Workload Consumption Metric Forecasting

Naveen Sankaran, Sourabh Maity, Lovlean Arora, Bhashyam Ramesh

Teradata India Pvt. Ltd

Corresponding author: Naveen Sankaran, Email: naveen.thaliyilsankaran@teradata.com

The success of an autonomous database largely depends on its ability to predict the future. With the knowledge of what is going to happen in the future, an autonomous database can choose proper optimization strategies at the right time. This helps an autonomous database to be proactive rather than being reactive. Here, in this paper, we will concentrate on workload management task of a database management system. There can be multiple workloads that are active during a time period with each workload having different priorities. Amount of resources available to lower priority workloads depend upon the load on higher priority workloads. If a system can model its user's workload consumption from history, it can use this knowledge to serve the following use cases: 1. Depending on the system load, dynamically scale up or scale down the system capacity. 2. Dynamically allocate resources to workload groups such that overall system performance is optimized. In this paper we present a predictive system which predicts the following metrics for each workload groups for the next (few) timestep(s): 1. Percentage of total CPU that a workload group is going to consume in the specified timestep. 2. Amount of IO (data) in KB that a workload group is going to consume in the specified timestep. 3. Number of queries which are going to arrive for a workload group in the specified timestep. The use cases mentioned above can be achieved once we can predict the workload group metrics listed above. Using the predicted values to dynamically scale up/down system capacity or dynamically allocate resources to workloads is out of scope of current work.

Keywords: Workload management, Database management system, Dynamically allocation

2023. In Saroj Hiranwal & Garima Mathur (eds.), *Artificial Intelligence and Communication Technologies*, 1093–1111. Computing & Intelligent Systems, SCRS, India. https://doi.org/10.52458/978-81-955020-5-9-104

1. Introduction

Faced with ever growing volumes of data, owing to variances in the origin of data, the format of data, data silos, and the like, storage requirements increase, and database administrators often find themselves increasingly challenged by elevations in performance goals to be met. To ensure queries meet their service level goals (SLGs), DBAs have to constantly update indexes, collect table stats etc. They have to constantly monitor their database for any performance issues, perform root cause analysis for the reported issues and take prompt actions to rectify the issue. Along with this, DBAs must also maintain a proper workload management system to ensure that all queries receive optimal resources to maximize overall system throughput. Loading external data into a database after cleaning and transforming the data and identifying its schema etc. involves another set of challenges for DBAs.

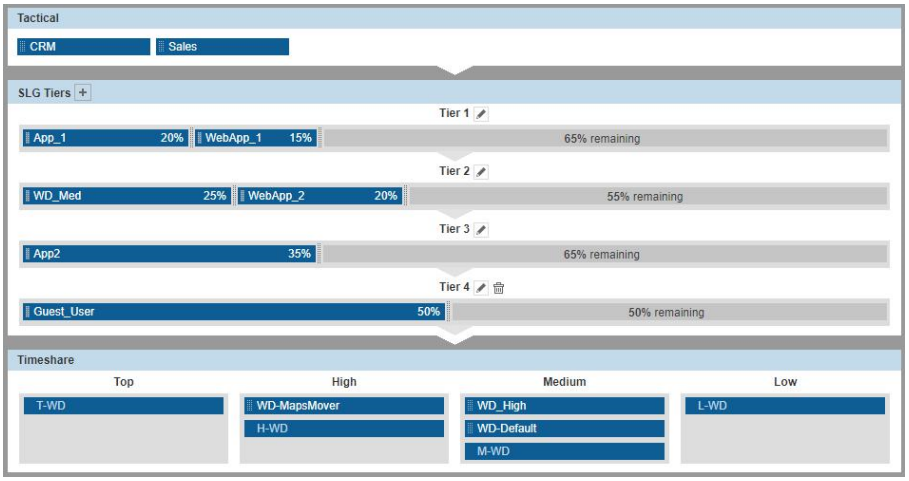


Fig 1: A typical workload hierarchy is shown in this figure. There are three major tiers namely Tactical, SLG and Timeshare. Each workload can be assigned to a tier and provided with resource percentage limits. Lower tiers can consume only left over resources after consumption by upper tiers.

It is due to such reasons that a lot of research is directed towards automation of database operations and to make database a 'self-driven' entity with zero or very low interference by DBAs [2, 3]. An autonomous workload management system (AWMS) should run all by itself, be 'driver-less', and should take all the data management decisions by itself to provide the optimal performance for user's workloads. Peloton [17] is a self-driven database which is getting built component by component by students and researchers at Carnegie Mellon Database Research Group. However, it is still at its initial stages. Similar efforts are also being driven by industry. IBM has developed a proof-of-concept for DB2 database [19] and Oracle Corp. has developed similar system for their Oracle database [5, 12]. SageDB [11] uses machine learning to model data distribution, workloads and hardware to provide optimal access methods and query plans.

DBSeer [20, 15] provides a solution for database administration using machine learning by which they provide tools like automatic workload detection and classification, anomaly detection etc. They provide answers to "what-if" questions on workloads by which they try to model database performance. Das et. al. [4] have designed a hierarchy of rules by which they can perform auto-scaling of database in a cloud

environment. They have used their knowledge of database-as-a-service to define these rules. Gong et. al. [6] provides auto scaling by estimating if there will be a resource demand in near future. Ma et. al. have come up with a workload query forecasting system [13] in which they can predict the number of queries arriving for a workload in future hour based on previous query arrival pattern for that workload.

However, none of the above works focus on resource prediction when workloads having different priority are running in parallel. Existing prediction systems either consider single workload/query or multiple queries having same priority. Scaling a system which predicts for a single workload to a system which predicts multiple workload (with each workload having different priority) running in parallel is non-trivial as resources which a workload gets may depend on other active workloads in the environment. A workload consumes resources which is a function of what it demands, what is allocated to it, what resources are consumed by higher tier queries and what is left for it to use. This will be explained in later sections.

Our current work focuses on forecasting different workload metrics like CPU/IO consumption, query arrival rate etc. for a workload in an environment where multiple workloads of different priority are running in parallel. How such predicted metrics can be consumed by other systems are out of scope for this work. We have considered a mixed-workload setting in which there are multiple workloads running in the system and CPU consumed by one workload will depend on the query load present in other workloads. A multi-workload setting was chosen as most of the large database users will have multiple workloads for proper resource management. The forecasting will be done for short term and long term to help users plan better.

A general workload management set up process is presented below. We take Teradata Active System Management (TASM) to explain the details of the set-up process, but the process (with minor changes) can be applied to any other workload management system.

In current workload management system, user must specify three main things:

1. Classification rules to define workload: Among all incoming queries, there can be many sub-groups of queries having some commonality among them. This might be due to nature of queries or due to any business rules. User must set up rules to identify and map these sub-groups into different workload groups. This helps user to manage and control queries easily. There are multiple ways in which user can create workload classification rules like source of query (eg: sales team), query characteristics (number of joins etc.) or query target (specific tables etc.).

2. Priority of defined workloads: User must set the priority of each of the workload group. The basic need for assigning different priorities can be well explained by the following intuition: For example, the Tactical queries will be given higher access to priority than the DSS queries in Timeshare. In case of TASM, a user must place the defined workload groups into either of the following tiers (in descending order of priority), Tactical, SLG or TimeShare. SLG and Timeshare tiers can be further sub-divided to create sub-tiers. Tactical tier is for very high priority workloads that are highly tuned. SLG Tier are used for workloads supporting high priority queries with service level goals (SLGs) and Timeshare tiers are used for standard or background workloads like data load. Multiple workloads can be assigned to same tier and workloads within same tier have the same priority. Figure 1 shows priority order of tiers for TASM in which we have shown three major tiers and some sample workloads assigned to those tiers.

3. Allocate relative share of resources (RelShare) and Service Level Goal (SLG) for the workloads: Each workload group may have different resource demand and SLG expectations. The basic need for assigning different RelShare and SLG can be well explained by the following intuition: The tactical queries will consume less CPU, will have less elapsed time and quicker service level goal than the queries that are intended for some complex data analysis. A user must specify the relative percentage of resources allocated to each workload groups. For TASM, the resources a user considers are CPU share percentages and IO share percentages. The tiers function in a waterfall model wherein the bottom tiers get leftover resources from the tiers above it; e.g., assume tier 1 is allocated 70% resources and tier 2, which is below tier 1, is allocated

50% resources. This means that tier 2 gets 50% of leftover resources from tier 1. i.e., 50% of (100-70) % = 15% system resource. If this allocation is not done carefully, this may lead to starvation in lower tiers and can lead to SLG slippage by workloads present in lower tiers.

2. Motivation

Among the three tasks mentioned above, the first two tasks (workload classification and assigning priority to workloads) requires significant knowledge of the user's business rule. This is because users can decide on an arbitrary rule like 'all queries from sales team should be classified into single workload' or 'all queries from this specific web application should be in a single workload'. This may result in having different type of queries (light weight as well as heavy) belonging to same workload. Thus different query clustering methods which suggest workloads to users may not be effective as they look at query characteristics and ignore the business rules. This is also the case with assigning workload priority as users can decide the priority based on certain business reasons. Hence, we decided to have our initial focus on helping the user with the task of allocating resources to each workloads. We shall provide a mechanism by which users can have an idea on what will happen in (immediate) future so that they can plan better.

In current workload management system, DBAs set resource allocation for workloads based on historical observations of the load pattern for the workload. However, the process is mostly heuristic-driven in nature and multiple trial and error are required before one identifies an ideal resource share allocation for workloads. And once the resource limits are set, they are not changed frequently. There are two main use cases by which we can automate resource allocation process:

1. Dynamically scale up or scale down the system capacity: Depending on the expected load, an autonomous workload management system (AWMS) can scale the database resources up or down to meet the performance goals of the workloads. This capability to dynamically scale up and down at right time for the right duration is of utmost importance when the database is deployed in cloud environment to make maximum value out of "pay-as-you-use" approach of the cloud service providers. i.e: If users can know in advance that there will be an increased demand for resources but only for next one hour, they can decide if it is better to let workloads miss their SLG for the next one hour instead of scaling up the system which may be expensive.

2. Allocate resource to workload groups: Depending on the arrival of queries in different workload groups, resources allocated to those workload groups can be tuned dynamically to maximize the number of queries meeting their service level goals (SLG). During situations when there is resource constraint resulting in queries missing SLG, we need to optimize resources in such a way that overall system performance is optimized. We define system performance as maximum queries meeting their SLGs weighted according to their priority. So, if users can know in advance which workloads will consume resources in the next hour, they can take resources from other workloads and assign it to those workloads whose resource demands will increase. This helps in optimizing system resources.

To be able to serve the above use-cases, the first problem to solve is: "The autonomous workload management system (AWMS) must be able to predict the future load for each workload group. Being able to foresee the future will enable it to make better decisions".

In this paper, we present a predictive system which predicts workload group metrics by learning history pattern of workload load. Thus, this predictive system makes an AWMS proactive, rather than the current reactive approach. The systems which can consume the predicted values will be out of scope of current work.

3. Problem Formulation

We are presenting a predictive system which predicts the following metrics for each workload group per timestep:

1. Percentage of total CPU a workload is going to consume in the specified timestep.
2. Amount of IO in KB a workload is going to submit for consumption in the specified timestep. In this paper, we use the term IO Submitted and IO Consumed interchangeably.
3. Number of queries which are going to arrive for a workload in the specified timestep.

This is the minimal set of metrics that must be predicted to serve the use-cases mentioned in section 2. However, it can be easily extended to predict any other metric (eg: number of worker threads). The first two metrics are the core resources for a database system and can be controlled by the resource allocation value. To achieve optimal system performance by dynamically allocating resources, one must predict query arrival rate.

One way to formulate this problem would be to consider current system parameters and predict what will happen next. However, we have seen that representing the input as a series of events ordered across uniform time intervals (time series) provided us with better results. Formulating the problem as a time series problem will also helps us in capturing the seasonality of the input pattern (eg: day vs night, weekend vs weekday etc.) and identify if the target variables are autocorrelated or not. Hence, we decided to formulate the prediction problem as a time-series problem.

As mentioned in above section, we define system performance as maximum queries meeting their SLGs weighted according to their priority. Here we use the word timestep to represent an "interval of time", e.g., an hour, 10 minute etc. Considerations related to the choice of the value of the time interval are:

- It should not be too small an interval: Here we try to predict the resources consumed by queries/workload groups; an interval of second or millisecond order is too granular for the purpose.
- It should not be too long an interval: If the time interval is too long then target values may be averaged out to be similar at every prediction point.

We recommend using a time interval no less than 10 minute and no greater than 3 hours. For our current experiment, we have chosen to extract features per hour and predict system parameters for next hour(s). To emphasize on the generality of the solution, we decided to use a generic word timestep throughout the paper.

4. Solution

In this section, we explain the processes involved in our solution, right from data extraction till prediction of the target variables. We introduce our solution pipeline first and then proceed to explain each module in detail. Our solution architecture is shown in Figure 2. We have three main modules in our architecture namely: Data Module, Training Module and Prediction Module. We explain each of these modules in detail in below sections.

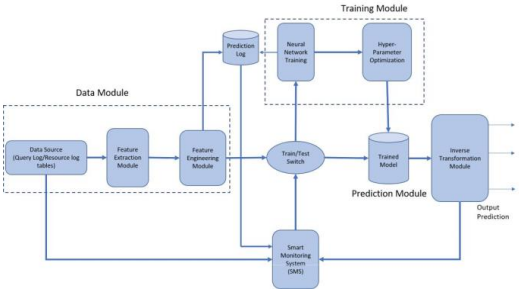


Fig 2: Solution pipeline: Major modules are Data Module, Training module and Prediction module.

4.1. Data Module

4.1.1 Data Sources

This module is responsible for extracting data from database. We collect data from three sources:

1. Query Logging Tables: These tables have information regarding query arrival rates, which workload does it belong to, how much was query elapsed time and was it meeting its SLG.

2. Resource Usage tables: We extract different resource usage parameters per workload from these tables. Some of the parameters which we extract include CPU Utilization, IO consump-tion, worker thread consumption (AWTs) etc.

3. Teradata workload Management (TDWM) tables: TASM definitions present in TDWM tables are used to identify the workload groups, the tier which the workload groups belong to, their priority, relative resource share for the workload groups and their Service level Goals (SLGs).

4.1.2 Feature Extraction

The Feature Extraction module extracts features from the dataset. Since, we want to predict CPU, IO and query arrival rate (ARM) for each workload, we extract all such features which directly or indirectly impact them. Keeping this in mind, we extract the below mentioned 46 features for each workload:

1. CPU Related features (2 features):

- CPU consumed, CPU wait
- Since, we want to allocate resource share for CPU, we need to predict how much CPU will be consumed. CPU consumed is represented as percentage of total CPU available in entire system.
- A CPU Wait captures how much was scarcity for the CPU and hence captures total CPU demand. This CPU demand is needed for dynamically scaling and descaling system capacity.

2. IO Related features (10 features)

- Logical IO submitted and completed represented as total number of IO requests (IO count), total IO requested in KB and average KB per IO request. We have used three representations to distinguish between multiple IO requests each demanding very small IO Vs single IO request demanding large amount of IO.
- Physical IO in count, in KB, in average KB per IO and IO wait in milliseconds
- Logical IO captures total IO demand. Some IO requests may be fulfilled via cache and queries need not actually do physical IO. So, we capture both physical and logical IO. The difference between logical and physical IO is that logical IO captures cache effect and hence indirectly measures the effect of concurrency in queries (more the concurrency, more the cache purging, more the difference between physical and logical IO)

3. Arrival Rate Metrics (ARM) Related features (2 features)

- Number of Queries arrived and number of queries which completed the execution
- These together captures complexity of queries along with how many queries are spilled over to next hour and thereby impacting load in next hour.

4. Worker thread (AWT) Related features (25 features)

- We have 6 worker thread classes[1] namely New, Spawn, Utilities, Expedited, Abort, Misc. From each of these worker thread classes, we have extracted 3 features: InUse, Max, Exhausted giving us total 18 features.

• We have also extracted Total InUse, Exhausted and Max across all 6 classes which are mentioned above. (Total 3 features)
• Worker threads Assigned, Released, wait time and average wait time. (Total 4 features)
• A query uses many worker threads as it comprises of many steps. Also, the steps in a query can be running in parallel. These numbers capture how many worker threads are being used (InUse) in whole system, what is the maximum number of worker threads used (Max), and how many times there was shortage for worker threads (Exhausted) and for how long this shortage lasted (AWT wait time). These capture the effect of complexity (less queries and more worker threads imply complex queries) and concurrency in the system.

5. Lock wait and other waits (6 features)

• Data block lock, lock for memory segments aggregated into single feature. This is then expressed as total number of locks requested (lock count), lock time millisecond and average wait time in millisecond per lock wait. These features capture concurrency especially if queries/steps are fighting for same resources.

6. Number of active sessions (1 feature)

• More number of active sessions indicate more number of concurrent queries.

As, already discussed, tiers behave in waterfall model with respect to resources, i.e. the left-over resources from top tier are passed to lower tiers. Tactical queries have privilege of using whatever resources they want. For the workload groups in other tiers, TASM allocates resources per workload to control how much maximum resources they can consume in a heavily loaded system from whatever is left over from upper tiers. So, a workload consumes resources which is a function of what it demands, what is allocated to it, what resources are consumed by higher tier queries and what is left for it to use.

Hence, the feature vector for each workload should have features from all other workloads as well. However, this could lead issues in case new workloads are added to system or if existing workloads are deleted/deactivated or if workload priority changes. In such cases, the workload models have to be retrained to adjust to changed environment as feature vector length will change. An easy way to resolve this issue is to group workloads into bins. This will ensure that the feature length is constant and the effect of adding/removing workloads will be smoothed out.

Hence, for each workload under consideration, i.e. for the workload for which we are predicting resource consumption parameters, we will have a feature vector that finds same features for four bins as mentioned below:

1. Workload under consideration (denoted by _WD, e.g.: 'CPU_LOAD_MSEC_WD' denotes total CPU load in milliseconds for workload under consideration).

2. All workloads having higher priority than workload under consideration (denoted by _HP).

3. All workloads having lower priority than workload under consideration (denoted by _LP).

4. All workloads which belong to same tier(siblings) as that of workload under consideration (denoted by _SP).

This increases the total number of features from 46 to 184 (= 46*4). Additional features which denote day of week (numbered 1-7) and hour of day (numbered 0-23) is used to capture patterns which are specific to day of week (weekday Vs weekend) or time of day (morning Vs night). These are required to capture patterns specific to days (eg: Weekend load Vs weekday load) and time (eg: night Vs day time load). This brings our final feature count to 186. For day of the week feature, value 1 is assigned to the first working day of the week (and not Monday/Sunday specifically) as different regions in world can have different days as start of week.

Kindly note that the four bins mentioned above many not be applicable for all workloads. e.g: Tactical workloads will not have any workloads having more priority than them. So, bin 2 features are not required for them. Similarly, some workloads may not have any sibling workloads (bin 4) and hence we won't be using those features. So, different workloads will have different number of total features extracted.

4.1.3 Feature Engineering

Feature Engineering module is responsible for reducing the number of features and normalizing the features. There are four steps involved in reducing feature dimension. These are listed below:

1. Skew removal: A feature is said to be having skewed distribution if its skewness factor is less than -1 or greater than +1. In such cases, we apply cube-root transformation to remove the skew.

2. We then proceed to standardize the feature values by removing the mean and scaling it to unit variance. This is especially useful for regression using neural networks. For our experiments, we have tried both Standard Scaling and Min-Max Scaling for feature normalization and have chosen Min-Max scaling for our experiments.

3. Encoding Cyclical features: Features like hour_of_day present us with different challenge during normalization as they are cyclic in nature,i.e: after 23 hours, the next value is 00 and not 24.This is shown in Figure 3 in which we can see that even though hour 23 of day 1 and hour 01 of day 2 re actually closer (difference of 2 hours), when we represent them as raw values, they look much further apart. Hence, we encode the hour value using the formula mentioned below:

$$\text{sin\_enc\_hour\_o f \_day} = \sin(2 * \pi * \text{hour\_o f \_day}/24)$$

Using the above mentioned encoding we can see that, $\sin(20) = -0.86$, $\sin(23) = -0.25$ and $\sin(01) = 0.25$ and thus ensuring that hours 23 and 01 are more closer than hours 20 and 23. This is shown in Figure 4. However, the above sine encoding too have some issues as $\sin(02) = \sin(10) = 0.5$. So, we use an additional cosine encoding for hour_of_day feature thereby using two features to represent a single hour in a day. Cosine encoding is done using the below mentioned formula:

$$\text{cos\_enc\_hour\_o f \_day} = \cos(2 * \pi * \text{hour\_o f \_day}/24)$$

Figure 5 shows how hours in a day will be after they are encoded using both sine and cosine encoding.

4. The present set of about 180 skew-corrected and normalized features needs to be reduced further down to a more reasonable number. For this, we have used Principal Component Analysis (PCA) [9] to reduce the feature from about 180 to around 15. During PCA, we pick only those features which cumulatively retain 99% of variance.

4.2. Training Module

For this prototype, we would be predicting the CPU Consumption in percentage, IO Consumed in KB and Query Arrival Rate for a given workload. A single neural network model will be built per workload to predict these values. We decided to have one model per workload rather than a single model for entire system as:

1. Each workload can have different data distribution and different number of features and hence has to be modelled individually.

2. In case the data pattern has changed over time for a workload, the corresponding model can be retrained while predictions for other workloads continue to function as usual.

3. Handling addition/deletion of workloads would be easier.

Our input is a sequence of data points ordered by time. What happened during last hours will influence what will happen in future. Queries which had been running in previous hour but have not yet completed can affect resource consumption for current hour. There can also be patterns which gets repeated like light

load during an hour just before a business opens in morning or light load during lunch time etc. Commonly used statistical methods like ARIMA and SARIMA [1] are more suited towards uni-variate time-series forecasting. Other machine learning models like decision trees and random forests are unable to capture the properties of timeseries properly. Besides, they are commonly used for classification than for regression. Hence, we decided to use Recurrent neural networks (RNNs) [14] as such networks are known to work well with timeseries data. For the current prototype, we use a specific form of RNN known as Long Short Term Network (LSTM) [8]. This is because LSTMs are known to handle issues related to Vanishing Gradient [7] very well. Figure 7 shows the distribution of errors (RMSE) for a single workload across 9 future predictions. As can be seen from the figure, neural network models perform better than statistical models for multi-variate roll-over prediction for timeseries data. Do note that the errors shown here is the average RMSE error for CPU, IO and query arrival rate.

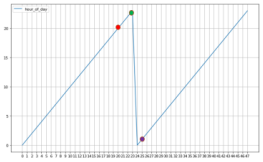


Fig 3: Plot of hours_of_day for two days. Hour 20 and hour 23 of day 1 are actu-ally further away than hour 23 of day 1 and hour 01 of day 2

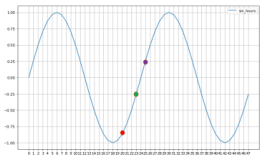


Fig 4: Sine encoding of hour_of_day solves the above mentioned issue

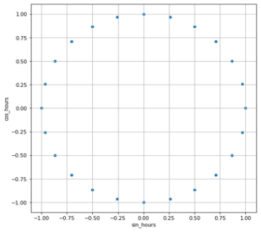


Fig Fig. 5: A day represented using sine and cosine encoding of hours. The distance be-tween two points is propor-tional to the difference in time as we expect from a 24-hour cycle.

Fig 6: We show how raw representation of hours can be misleading and hence cyclic features like hours has to be encoded using sine and cosine functions for better representation.

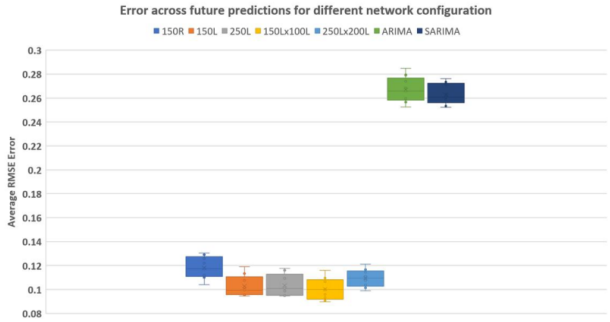


Fig 7: Different network configurations which we have tried to identify the best configuration. R denotes RNN layer and L denotes LSTM layer. 150Lx100L denotes 2 hidden LSTM layers having 150 and 100 nodes respectively. Y axis denotes the distribution of average RMSE error across 9 future prediction for 3 metrics for which we do prediction.

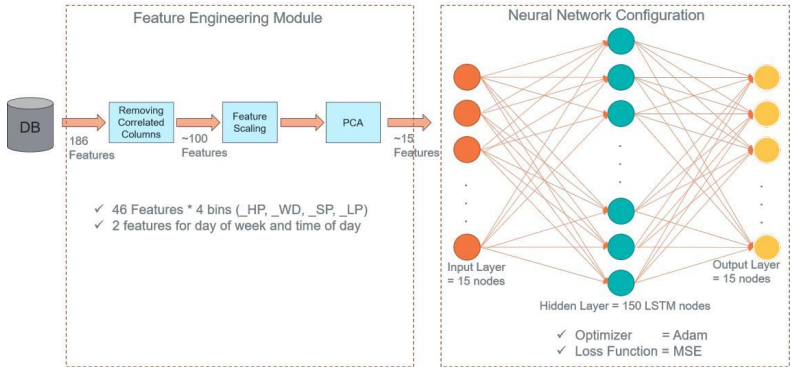


Fig 8: Figure showing general summary of our process. Correlated features extracted from database are removed which then undergoes PCA to reduce dimensions. These features are sent to a neural network model which is used to predict system parameters for future. Number of features and network configuration may change slightly for different workloads.

Another point to consider while building the model is how much previous timesteps should be provided to the model so that it can predict the next (future) timestep. For this, we have used Auto Correlation Function (ACF) [18] plot to identify how much history is ideally needed to predict the future value. We can use ACF to identify the range of lag values with which to experiment with as ACF may not provide us with the exact lag value to be used for our experiments. The series was made stationary before we calculated ACF.

An ACF plot tells you the lag value with autocorrelation as it is the similarity between observations as a function of the time lag between them. For current experiment, we have chosen 10 previous history points to predict future. This number was arrived from running multiple experiments and choosing the best value.

The trained model is stored locally to be used during prediction. Figure 8 shows a general summary of our training process. Kindly note that number of features and network configuration may change depending on the workload for which training was initiated. The output of the model is passed through an inverse PCA to reconstruct original variables.

4.2.1 Prediction Model

As mentioned above, we will be using an LSTM model for our prediction system. We have tried multiple network configurations and the best performing configuration had a single hidden layer with 150 LSTM units. Different network configurations which was tested is shown in Figure 7. As can be seen from the figure, having a deeper configuration (2 layers) or increasing the number of nodes in a layer did not improve the accuracy by much. Hence we decided to use a network with single LSTM hidden layer having 150 nodes.

Mean Squared Error was the loss function along with Adam [10] as optimizer. Early stopping was applied to finish training quickly. Our system can take multiple network configurations (in form on JSON file) as input, run training for all these model configurations and chose the best model depending on accuracy.
The best model is deployed for prediction. Thus, different workload models can have different network configurations.

4.2.2 Evaluation Metrics

We have used Root Mean Squared Error (RMSE) as the evaluation criteria. As mentioned earlier, for a given workload, we will be predicting three system parameters using our prediction module:

1. CPU Load Consumption as a percentage of total CPU available in system
2. IO Consumption in KB
3. Query Arrival Rate Metrics (ARM)

4.3. Prediction Module

During prediction, we extract features for past 10 hours and pass them through the feature engineering module as mentioned above. These extracted features are then sent to the trained model to predict workload parameters for future 9 hours. We apply inverse feature transformation on the generated outputs to convert the scaled features into actual values. For PCA, converting the output dimension to its original dimension will result in some loss of accuracy. However, we feel this is within acceptable limits for our use cases.

A sliding window approach with step size of 1 will be used for prediction. i.e:

- To predict the immediate future hour (N+1), we use previous 10 values i.e: N-9, N-8, ..., N-1, N.
- To predict N + 3 hour, we use N-7, N-6, ..., N+1, N+2 as input where N+1 and N+2 are predicted values while other points(N-7...N) are actual values.

4.4. Smart Monitoring System (SMS)

One challenge with continuous prediction is that the underlying data distribution upon which we perform the prediction may change over time. Hence, we have a smart monitoring system which will continue to monitor our predictions and compare them with actual values for any deviations. The predicted values are logged in an internal log table and the actual values are obtained from database. i.e: Assume our model has predicted that in the next hour, CPU consumed by a workload will increase by 20%. After the next hour, our SMS module extracted logs for the workload from database and realized that the CPU consumption increased only by 5%. This means that our prediction was off for the workload. If the difference between predicted values and actual values increase over a certain threshold, our monitoring system can either inform the user regarding the deviation observed in prediction or else it can initiate auto-retraining of the workload under consideration using the Train/Test switch module. During our experiments, we have observed that re-training of workloads are required not more than once every month.

Table 1: Table showing single stream run of our benchmarks and their corresponding tier and resource allocation. Note that Tactical queries can use all available resources and hence a resource limit cannot be set. No. of Queries shown here represents the unique queries. These were picked in random and executed repeatedly over multiple hours to simulate a workload.

WDID	No. unique Queries	Avg. CPU per query	Queries / hour
1	1300	8.49	433
2	289	52.36	20
3	1367	145.23	227
4	103	5643.51	12
5	48	1452.56	7

(a) Single stream run results for the 5 benchmarks resource which we have used each

WDID	TIER	Resource Allocation Percentage
1	Tactical	N.A.
2	SGL 1	25
3	SLG 2	40
4	SLG 3	50
5	SLG 3	40

(b) The corresponding tier and allocation percentage for benchmark

Table 2: Workload details of the customer dataset which we used for our experiments

Tier	Workload ID	No. of queries	Avg. CPU per query
Tactical	1	168,395	3.86
SLG 1	2	24,315	128.64
SLG 2	3	51,642	253.85

5. Dataset

For our experiments, we require a multi-workload dataset on which we can see how load on a high priority workload will have impact on lower priority workload. We should have a mix of workloads which are not homogeneous across them. Some workloads should have quick running queries while others should have heavy queries. We would like to have an environment in which such workloads are running in parallel. We have used two sets of data for current set of experiments, a customer dataset (CD) having three workloads and a synthetic dataset (SD) having five workloads.

5.1. Customer Dataset (CD)

This data was extracted from a customer having three active workloads, each belonging to tactical, SLG tier 1 and SLG tier 2. For our experiments, we have directly extracted the workload features from the query log tables while ensuring no PII (Personally Identifiable Information) information was extracted. Details of the dataset is shown in Table 2. As can be observed, majority of the customer queries are fast running tactical queries.

5.2. Synthetic Dataset (SD)

To generate synthetic dataset, we have used 5 benchmark datasets and assigned each benchmark to a workload. These data sets include publicly available datasets like TPC-DS [16] and some of our internal benchmark datasets. The internal dataset contain queries from customer data (after sanitizing them by removing all sensitive information) and can be considered as representation of type of workloads which is used in the real world.

These benchmarks were ran individually in a single session in our test system to have an idea about query characteristics. The results of our single stream runs are shown in Table 1a. Using this information, we have assigned each workload to a tier. As mentioned in Section 1, lower tiers can consume only the left over resources from upper tier and tactical tier have no limit on the amount of resources it can consume. Hence, heavy workloads are assigned to lower tiers while quick running workloads(w.r.t elapsed time) are assigned to upper tiers. Similar pattern was observed on customer workloads as well and we have decided to practice the same four our synthetic workloads as well. Accordingly, we have assigned the workloads into tiers and this is shown in Table 1b. As can been seen from the table, workload 2 is assigned into tier 1 and have a resource limit of 25% of system resource. Remaining 75% system resource is available for lower tiers from which tier 2 can consume maximum of 40% which is 30% of actual system resources (75*40/100).

Armed with above information, we built a data generation tool which can be customized to generate different types of data patterns. Our data generation tool provides users with multiple configuration options. Users can configure the following:

1. Number of concurrent sessions: Users can decide how much concurrent sessions can be started per workload.
2. Query Arrival Rate (ARM): Users can specify how many queries can arrive per workload per unit of time.
3. Load pattern of workload: Users can specify the load pattern for a workload for a given day.
 eg: Heavy load during day and light load during night.

4. Users can also specify load pattern for a workload for a week. i.e: Users can specify workload 1 will have maximum load on Monday and minimum load on Sunday.

To generate the required pattern, we cluster the workload queries into different bins depending on the resources consumed by queries. If a workload is expected to have maximum load on hour 10 of day 3, majority of the queries which is to be run will be picked from the bin having heaviest queries. The queries are picked up in random so that exact same pattern doesn't appear for two separate weeks. The queries thus selected are equally distributed among the concurrent sessions. Even though users can specify query arrival rate, we do not pick this exact number. We randomly pick number of queries within a range of ±0.1 of what user has provided.

Once we decided on the pattern, we ran the actual queries in our test machine to generate the required pattern. Even though the query characteristics were based on individual runs, we were able to achieve the overall pattern which we desired for each workload. One challenge which we encountered was that of session overflow which can occur in case some sessions were not completed within the required time. In such cases, we waited till the sessions were finished before we launched new sessions. eg: For a workload, in first hour, we had launched 5 sessions. However, by the end of first hour, only 4 sessions were completed and one session was still running. So, at the beginning of second hour, we launched 4 new sessions for the workload and waited for some time for the pending session from first hour to be completed before launching the 5^{th} session for second hour. This will also have effect on query arrival rate as if not all sessions are being run in an hour, number of queries which are getting executed in that hour will be less.

To conclude, we have used real data/queries which have been grouped into workloads. These workloads were assigned priority by us and all the workloads were ran in parallel. The workloads were ran by trying to simulate how it would have ran in a customer environment (heavy DSS type queries during night time, light queries during day etc.).

6. Results and Discussions

The benchmark queries (mentioned in Section 5) were ran in parallel in a test system running Teradata database v16.20. The queries were logged and corresponding resource consumption numbers were stored in default Teradata logging tables from which we extracted the data for our experiments. The customer dataset queries were executed at customer environment and we extracted the query features from the query logging tables from customer environment. No PII (Personally Identifiable Information) is present in customer dataset. For our current experiment, hourly data were extracted and used to predict workload parameters for future hour(s).

We have used approximately one month of data for our experiments and hourly data (about 700 hours) was extracted. We have built individual models for each workload by training it over 80% of available data. 10% data was used for validation while 10% was reserved for testing.

The neural network was trained on a machine having Intel i7 processor with 16GB RAM. No GPU was used for training. We have observed that no workload model took more than 15 minutes for training. The entire training on 5 synthetic workloads was completed within an hour. This means that retraining of model should not be a time consuming process and can be done once every month to improve the prediction performance.

We have used RMSE as the evaluation metric for our experiments. RMSE was calculated on normalized data.

6.1. Prediction on prediction

Our focus in this work is not just to predict the future parameters for next one hour, but for future 'n' hours. In the current work, we will be predicting workload parameters for 9 future hours while using 10 previous observations as inputs. While we focus on only 3 workload parameters (CPU, IO and arrival rate (ARM)), if we wish to do prediction for second hour (t_{i+2}), we do need to have predicted values for all input parameters.

We will use these predicted values along with 9 other actual values (observations) from history to perform prediction-on-prediction (rollover prediction). What this means is that if the current time is t_i and if you wish to predict the parameters 3 hours from now(t_{i+3}) using 10 previous history points, you should provide the following as input:

1. Predicted outputs for two hours (t_{i+1}, t_{i+2}).
2. Real values for 8 hours (t_{i-7}...t_i). (Since we need 10 previous points to predict for 11th point.)

Table 3 shows the consolidated results for all workloads. Average RMSE denotes the average of all RMSE values across all predicted parameters. The distribution of errors across 9 future predictions for each of the predicted parameters is shown in Figure 9. Figure 9a shows the result for all 5 synthetic workloads and Figure 9b shows the RMSE values for all 3 customer workloads. All errors are calculated on original data and not on normalized data.

Table 3: Consolidated results for all workloads which we have used for testing. Average RMSE denotes the average of all RMSE values across all predicted parameters. Results are shown for synthetic and customer datasets.

Tier	WDID	Average RMSE Synthetic workload	Average RMSE Customer workload
SLG 3	5	0.1371	N.A.
SLG 3	4	0.1057	N.A.
SLG 2	3	0.1435	0.16
SLG 1	2	0.1078	0.12
Tactical	1	0.1039	0.14

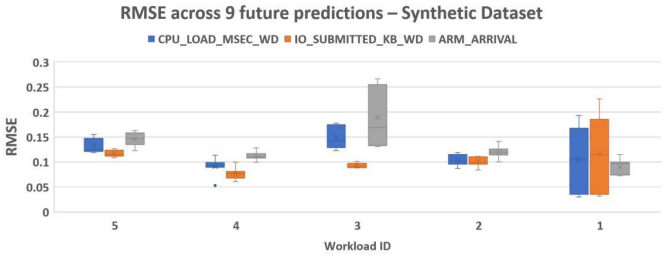
One quick observation from these results is that Arrival Rate (ARM) prediction is commonly having worse prediction across all workloads as compared to CPU or IO. One reason could be that while prediction of system parameters like CPU consumption will have more help by looking at other system parameters (eg: No. of threads), query arrival rate is sorely dependent on how the arrival rate was in history. It is more dependent on day of week and time of day.

We have also observed cases when certain query priority was changed during query execu-tion.e.g: queries started with lower priority but during their execution, priority was increased. However, the number of such queries were limited and we have observed that this dynamic re-prioritizing did not affect the overall workload pattern and hence had minimal impact on our predictions. This is because the ratio of resource consumed by the individual query to total resources consumed by all queries in a workload is very low. Also, please note that we are calculating the resource consumption for a workload in a given hour, so the impact of change in priority of a single (or few) queries will not be significant.

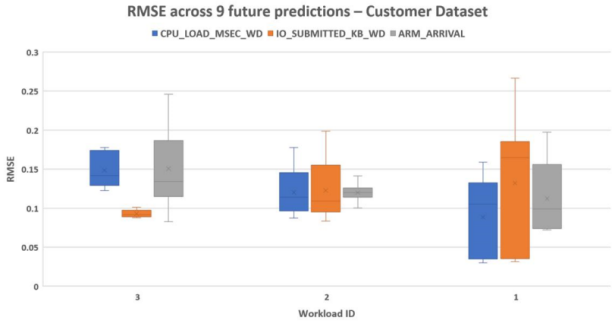
In figures 12a - 12c, we show how the actual and predicted values are for next 3 hours for workload ID 2 of customer dataset. We have shown results only for a single workload due to space constraints and other workloads show results similar to what have been shown here. X axis denote time in hours and Y axis denote the metric values. CPU load is shown in percentages, IO in amount of KB and query arrival rate (ARM) in count. The output values are transformed back to original scale and those are shown in these figures. We have shown the results for about 60 hours in the figure. Figure 12a shows what will be predicted values for these 60 points for next hour. Similarly, 12b and 12c shown the predicted values for these 60 points after two hours and after 3 hours. As can be seen, the actual and predicted values are having significant overlap. Even in cases where there is not much overlap, it can be seen that the general trend of the prediction maps well with the actual values. The ability to predict general trend with some minor errors in predicting the exact values are good enough for use cases like auto fold/unfolding of system resources in a cloud environment. If a user can know in advance that CPU consumption will increase for next two days, then they can plan accordingly and allocate more CPU resources for that specific workload.

Our method can be easily tweaked to predict other parameters like CPU Wait, worker threads etc. CPU Wait can be useful in cases when CPU consumption is constant but CPU Wait has gone up which might indicate resource starvation for that specific workload.

For our experiments, we show our results on 9 future predictions while taking 10 history points for training. Figure 10a and Figure 10b show RMSE values for 9 future predictions for workload ID 2 and 3. As can be seen, RMSE values continues to increase for future predictions as those predictions are using another prediction as input (prediction on prediction), and hence the increase in RMSE. However, we are still having manageable RMSE values with peak RMSE of around 0.14 for WDID 2 and 0.26 for WDID 3.

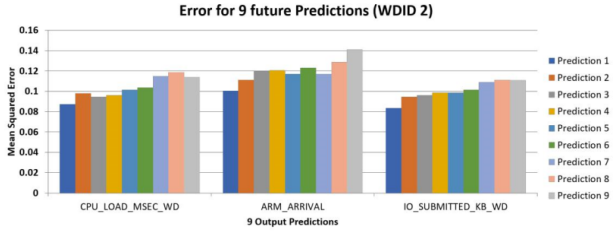


(a) RMSE for CPU, arrival rate (ARM) and IO for all synthetic dataset workloads.

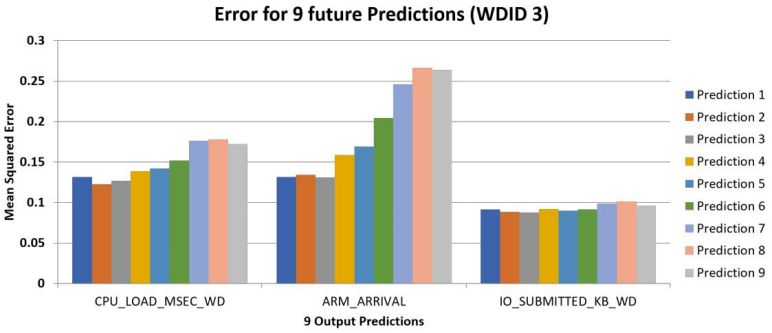


(b) RMSE for CPU, arrival rate (ARM) and IO for all customer dataset workloads.

Fig 9: Figure showing distribution of RMSE values for all workloads belonging to customer and synthetic datasets. For each workload, we have shown the RMSE distribution for each of our predicted values (CPU, IO and Arrival Rate) across 9 future predictions.

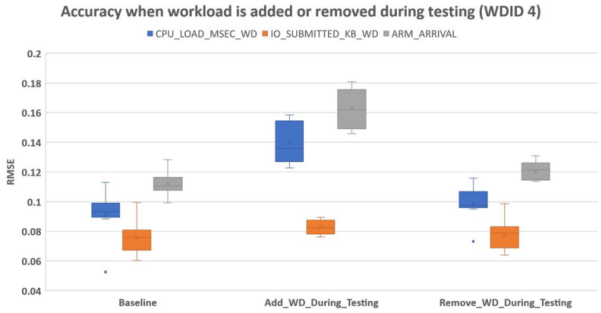


(a) RMSE for CPU, arrival rate (ARM) and IO for WDID 2 for 9 future hours.

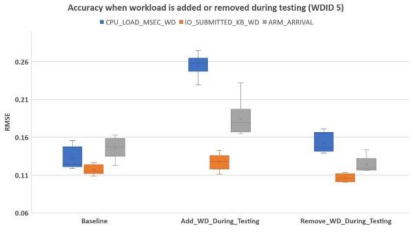


(b) RMSE for CPU, arrival rate (ARM) and IO for WDID 3 for 9 future hours.

Fig 10: Figure showing RMSE value for CPU, IO and Query Arrival Rate Metric (ARM) for workloads 2 and 3 for 9 future hours for customer dataset. Y axis shows the RMSE values for prediction of each metric.

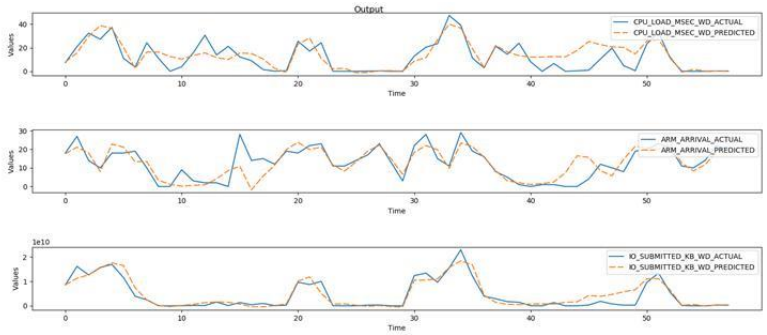


(a) Distribution of CPU, IO and Arrival Rate errors for workload 4 across 9 future predictions.

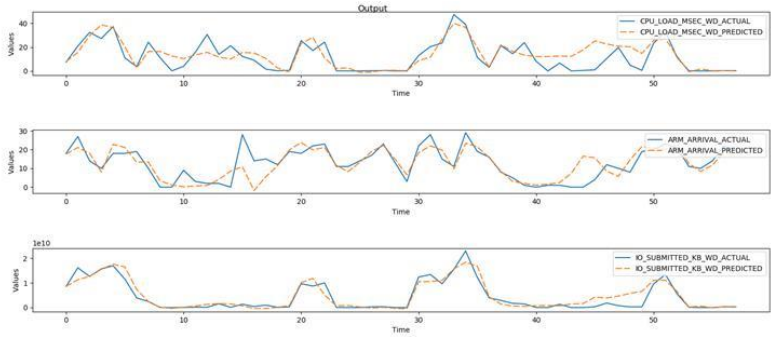


(b) Distribution of CPU, IO and Arrival Rate errors for workload 5 across 9 future predictions.

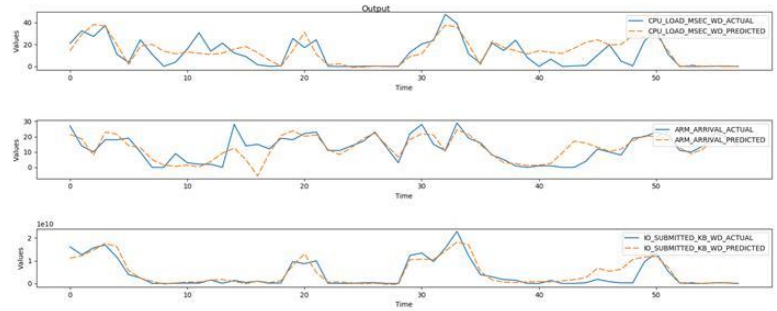
Fig 11: Figure showing effects of adding/removing a workload during testing. We show what was the baseline workload accuracy, workload accuracy when we introduce a workload during testing (which was not present while training) and workload accuracy when an existing workload was removed during testing. No significant drop in accuracy is observed for both workloads. Y axis shows the RMSE values for prediction of each metric. This result is shown on synthetic dataset as we have no control over customer workload configurations



(a) Actual and Predicted value of CPU, IO and query arrival rate for immediate future hour (i.e: First hour)



(b) Actual and Predicted value of CPU, IO and query arrival rate for second hour



(c) Actual and Predicted value of CPU, IO and query arrival rate for third hour

Fig 12: We show actual and predicted values of CPU, IO and query Arrival Rate Metric (ARM) for three future hours for workload 2 in customer dataset. X axis denotes time (in hours) and Y axis denote metric values.

6.2. Addition/Deletion of workloads

In above experiments, we have considered a system in which all workloads are active workloads. The prediction model for a workload was built by taking into account the resource consumed by the workload under consideration and resource consumed by other active workloads. However, new workloads could be added by DBA and existing workloads could be disabled/removed. Since we group the features into bins (with each bin representing multiple workloads), we feel the impact of addition/deletion of workloads should be minimal. To test our claim, we did two experiments in which we added and removed a workload to see its impact on an existing workload. This specific experiment was performed on synthetic dataset as we do not have control over customer workload configurations.

For first experiment, we took 4 workloads for training (workload ID 1, 2, 4 and 5). Workloads 4 and 5 both belong to the lowest tier and share resources available for lowest tier. The prediction model for workload 4 and 5 have been trained in this environment. During testing time, we introduce a new workload (workload 3) in the lowest tier so that now workloads 3, 4 and 5 have to share the resources allocated for the tier.

In the second experiment, all the 5 available workloads are available during training but while testing one workload (workload 3) is removed. Workloads 3, 4 and 5 belonged to same tier during training. Since during testing workload 3 was removed, workloads 4 and 5 now have more resources available for consumption.

Figure 11 shows the results when we add/remove workloads. Figure 11a shows the RMSE distribution for CPU, IO and arrival rate for workload 4 and Figure 11b shows the results for workload 5. In the figures, we show what was the baseline results for the workload, what happened when a new workload which was not present during training was introduced during testing, and what happened when an active workload was removed during testing. Baseline result refers to the case when all 5 workloads were available for training as well as testing. As can be seen from the results, even though the accuracy has reduced, the drop is still within allowed limits for the prediction model to function satisfactorily. Hence, no immediate disruptions will occur due to addition/deletion of workloads. Once enough data have been collected for the modified environment, new training can be triggered for this environment. Till that time, the existing model can perform with acceptable accuracy.

7. Conclusions

A novel method in which we provide prediction of workload parameters like CPU consumed, IO consumed and query arrival rate for a multi-workload environment was introduced in this paper. We have described our detailed approach by which we identify potential features and group them into bins by which we can capture the impact of high priority workload having on lower priority workloads. We then reduced the number of features to a more manageable number using PCA. We also have a mechanism by which users can provide multiple neural network configuration and we can pick the best model among these configurations for deployment. We have used a customer dataset having 3 workloads for our experiments. We have also synthetically generated a multi-workload dataset to simulate different customer environments as well as to run other experiments. The validity of our approach was proved using experimentation and results were provided. We have shown that there is minimal impact even if we add or remove a workload.

The predicted parameters can be consumed by users for different tasks like auto scale up/scale down, scheduling workloads, allocating appropriate resources for workload in future etc. We also have a continuous monitoring system by which users can be informed if the prediction accuracy drops below a certain threshold.

In future, we would like to have a control system which will consume this prediction and can automatically allocate resources to workloads depending upon the future load of the workload. The control system can also provide recommendation to users if they should scale up/scale down and if so, the duration of scale up/scale down as well.

References

[1] G. E. P. Box and G. Jenkins. Time Series Analysis, Forecasting and Control. Holden-Day, Inc., San Francisco, CA, USA, 1990.

[2] S. Chaudhuri and V. Narasayya. Self-tuning database systems: A decade of progress. In Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, pages 3–14. VLDB Endowment, 2007.

[3] S. Chaudhuri and G. Weikum. Rethinking database system architecture: Towards a self-tuning risc-style database system. In VLDB, 2000.

[4] S. Das, , V. Narasayya, and A. C. König. Automated demand-driven resource scaling in relational database-as-a-service. In ACM International Conference on Management of Data (SIGMOD) 2016. ACM - Association for Computing Machinery, June 2016.

[5] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood. Automatic performance diagnosis and tuning in oracle. In CIDR 2005, Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings, pages 84–94, 2005.

[6] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In 2010 International Conference on Network and Service Management, pages 9–16, Oct 2010.

[7] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int. J. Uncertain. Fuzziness Knowl.-Based Syst., 6(2):107–116, Apr. 1998.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Comput., 9(8):1735–1780, Nov. 1997.

[9] I. Jolliffe. Principal Component Analysis. Springer Verlag, 1986.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.

[11] T. Kraska, M. Alizadeh, A. Beutel, E. H. Chi, A. Kristo, G. Leclerc, S. Madden, H. Mao, and V. Nathan. Sagedb: A learned database system. In CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings, 2019.

[12] S. Kumar. Oracle database 10g: The self-managing database. In White Paper, 2003.

[13] 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 Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, pages 631–645, New York, NY, USA, 2018. ACM.

[14] D. P. Mandic and J. Chambers. Recurrent Neural Networks for Prediction: Learning Algorithms,Architectures and Stability. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[15] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and resource modeling in highly-concurrent oltp workloads. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 301–312, New York, NY, USA, 2013. ACM.

[16] R. O. Nambiar and M. Poess. The making of tpc-ds. In Proceedings of the 32Nd International Conference on Very Large Data Bases, VLDB '06, pages 1049–1058. VLDB Endowment, 2006.

[17] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-driving database management systems. In CIDR 2017, Conference on Innovative Data Systems Research, 2017.

[18] J. R. Pierce and E. C. Posner. Autocorrelation and Stationarity, pages 129–140. Springer US, Boston, MA, 1980.

[19] D. Wiese, G. Rabinovitch, M. Reichert, and S. Arenswald. Autonomic tuning expert: a framework for best-practice oriented autonomic database tuning. In Proceedings of the 2008 conference of the Centre for Advanced Studies on Collaborative Research, October 27-30, 2008, Richmond Hill, Ontario, Canada, page 3, 2008.

[20] D. Y. Yoon, B. Mozafari, and D. P. Brown. Dbseer: Pain-free database administration through workload intelligence. Proc. VLDB Endow., 8(12):2036–2039, Aug. 2015.