then defined as follows.

Definition 6: For the error vector $\mathbf{e}(\mathbf{i})$, its forecasting distance is

$$\Gamma(\mathbf{e}(\mathbf{i}), t) = \sum_{j=1}^{t} \delta^{t-j} e_j(i) \tag{7}$$

$\Gamma(\mathbf{e}(\mathbf{i}), t)$ uses the most recent information to measure a model's forecasting ability, which is time-sensitive and robust. Moreover, such diversified dynamic weighted function can efficiently handle incoming workload traces that have varying conceptual distributions [30]. Correspondingly, to determine the ensemble weight for each model, we normalize the forecasting distance, take the inverse, and get the time-varying weight of each model over time t.

$$w_t(i) = \frac{\sum_{j=1}^{3} \Gamma(\mathbf{e}(\mathbf{j}), t) - \Gamma(\mathbf{e}(\mathbf{i}), t)}{2 \times \sum_{j=1}^{3} \Gamma(\mathbf{e}(\mathbf{j}), t)} \tag{8}$$

At this stage, the final ensemble forecasting value at time $t + H$ can be expressed by Equation (8), where $w_t(i)$ and $\hat{x}_{t+H}(i)$ respectively represent the ensemble weight and the predicted workload value of the i-the forecasting model, *i.e.,* $\hat{x}_{t+H} = \sum_{i=1}^{3} w_t(i) \cdot \hat{x}_{t+H}(i)$.

Discussions. So far, we have covered all the components of the framework. There are primarily three points to distinguish it from traditional time series prediction. First, DBAugur inlucdes data preprocessing, trace clustering, and workload forecasting, in which the forecasting module is only one component of DBAugur. To perform database workload forecasting, we need to preprocess diversified workload traces, including extracting, cleaning, and transforming them into standard forms. Then we cluster the traces to select the top-K representative clusters so that the forecasting process can be efficiently carried out. Second, the workload patterns are implicit. For example, for an application with thousands of queries, it is tricky to capture the trace pattern of a single query. Alternatively, we characterize each group of traces by clustering the traces together and extracting timing information. Although the clustering topology evolves over time, DBAugur is able to react quickly and capture the changes of workload patterns. Lastly, the primary goal of database workload forecasting is to assist downstream modules, such as knobs tuning and index selection. In contrast to traditional time series prediction (e.g., electric demand forecasting and sales prediction [31], [32]), downstream modules focus more on workload trend in the near future, where there are sudden bursts. As a result, DBAugur further employs an ensemble algorithm to handle these problems.

VI. Experiments

In this section, we evaluate the performance of DBAugur for workload forecasting.

A. Workloads, Setup and Baselines

Workloads. We use two real-world workload traces as follows.

- *BusTracker Trace [1].* This dataset is collected from a mobile phone database application, which is used for timely tracking of the public transit bus system. The dataset size is about 800MB and it contains SQL queries from Nov. 29, 2016 to Jan. 25, 2017.

- *Alibaba Cluster Trace.*[3] This dataset is sampled from a production cluster and includes about 4000 machines in a period of 8 days. The dataset size is 8GB and it contains batch workloads collected in every machine in the cluster. We use the Disk utilization about six days as the third dataset.

Learning Setup. We implement the forecasting framework using Python3 and Keras with Tensorflow 2.0 as its backend. The experiments are performed on a Centos server with an Intel Xeon E5-2620 v3 CPU and 62 GB RAM. Adam [33] is used as stochastic gradient-based optimization to train the models. We use the first 70% of the dataset as the training set and the rest as the test set.

Baselines. We compare the performance of DBAugur with other classic and state-of-the-art models. The parameters of each model are determined by Grid Search [34]. Specifically, the learning rate is set to $1e - 3$.

- *Linear Regression (LR).* LR is a linear approach to model the relationship between history and future workload series.

- *Autoregressive Integrated Moving Average (ARIMA) [11].* This method combines autoregressive (AR) and moving average (MA). In the experiment, we set the parameters (p, d, q) in ARIMA to be $(2, 1, 2)$ respectively.

[3]https://github.com/alibaba/clusterdata

- *Multilayer Perceptron (MLP) [12].* We build a two layer MLP, with 32 units and 16 units respectively.

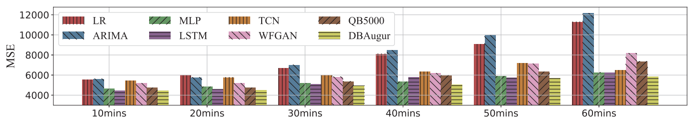
- *Long Short-Term Memory (LSTM) [13].* LSTM is a variation of RNN. The input length is set to 30, and the output dimension is set to 16 with a dense layer to get the final result.

- *Temporal Convolutional Networks (TCN) [15].* We create a five-layer TCN, where the dilated convolution factors are $1, 2, 4, 8, 16$ respectively.

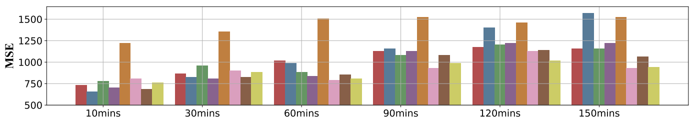
- *QB5000 [1].* QB5000 makes the forecast by equally averaging the results of LR, LSTM and KR.

- *WFGAN.* Both the generator and the discriminator are implemented by one LSTM layer with 30 cells, following a temporal attention layer.

- *DBAugur*[4]. DBAugur fuses the features of WFGAN, TCN, and MLP. As discussed in Section V-C, the value of attenuation factor δ is 0.9.



(a) BusTracker Workload



(b) Alibaba Cluster Workload

Fig. 5: Forecasting Model Evaluation

B. Forecasting Accuracy

In this section, we focus on DBAugur's ability to forecast the database workload. Mean Square Error (MSE) is used to quantify the forecasting accuracy, i.e., the average squared difference between the predicted workloads and real workloads, as is the case with other forecasting tasks [1], [35]. Figure 5 shows the results on traces of **BusTracker** and **Alibaba Cluster**, of which the forecasting interval is set to 10 minutes. The horizontal axis in the figure denotes the forecasting horizon, which is determined based on the distribution of each dataset.

As denoted in Figure 5(a), when the forecasting horizon is small, there is no significant difference in prediction accuracy for all the models. The reason is that both the linear models and the machine-learning based models can efficiently catch the workload patterns, allowing for reliable short-term prediction. However, traditional models, such as LR and ARIMA, lack the ability to match more complex workload patterns, leading to decreasing forecasting accuracy as the prediction horizon gets longer. Differently, machine-learning based models, such as MLP, LSTM, and TCN, demonstrate a higher capacity to fit workload patterns. As for QB5000, it is unable to maintain a high prediction accuracy as the forecasting horizon becomes larger. The reason is that LR's performance degrades sharply, thereby impacting the total prediction accuracy of QB5000. Different from Figure 5(a), traditional schemes, especially the LR model, make the good forecasting accuracy as denoted in Figure 5(b). The reason is that **Alibaba Cluster Trace** has good local linearity. As a result, a simple model can fit workload patterns effectively (except for bursts). Similarly, QB5000 that equipped with LR is slightly better than DBAugur when the horizon is small.

For WFGAN, it is able to fit data distribution by adversarial learning. With the strong generative ability and attention mechanism, WFGAN develops an efficient mapping to capture the correlation and burstiness of workload patterns. WFGAN achieves similar forecasting performance to TCN on **BusTracker Trace** while obtaining excellent forecasting accuracy on **Alibaba Cluster Trace** because of its specialty for bursts. Finally, the DBAugur model, which fuses WFGAN, TCN, and MLP, outperforms all the models under different prediction horizons on all of datasets. From Figure 5, we can see that DBAugur is capable of accurately forecasting the workload over both short and long term horizons. For diversified workloads, DBAugur can adapt to each model's strengths. DBAugur performs consistently and competitively for both simple and complicated workloads, demonstrating its generalization and adaptability.

C. Forecasting Horizon Evaluation

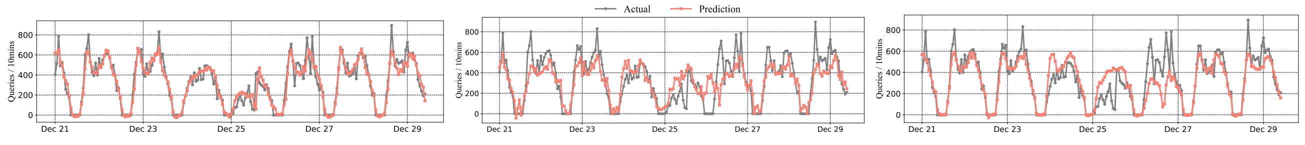
In this section, we explore the performance of DBAugur under different forecasting horizons on **BusTracker Trace**. The forecasting interval is set to 10 minutes. We respectively set the forecasting horizon to 60-minutes, 12-hours and 1-day. The result is shown in Figure 6. Figure 6(a) demonstrates forecasting results under 60-minutes horizon. We can see that the predicted workloads closely match the actual workloads, which means that DBAugur is adequate to make accurate forecasting ahead of time. Besides, DBAugur is able to adjust quickly to optimize its forecasts when the workload patterns change rapidly, i.e., the sudden spike in the figure.

Figure 6(b) shows forecasting results under 12-hours. Under this circumstance, DBAugur can fairly forecast the workload trend when the workload pattern is stable and less complex. As discussed before, if the forecasting horizon is large, learning the relationship between the historical workload data and the future workload is difficult. Therefore, the learning model cannot capture the accurate and sufficient knowledge from the historical workload in order to generate a precise prediction, causing forecasting capacity to deteriorate. For example, as denoted in Figure 6(b), we cannot respond quickly when workload patterns change suddenly. This scenario becomes more apparent in Figure 6(c) under 1-day horizon.

D. Computation and Storage Efficiency

In this section, we study the computation and storage cost of forecasting models. Since ARIMA is an on-time algorithm, we do not consider this algorithm. Besides, QB5000 and DBAugur can be can be derived from other models. Therefore, these two models are also not denoted. The results are shown in Table II. We record the CPU time of one epoch

[4]github.com/gaoyuanning/DBAugur



(a) 60-Minutes Horizon (b) 12-Hours Horizon (c) 1-Day Horizon

Fig. 6: Forecasting Horizon Evaluation – The horizontal axis denotes the time line.

during the training process, as well as the storage size of the trained models. Specifically, we set the number of epochs for LR and MLP to be 40, and the number of epochs for the remaining models to be 50.

TABLE II: Computation and Storage Efficiency

	CPU Time		Inference	Storage
	BusTrac	**AliClus**		
LR	1.471s	0.113s	4ms	29KB
MLP	2.638s	0.207s	5ms	29KB
LSTM	8.159s	1.454s	30ms	23KB
TCN	8.849s	1.577s	42ms	185KB
WFGAN	9.281s	3.707s	30ms	28KB

Table II demonstrates that simple models take less time to train. For example, LR takes only $1.471s$ and $0.113s$ when trained on two workloads respectively. Machine learning based models typically require more time to train. This training cost, however, is acceptable. To further improve training efficiency, we can early terminate the training as soon as models start to converge. As for the model storage cost, we can see that all of the models only use a little amount of memory to perform forecasting. Since the TCN model is deep and complex, it takes up a bigger space than other models. Besides, as shown in Table II, model inference is very fast compared to training process, which is negligible.

E. Ensemble Method Evaluation

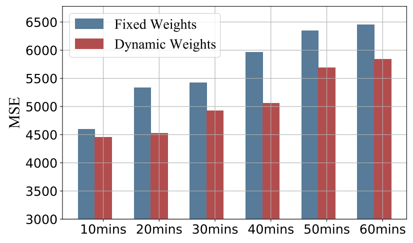
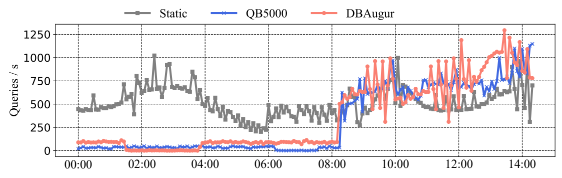


Fig. 7: Different Ensemble Methods

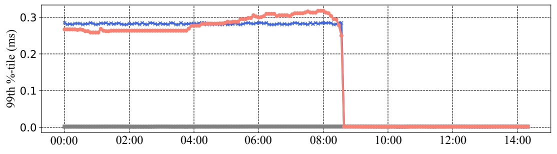
In this section, we evaluate the ensemble method on **BusTracker Trace**. Correspondingly, we respectively using dynamic ensemble weights and the fixed weights to generate the forecasting results. As discussed in Section V-C, we propose a time-sensitive ensemble algorithm that integrates the capabilities of WFGAN, TCN and MLP to produce the final forecasting results. As shown in Fig. 7, the dynamic ensemble method outperforms the fixed method both on short and long term forecasting horizons. This is because our algorithm dynamically calculates the ensemble weights based on each model's immediate forecasting ability, which is more adaptive. On the other hand, the static ensemble method is unable to fully utilize the capabilities of high-accuracy models in terms of the current workload patterns.

F. Case Study: Index Selection

We use a test case (i.e., automatic indexing selection) to illustrate how workload forecasting can facilitate the DBMS. In order to speed up query execution, it is necessary to create indexes on the database table. The index recommendation algorithm will select one or more indexes based on the historical workload, which we refer to as the **Static** strategy in this paper. In contrast, we can also leverage the predicted workload to make the index recommendation, which we refer to as the **Auto** strategy. In the experiment, AutoAdmin [36] is employed as the index recommendation algorithm. In the experiment, we leverage the simulator to perform SQL queries in **BusTracker Trace** on PostgreSQL-12, and input the historical and predicted workload to AutoAdmin respectively.



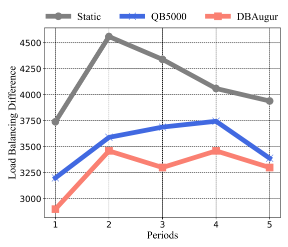
(a) Throughput



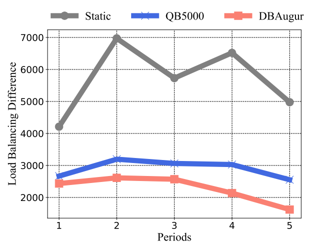
(b) Latency

Fig. 8: Index Selection

Figure 8 shows the results. In the experiment, the future workload generated by QB5000 or `DBAugur` is input into AutoAdmin after time 08:00. As a result, the query throughput under the **Auto** strategy is very low at the beginning of the time period in Figure 8(a). In contrast, the **Static** strategy achieves higher query performance because AutoAdmin will build the appropriate indexes for it in the beginning. Similarly, the **Static** strategy has a low latency in Figure 8(b). However, after time 08:00, AutoAdmin gradually builds the indexes with the guidance of the predicted workload. Correspondingly, the query performance under the **Auto** strategy improves over time and eventually surpasses that of the **Static** strategy. This is because with the input of the predicted workload, AutoAdmin can guide the database to build more precise indexes on proper columns to accelerate the query process, which demonstrates that accurate workload forecasting is beneficial to the database tuning process. Besides, as shown in Fig. 8(a), DBAugur is superior to QB5000 since it is more capable of forecasting long-term and complicated workloads.



(a) The Periodic Pattern



(b) The Complex Pattern

Fig. 9: Data Region Migration

G. Case Study: Data Region Migration

In this section, we conduct another case study. Assume that the database is partitioned horizontally into non-overlapping regions that assigned to each server. However, some servers may get overloaded from processing too many query requests. Therefore, we need to dynamically balance the system load by migrating data regions from the overloaded servers to slightly loaded ones. Traditionally, we input the historical workload data (i.e., **Static** strategy) into the load balancing algorithm to infer a global migration strategy. Historical workload, however, may be lagging and unadaptable to changing states. Alternatively, we can leverage forecasted workload to facilitate system load balancing (i.e., **Auto** strategy).

We generate two synthetic workloads in the experiment. The first workload follows periodic patterns. The second one has more complex patterns that incorporate linear trends, white noise, as well as seasonal, holiday, and weekday factors. At each period, we calculate the system's load balancing difference. We then use the **Static** or the **Auto** strategy to guide data migration in the next period. As shown in Fig. 9, the load balance of the system is bad when only using historical workload data, both for periodic workloads (i.e., Fig. 9(a)) and complex workloads (i.e., Fig. 9(b)). On the other hand, the system adopting the forecasted workload (i.e., QB5000 and DBAugur) to migrate data regions enjoys a good load balancing degree, because the migration scheme is prospective and thus adaptable to future states. Moreover, DBAugur outperforms QB5000 because it can offer more accurate future information, which aids the load balancing algorithm in making wise decisions.

VII. Related Work

Previous works on DBMS workload prediction include trend and performance prediction.

(1) Trend prediction. It mainly includes query arrival rate prediction and resource utilization prediction. Ma et al. [1] utilize clustering methods to deal with SQLs and propose an ensemble learning method that combines LR and RNN to predict the arrival rates. Huang et al. [37] leverage the multi-head attention mechanism and convolution operations to make probabilistic predictions. Taft et al. [38] adopt sparse periodic auto-regression to find out if load spikes occur. Liu et al. [39] propose a RNN based encoder-decoder model to predict future workload but they do not combine query arrival rates and resource utilization.

(2) Performance prediction. Raza et al. [40] propose a case-based reasoning approach to predict and adapt the DBMS workload performance. Singhal et al. [41] utilize a modular approach to estimate SQL query execution time for high data volumes. Marus et al. [42] generate a tree-structured neural network to model query execution plans and predict SQL performance. Zhou et al. [43] use a graph embedding network to encode the features and a prediction network to predict query performance. DBSeer [44] looks into the performance metrics (e.g., query latency and throughput) using statistical regression methods.

However, the above studies on trend prediction fail to effectively combine both query and resource forecasting, or use simple yet insufficient methods to model short- and long-term workload. Others focus on performance prediction incompatible with our research goals, but their workload-based features can still benefit from this work. Our work differs from existing studies as it provides an end-to-end framework that comprehensively considers more aspects of database workloads, enabling more workload temporal characteristics and leaving more space for other optimization tasks.

VIII. Conclusions

In this paper, we propose DBAugur, an adversarial-based trend forecasting system designed to predict the trends of diversified workloads. First, DBAugur collects query logs and resource utilization statistics, and formalizes them into different workload traces. To reduce the number of traces, DBAugur proposes an online DTW-based clustering algorithm which employs the DTW technique to efficiently cluster the workload traces according to their trend patterns. Next, we develop WFGAN that leverages the generative capability of GAN to forecast the database workload. To further capture the diversified workload patterns, a time-sensitive ensemble algorithm that fuses WFGAN, TCN, and MLP is proposed to produce the final prediction. Experiments on two real-world datasets demonstrate the efficiency and effectiveness of the proposed model. Furthermore, two case studies on index selection and data migration denote that DBAugur can effectively facilitate the management of the DBMS.

Acknowledgment

This work was supported by the National Key R&D Program of China [2020YFB1707900], the National Natural Science Foundation of China [62272302, 61972254, 61925205, 62232009, 62102215], Shanghai Municipal Science and Technology Major Project [2021SHZDZX0102], the Huawei Research Project [TC20220718012, CCF-Huawei DBIR2019002A], TAL education, and Beijing National Research Center for Information Science and Technology (BNRist). We also thank Gauss Department (Huawei Company) for their early contributions.

References

[1] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon, "Query-based workload forecasting for self-driving database management systems," in *ACM International Conference on Management of Data (SIGMOD)*, 2018, pp. 631–645.

[2] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *ACM International Conference on Management of Data (SIGMOD)*, 2017, pp. 1009–1024.

[3] G. Li, X. Zhou, S. Li, and B. Gao, "Qtune: A query-aware database tuning system with deep reinforcement learning," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 12, no. 12, pp. 2118–2130, 2019.

[4] J. Wang, I. Trummer, and D. Basu, "Udo: Universal database optimization using reinforcement learning," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 14, no. 13, pp. 3402–3414, 2021.

[5] L. Ma, B. Ding, S. Das, and et al, "Active learning for ML enhanced database systems," in *ACM International Conference on Management of Data (SIGMOD)*. ACM, 2020, pp. 175–191.

[6] J. Kossmann, S. Halfpap, M. Jankrift, and R. Schlosser, "Magic mirror in my hand, which is the best in the land? an experimental evaluation of index selection algorithms," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 13, no. 11, pp. 2382–2395, 2020.

[7] X. Liang, A. J. Elmore, and S. Krishnan, "Opportunistic view materialization with deep reinforcement learning," *CoRR*, vol. abs/1903.01363, 2019.

[8] H. Yuan, G. Li, L. Feng, and et al, "Automatic view generation with deep learning and reinforcement learning," in *IEEE International Conference on Data Engineering (ICDE)*, 2020, pp. 1501–1512.

[9] Z. Yan, J. Lu, N. Chainani, and C. Lin, "Workload-aware performance tuning for autonomous dbmss," in *IEEE International Conference on Data Engineering (ICDE)*, 2021, pp. 2365–2368.

[10] L. Ma, W. Zhang, J. Jiao, W. Wang, M. Butrovich, W. S. Lim, P. Menon, and A. Pavlo, "MB2: decomposed behavior modeling for self-driving database management systems," in *ACM International Conference on Management of Data (SIGMOD)*, 2021, pp. 1248–1261.

[11] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Elsevier Neurocomputing*, vol. 50, pp. 159–175, 2003.

[12] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009.

[13] L. Yunpeng, H. Di, B. Junpeng, and Q. Yong, "Multi-step ahead time series forecasting for different data patterns based on lstm recurrent neural network," in *IEEE Web Information Systems and Applications Conference (WISA)*, 2017, pp. 305–310.

[14] W. Härdle and P. Vieu, "Kernel regression smoothing of time series," *Journal of Time Series Analysis*, vol. 13, no. 3, pp. 209–232, 1992.

[15] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 2672–2680.

[17] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2021.

[18] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16, 1994, pp. 359–370.

[19] S. M. Omohundro, *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[20] P. A. Dinda and D. R. O'Hallaron, "An evaluation of linear models for host load prediction," in *IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 1999, pp. 87–96.

[21] A. Abanda, U. Mori, and J. A. Lozano, "A review on distance based time series classification," *Data Mining and Knowledge Discovery*, vol. 33, no. 2, pp. 378–412, 2019.

[22] C. A. Ratanamahatana and E. Keogh, "Making time-series classification more accurate using learned constraints," in *SIAM International Conference on Data Mining (SDM)*, 2004, pp. 11–22.

[23] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, vol. 96, no. 34, 1996, pp. 226–231.

[24] S. Wu, X. Xiao, Q. Ding, P. Zhao, W. Ying, and J. Huang, "Adversarial sparse transformer for time series forecasting," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 17 105–17 115.

[25] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[27] M. Hou, C. Xu, Z. Li, Y. Liu, W. Liu, E. Chen, and J. Bian, "Multi-granularity residual learning with confidence estimation for time series prediction," in *Proceedings of the ACM Web Conference (WWW)*, 2022, pp. 112–121.

[28] L. Xuecheng, "Entropy, distance measure and similarity measure of fuzzy sets and their relations," *Elsevier Fuzzy Sets and Systems*, vol. 52, no. 3, pp. 305–318, 1992.

[29] Y. Yang and K. Chen, "Temporal data clustering via weighted clustering ensemble with different representations," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 23, no. 2, pp. 307–320, 2010.

[30] A. Cano and B. Krawczyk, "Kappa updated ensemble for drifting data stream mining," *Machine Learning*, vol. 109, no. 1, pp. 175–218, 2020.

[31] V. Ekambaram, K. Manglik, S. Mukherjee, S. S. K. Sajja, S. Dwivedi, and V. Raykar, "Attention based multi-modal new product sales time-series forecasting," in *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, Eds. ACM, 2020, pp. 3110–3118.

[32] Z. Wang, X. Xu, G. Trajcevski, K. Zhang, T. Zhong, and F. Zhou, "Pref: Probabilistic electricity forecasting via copula-augmented state space model," *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 12 200–12 207, 2022.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR Poster)*, 2015.

[34] Y. Sun, S. Ding, Z. Zhang, and W. Jia, "An improved grid search algorithm to optimize svr for prediction," *Soft Computing*, vol. 25, no. 7, pp. 5633–5644, 2021.

[35] A. S. Higginson, M. Dediu, O. Arsene, N. W. Paton, and S. M. Embury, "Database workload capacity planning using time series analysis and machine learning," in *ACM International Conference on Management of Data (SIGMOD)*, 2020, pp. 769–783.

[36] S. Chaudhuri and V. R. Narasayya, "An efficient, cost-driven index selection tool for microsoft sql server," in *International Conference on Very Large Data Bases (VLDB)*, vol. 97, 1997, pp. 146–155.

[37] X. Huang, S. Cao, Y. Gao, X. Gao, and G. Chen, "Lightpro: Lightweight probabilistic workload prediction framework for database-as-a-service," in *IEEE International Conference on Web Services (ICWS)*, 2022, pp. 1–13.

[38] R. Taft, N. El-Sayed, M. Serafini, Y. Lu, A. Aboulnaga, M. Stonebraker, R. Mayerhofer, and F. Andrade, "P-store: An elastic database system with predictive provisioning," in *ACM International Conference on Management of Data (SIGMOD)*, 2018, pp. 205–219.

[39] C. Liu, W. Mao, Y. Gao, X. Gao, S. Li, and G. Chen, "Adaptive recollected rnn for workload forecasting in database-as-a-service," in *International Conference on Service-Oriented Computing (ICSOC)*, 2020, pp. 431–438.

[40] B. Raza, Y. J. Kumar, A. K. Malik, A. Anjum, and M. Faheem, "Performance prediction and adaptation for database management system workload using case-based reasoning approach," *Information Systems*, vol. 76, pp. 46–58, 2018.

[41] R. Singhal and M. K. Nambiar, "Predicting SQL query execution time for large data volume," in *ACM International Database Engineering & Applications Symposium (IDEAS)*, 2016, pp. 378–385.

[42] R. Marcus and O. Papaemmanouil, "Plan-structured deep neural network models for query performance prediction," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 12, no. 11, pp. 1733–1746, 2019.

[43] X. Zhou, J. Sun, G. Li, and J. Feng, "Query performance prediction for concurrent queries using graph embedding," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 13, no. 9, pp. 1416–1428, 2020.

[44] B. Mozafari, C. Curino, A. Jindal, and S. Madden, "Performance and resource modeling in highly-concurrent oltp workloads," in *ACM International Conference on Management of Data (SIGMOD)*, 2013, pp. 301–312.