Applied Soft Computing Journal 91 (2020) 106216

Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

Autonomic performance prediction framework for data warehouse queries using lazy learning approach

Basit Raza [a,*], Adeel Aslam [b], Asma Sher [a], Ahmad Kamran Malik [a], Muhammad Faheem [c]

[a] Department of Computer Science, COMSATS University Islamabad (CUI), Islamabad, 45550, Pakistan
[b] School of Computer Science and Technology, Huazhong University of Science and Technology Wuhan, 430074, China
[c] Department of Computer Engineering, Abdullah Gul University, Kayseri, 38039, Turkey

ARTICLE INFO

Article history:
Received 6 April 2018
Received in revised form 2 March 2020
Accepted 4 March 2020
Available online 6 March 2020

Keywords:
Data warehouse
Autonomic computing
Decision support system
Lazy learning
Case-based reasoning

ABSTRACT

Information is one of the most important assets of an organization. In recent years, the volume of data stored in organizations, varying user requirements, time constraints, and query management complexities have grown exponentially. Due to these problems, the performance modeling of queries in data warehouses (DWs) has assumed a key role in organizations. DWs make relevant information available to decision-makers; however, DW administration is becoming increasingly difficult and time-consuming. DW administrators spend too much time managing queries, which also affects data warehouse performance. To enhance the performance of overloaded data warehouses with varying queries, a prediction-based framework is required that forecasts the behavior of query performance metrics in a DW. In this study, we propose a cluster-based autonomic performance prediction framework using a case-based reasoning approach that determines the performance metrics of the data warehouse in advance by incorporating autonomic computing characteristics. This prediction is helpful for query monitoring and management. For evaluation, we used metrics for precision, recall, accuracy, and relative error rate. The proposed approach is also compared with existing lazy learning techniques. We used the standard TPC-H dataset. Experiments show that our proposed approach produce better results compared to existing techniques.

1. Introduction

A data warehouse (DW) holds historical and metadata information of organizations. It acts as a central repository where integrated information can be queried or analyzed. The main use of a DW is in decision support systems that help top management in their decision making by analyzing DW data. Performance tuning of DWs is required for efficiently analyzing significant amounts of data. Performance tuning requires models, such as multi-dimensional models, that assist in the performance enhancement of queries by forecasting users' intentions. State-of-the-art approaches related to the modeling and designing of DWs are provided in [1]. Query management in a DW is the process of initiating, monitoring, controlling, and executing different requests for DW servers to make optimal use of system resources [2]. Different kinds of queries exist in a single DW server at the same time. There are mainly two types of queries, online transaction processing (OLTP) and decision support system (DSS). OLTP type queries are simple, short, and require a limited amount of resources, such as milliseconds of CPU time and amount of I/O. On the other hand, DSS type queries are complex, long, and require extensive resources, sometimes running for hours or even longer. Study [3] describes the BI Batch Manager, which executes a batch of queries and manages query admission, scheduling, and execution. The management of complex DSS queries becomes crucial for efficient processing. Query management is concerned with controlling aspects of the query, such as how long a request has to wait for system resources. That, in turn, depends upon different query characteristics, which include cost, resource demand, priorities of tasks, and different performance measures, whose identification can enhance the performance of database query planning and execution. Prediction-based techniques attempt to predict the performance characteristics of a query before query execution [4]. Decision tree and correlation-based approaches have been proposed to predict query performance parameters, such as execution time, throughput, elapsed time, and disk I/O.

For improving the DW systems performance, existing strategies mainly tried to reduce the hardware devices, query execution cost, consumption time, memory, human intervention, etc. Computing systems have a very complicated architecture and have reached a high level of complexity, reducing the human effort required to get a task done. Autonomic computing (AC) is a technology that relates to the self-management of computer systems.

* Corresponding author.
E-mail address: basit.raza@comsats.edu.pk (B. Raza).

https://doi.org/10.1016/j.asoc.2020.106216
1568-4946/

List of abbreviations

AC	autonomic computing
DW	data warehouse
OLTP	online transaction processing
DSS	decision support system
K*	KStar classifier
LWL	locally weighted learning
SQL	Structured Query Language
DBMS	database management systems
MAPE-K	monitor, analyze, plan, execute - knowledge
CBR	case-based reasoning
CB	case-base
XML	Extensible Markup Language
JDOM	Java-based document object model
PCA	principal component analysis
CCB	cluster case base
SMOTE	synthetic minority oversampling technique
ED	Euclidian distance
MAE	mean absolute error
SOM	self-organizing map
FCM	fuzzy c-means

AC encompasses several characteristics, such as self-monitoring, self-protecting, self-predicting, self-configuring, self-healing, and self-adapting and reduces the human effort to a minimum. The objective of AC is to make machines and software applications self-managing so that any random behavior or situation can be managed easily with limited interference from knowledge workers. AC techniques can be very helpful for query performance prediction, which is one of the methods used for increasing the performance of the DW that offers numerous benefits [5–7]. The self-predicting system can be used to monitor all activities and provide accurate predictions when the query changes. The self-predicting system predicts based on prior knowledge and mathematical models. However, with the increase of DW size the system takes more query response time depending on the type of query. To overcome these issues, we propose an autonomic performance prediction framework for DW queries. We propose a cluster-based case-based-reasoning (CBR) approach, which is categorized as lazy learning. It could be an appropriate solution for query performance problems, as the prior knowledge concerning queries is stored in the main repository called the case-base (CB). As the behavior of a query is non-deterministic, the lazy learning CBR approach could be a better solution to the query performance prediction problem because CBR inherently contains adaptability features.

In this study, we used the CBR approach, which consists of four phases, namely, *retrieve*, *reuse*, *revise*, and *retain*. Upcoming queries can be categorized into three types of similarity: they may be exactly similar, slightly different, or totally different from the queries stored in the repository. In the *retrieve* phase, data required by the query is collected. If an unseen query is more similar to one of the queries stored in the CB, the *reuse* phase is executed. Otherwise, if the unseen query is less similar, the *revise* phase is executed. After the *revise* phase, the *retain* phase is executed, in which the unseen case is stored in the CB for future use. We used CBR for retaining past experiences; however, it increased the complexity of the retrieval time of cases in the CB. To overcome this problem, a clustering technique is used to predict the performance of features quickly using a minimum retrieval time. To manage the retrieval time of a query, an unsupervised technique, *k-means* clustering, is used to reduce the search time in the particular cluster. The upcoming query will be matched with cases in the cluster, which are selected based on their distance from the centroid points. This will reduce retrieval time as the complexity of searching the query is limited to a cluster. We are not only limited to predicting the values of output features, but we also achieve improved retrieval times. The values of output features are predicted on the basis of similarity between the upcoming case and stored cases. To measure the similarity, the proposed system first selects the *retrieve* function of the CBR; then, it selects one of the cases of CBR, either *reuse*, *revise,* or *retain*. For the new query, one of the optimum functions is chosen to predict the solution for the query. We considered different lazy learning techniques such as instance-based learning with parameter K (IBK), instance-based classifier KStar (K*), and locally weighted learning (LWL). After applying the CBR, the false-negative rate decreases as compared to the other lazy learning techniques. A number of experiments were performed for the proposed framework, and the results show that accuracy was improved through the proposed CBR approach. The proposed system can efficiently predict the execution of a variety of queries. Enterprises can deploy the proposed system as it will help them not only in predicting the performance features but also in managing the queries according to their desired requirements. The key contributions of our study are as follows. Prediction of a query performance metric that consists of seven features (*cached plan size, compile time, CPU time, compile memory, estimated pages cached, size of nodes,* and *execution time)* that are extracted by applying feature selection techniques. The lazy learning approach CBR is used in DWs by retaining the past behavior of queries for extrapolating the solution of new queries. The clusters using CBR are introduced to minimize the retrieval time of cases from the large CB.

The objective of the study is to develop a cluster-based autonomic performance prediction framework for DW. To make our proposed framework autonomic, we incorporated three AC characteristics, which are *self-inspection* for monitoring incoming queries, *self-prediction* for forecasting the performance of DW queries, and *self-adaptation* for managing the evolving behavior of a query by retaining the query changes.

The remainder of the paper is organized as follows. Section 2 presents the related work on AC in DWs. In Section 3, the proposed cluster-based autonomic query performance prediction framework is presented. Section 4 provides experimental details. Section 5 presents results and discussion. Section 6 concludes the study and presents future work.

2. Related work

In the literature, many studies have been conducted to enhance the performance of DW systems. A fuzzy indexing technique for flexible query performance in relational database management systems (DBMS) was proposed in [8] by implementing six different indexing techniques with two different datasets for evaluation. A framework for workload adaptation in DBMS using the principle performance model to forecast workload performance prediction was proposed in [9]. The input for performance models consists of different workload parameters, such as resources and arrival time, whereas output consists of system performance and resource utilization. However, the study experimented with a stable workload, whereas, the dynamic nature of workloads with changing behavior of the queries at runtime has not been examined. Study [10] claimed that map-reduce programming is low-level and required too much effort to reuse

code, and it provided a new model for Structured Query Language (SQL) queries, named HiveQL, which contains schemas and statistics that are beneficial for query optimization. However, HiveQL is based on naive rule-based approach, having a small number of simple rules. For DW scalability, an aggregate cache technique was proposed for efficient execution and maximum throughput of costly aggregated queries of multiple parallel users by implementing different query compensation techniques [11]. The importance of an autonomic manager using a knowledge-base for the business intelligence system was discussed by providing a scheme that formalizes different aspects in the form of ontologies and saves them in the knowledge-base [12]. It analyzed query performance and proposed a new cache allocation technique for better performance. A new architecture *HyPer,* which supports both OLTP transaction and OLAP queries concurrently, was proposed [13]. This technique is helpful in achieving efficient query performance and response time. Another study, [14], presents a modular approach to forecasting query execution times for large data volumes. An analytical model is proposed for predicting the execution times of DB access, I/O access, and different SQL operators. The proposed model was validated with three applications, and the TPC-H benchmark and prediction error of this model was within 10%. A cost model to forecast the execution times of SQL queries using the query execution plans was presented in [15]. This approach is limited to only one database, Postgres SQL. Another study, [16], describes a systematic approach that predicts the disk visits and CPU cost of extensive data. In [17], the consumption of IO access to quantify the query idleness in simultaneous workloads is presented. The validation of this model was done on the TPC-DS benchmark, and prediction accuracy was calculated through mean relative error. A prediction model was used to predict the execution times of complicated queries using benchmark datasets, namely TPC-W, RUBBoS, and RUBiS [18], and the behavior of data was used for the construction of a prediction model for a large number of clusters and data using the Amazon EC2 dataset [19].

The COMPASS framework was proposed in [20], which generates a performance model and automatically identifies parameters and predicts the CPU performance for nine real applications. The flexible and precise prediction provides several benefits, for example, identifying important parameters and understanding the tradeoff between design and performance. A neural system was used to predict the response times of subsequent queries, and the association between the throughput of the system and the response time of query in the OLTP system was analyzed. Three strategies were used to evaluate the response time of queries using TPC-DS benchmarks, such as multilayer perceptron using a backpropagation training algorithm with momentum method, multilayer perceptron using artificial bee colony training algorithm, and multilayer perceptron modified as a small-world network [21]. The results showed that MLP based SWN and BP performed better on longer queries, while ABC performed better on shorter queries. Study [22] presents a framework, DBSeer, for managing resource usage during highly concurrent workloads to improve the system performance of OLTP transactions. Statistical models were applied to real-world datasets for experiments. The NN**M**onitor, **A**nalyzer, NN**G**enerator (MAG), a layered framework, was used in [23], which demonstrated a method to monitor, analyze, predict, and control database parameters. Two metrics were used to measure the prediction accuracy: root mean square error and average error. A framework that predicted the performance of a workload by building a model using the TPC-C dataset was presented to measure several performance metrics like resource bottlenecks, throughput, and resource consumption [24]. A model that used the execution plan of a query and machine learning to predict the execution time was proposed [25]. The prediction model used historical data and built a prediction of query runtime tree. It generated the time ranges, and the child nodes used the ranges of the parent nodes. The limitation of this model is that it considered only one time span . The model did not cater to sudden changes. A framework was proposed for the prediction of performance metrics [26]. Machine-learning-based approaches were used to predict the performance of different attributes, such as records used, message bytes, elapsed time of the query, and disk input/output. The proposed approach used information prior to query execution and provided correct results. It predicted the elapsed time and provided information about short and long queries. Like machine learning, there exist many lazy learning approaches in the literature. An extensive survey on time-series algorithms that are used in different domains, such as sleep identification and recognition of human motion, was presented in the study [27]. The authors used 12 multivariate time-series datasets and presented an extensive evaluation of their experiments. Food sales prediction using time-series forecasting was presented in [28]. As the business environment constantly changes, there is a need for the timely and accurate prediction of food stocks for future sales. The CBR approach has been used in various domains. It has worked well for various businesses and engineering research problems. An architecture named SEA SALT was proposed [29]. It works with diverse data repositories. It is based on the CBR approach for maintaining, retrieving and adapting, and retaining cases. A CBR-based framework for automatically repairing extraction, transformation, and loading workflow processes in DW that are required owing to the changes in the database schema was presented in [30]. A clustering-based approach to predict the labels of traffic patterns automatically was proposed in [31]. The authors used point-based and area-based approaches to extract patterns from a large dataset.

In this study, we propose an autonomic performance prediction framework for DW queries for performance tuning. The proposed framework provides a better solution for performance modeling in DW through lazy learning approaches. Performance modeling in DW has been performed through various approaches, such as indexing, cache management, materialized views, ontology-based, business intelligence, AC, hybrid approaches, and map-reduce. The proposed framework is novel in that it was developed specifically for the DW environment. On the contrary, the previously published models and frameworks [32,33] are used for the DBMS environment. They perform query characterization into the OLTP and DSS types, and the performance prediction of the incoming queries is made before execution. The study [1] presents state-of-the-art work in large-scale data repositories and provides future research potential. This study is entirely related to the DW domain and uses different tools and technologies that will be discussed in detail. Based on the literature, we have proposed a cluster-based CBR framework for the prediction and adaption of DW queries to enhance performance and optimize efficiency. Above all, the proposed framework provides a better solution for performance modeling in DW through lazy learning approaches.

3. Proposed autonomic performance prediction framework

This section presents the proposed autonomic performance prediction framework for DW queries. Section 3.1 provides a feature selection of DW queries. Section 3.2 provides query performance prediction using CBR with its four phases and presents a cluster-based CBR algorithm.

The most important consideration for managing a query's complexity is prediction and adaptation of the query. Any type of query can enter the DW system, which may result in long execution times. Queries that involve the *select* statement are

Table 1
Feature selection techniques.

	Feature selection techniques						
	PCA	CFS subset	Correlation	Gain ratio	Info gain	Relief-F	Symmetrical uncertainty
Performance metrics	Compile memory	Compile time	Execution time	Execution time	Execution time	Execution time	Execution time
	Cache Plan Size	Non-parallel plan reason	Size of nodes	Id of input parameter	Compile memory	Id of input parameter	CPU time
	Estimated pages cached	Cached plan size	Compile memory	Cache Plan Size	CPU time	Size of nodes	Compile memory
	CPU time	Compile time	CPU time	CPU time	Compile time	Compile time	Compile time
	Compile time	CPU time	Compile time	Compile memory	Size of nodes	Compile memory	Id of input parameter
	Size of nodes	Compile memory	Estimated pages cached	Compile time	Estimated pages cached	Serial required memory	Size of nodes
	Execution time	Serial required memory	Cache Plan Size	Size of nodes	Id of input parameter	Cache Plan Size	Cache Plan Size

Table 2
Query input features and performance metrics.

Query input feature	Query performance metric
Number of sub-queries (Sq)	*Cached plan size (CPS)*
Number of equality predicates (Eq)	*Compile time (CT)*
Number of selection predicates (Sp)	*CPU time (CPUT)*
Number of non-equality predicates (Neq)	*Compile memory (CM)*
Number of joins (J)	*Estimated pages cached (EPC)*
Number of equijoin predicates (EqJ)	*Size of nodes (SON)*
Number of non-equijoin predicates (NeJ)	*Execution time (ET)*
Number of aggregation columns (Ag)	
Number of sort columns (Sc)	

generally DSS/OLAP queries, whereas the queries consisting of *insert*, *update*, and *delete* statements are OLTP queries. We observed that DSS queries are lengthy and complex; therefore, these kinds of queries consume more time. Unlike DSS queries, the execution time of OLTP queries is quite short. Performance tuning is an important aspect of complex DSS queries. AC is used to reduce the human effort to a minimum and relates to self-management, encompassing several characteristics, such as self-monitoring, self-protecting, self-predicting, self-configuring, self-healing, and self-adaptation. The objective of the AC is to make machine and software applications self-managing so that any random behavior or situation can be managed easily with less interference by knowledge workers. Self-adaptation can assist in the overall performance of a DW system. For predicting the performance of a targeted query, we have proposed a framework that is able to manage the query autonomically and predict its performance before execution. The reason behind the prediction is to attain pre-knowledge about the targeted query. As the long-running queries consume much execution time and create resource contention for the small running queries, the forecasting of the performance metric enables knowledge workers to make decisions regarding the processing of queries. The prediction of output performance is based on the information available in the stored CB. By predicting the performance of the query, the knowledge worker can manage the query, make strategies, and plan the execution process. Query prediction leads to query optimization. Query optimization is of high importance for the enhancement of DW performance, specifically during the execution process of complicated SQL queries. On the basis of the query optimizer, a better strategy can be determined for placing each query in a queue according to the priorities. By doing so, the knowledge workers will be able to handle the resource contention for their own business interests.

This study provides a framework to forecast a query's performance by predicting the values of a query performance metric. Fig. 1 shows the mapping of the standard AC model to the proposed approach. Fig. 1(a) presents the standard monitor, analyze, plan, execute — knowledge (MAPE-K) model that is the basic architecture for an autonomic system [4]. It consists of *managed element, sensor, effector*, and *MAPE-K* elements that are connected through a loop. Fig. 1(b) presents the mapping of the MAPE-K model to our proposed CBR approach, where a DW query acts as a *managed element*. The *sensor* observes changes in the behavior of a query, and the *effector* is used to incorporate those changes into the *knowledge-base*. To make our proposed framework autonomic, we incorporated three autonomic characteristics, which are *self-inspection*, *self-prediction*, and *self-adaptation*.

The proposed autonomic performance prediction framework is presented in Fig. 2. When a new query enters into the system, first, the query is examined, and a number of query input features are extracted. The query input features and query performance metrics are stored in the CB repository in the form of cases that become training and testing data.

3.1. Feature selection of data warehouse query

Large numbers of queries were executed on the Microsoft SQL server; as a result, a number of features were obtained in the form of an *Extensible Markup Language (XML)* file that contains the values of performance metrics. The XML data was parsed with the help of the *Java-based Document Object Model (JDOM)* [34]. However, the generated features are not contributing equally to improving the performance of DWs, so we selected only the 24 features that are mostly used in the literature. We applied several techniques, such as *principal component analysis (PCA),* CFS subset, correlation, gain ratio, info gain, Relief-F, and symmetrical uncertainty, to select the most relevant performance features among the 24 features. Table 1 shows performance metrics of DW queries consisting of the best seven features returned by each technique.

Among the listed dimensionality reduction techniques, PCA is an appropriate choice. The reason is that PCA selects attributes based on 95% of the total variance and permits only 5% information loss. It can be applied to any kind of dataset. PCA is able to reduce redundancy, computational complexity, and noise. Therefore, we have preferred PCA for conducting our research and selected the features returned by the PCA technique. Table 2 shows nine query input features and performance metrics

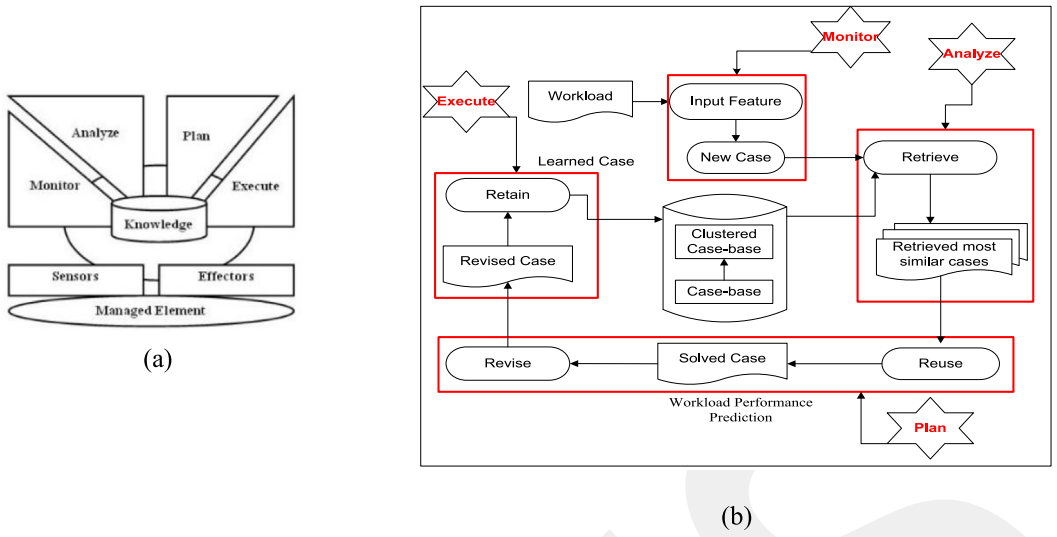

Fig. 1. Mapping standard AC model to proposed CBR approach: (a) MAPE-K Model, (b) Mapping of MAPE-K to CBR approach.

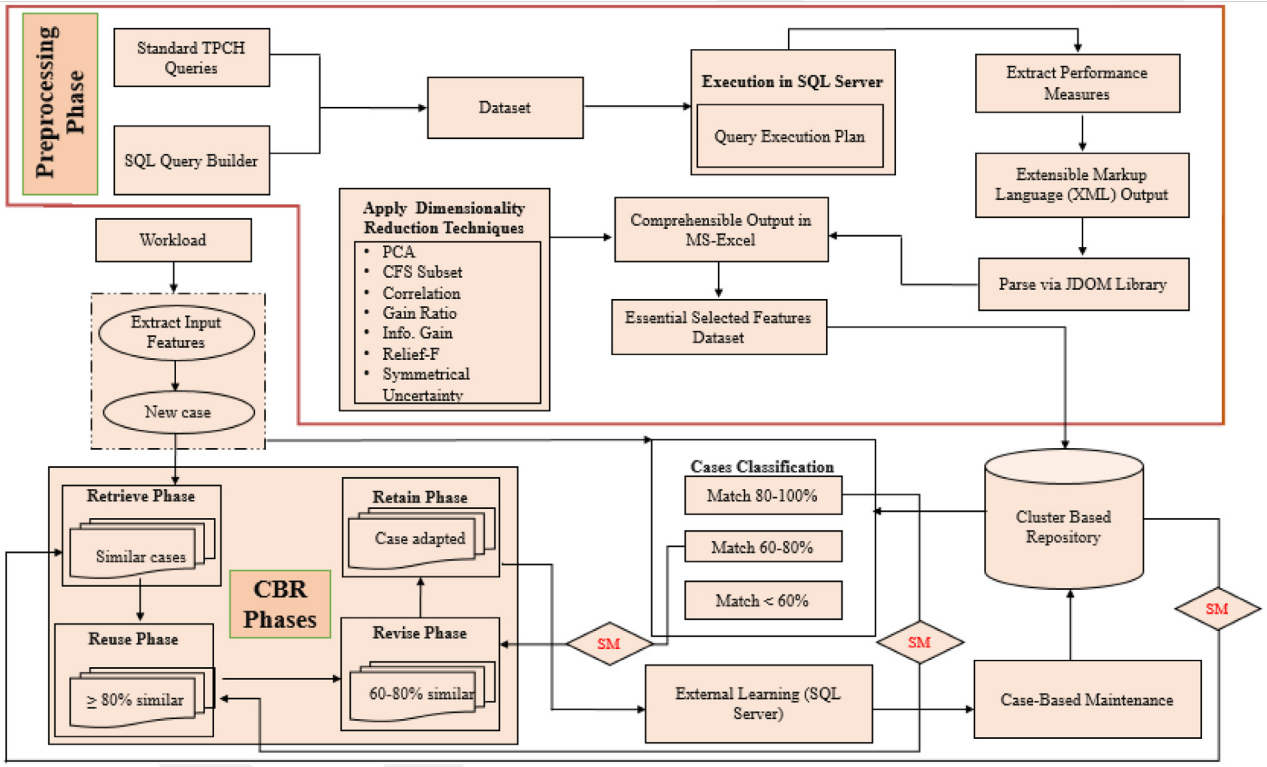

Fig. 2. Proposed autonomic performance prediction framework.

(consisting of seven features) identified by PCA and used for query performance prediction. Table 2 shows nine query input features [35] and performance metrics identified by PCA that are used for query performance prediction in the proposed model.

3.2. Query performance prediction using CBR

After the process of feature selection, the next step is to predict the performance of an incoming query. Many lazy learning techniques such as K*, IBK, and LWL [36] have been investigated in the literature. In this study, we applied the CBR approach along with existing techniques on the TPC-H dataset. The CBR approach solves the problem using past experiences relevant to the type of problem at hand. It solves the problems through reasoning that is based on historical data. It works well in scenarios where information regarding the targeted problem is not well-formalized and where classification learning algorithms fail to perform well. CBR enables learning, even if the background knowledge is incomplete or not accessible. CBR consists of four phases, also known as 4R's of CBR, the *retrieve*, *reuse*, *revise*, and *retain* phases.

3.2.1. Retrieve phase

In this phase, the test query is compared with cases in the CB repository, and the most similar cases are retrieved. We used Euclidean distance (ED), which is widely used by researchers to determine similar cases. This process returns a number of cases

related to the test case. The related significant solution to the test query is returned by the first phase of CBR.

3.2.2. Reuse phase

In the second phase of CBR, the cases returned by the *retrieve* phase are used to devise a solution to the test query. The most similar cases from the retrieved solution (*retrieve*) are obtained. The similarity measure is used to predict the performance of the test query. The *reuse* phase is performed through similarity measure if the test case is highly similar to one of the cases returned by the *Retrieve* phase. In our study, we defined a similarity range of 80%–100% similarity between the query input feature values and the stored cases. The result of the stored case is reused for the test query.

3.2.3. Revise phase

When high similarity is not found in the solution provided by the *retrieve* phase, the *revise* phase is used. The performance of the test query is forecasted. In the *revise* phase, when the test query's solution is not similar to the defined high similarity range, then the solutions in the defined low similarity range are selected to provide the solution for the test query. In our study, for revising the solution, the similarity range between the test query and the stored cases of the retrieved solution is 60%–80%. In CBR, if the solution for the targeted query is 60%–80% similar to the result of the stored relevant cases, then the case is revised in the *revise* phase, and prediction is performed.

3.2.4. Retain phase

The *retain phase* is also known as the adaptation phase. When a new target query is encountered and its match is not found in the solution of the *retrieve* phase, then the *retain* phase is used to adapt the cases that were revised in the *revise* phase. CBR can handle the evolving behavior of a query, and it adapts new cases dynamically. When an appropriate solution for the test query does not exist in the CB repository, the revised case obtained through the *revise* phase is retained in the CB repository as a new case for predicting the performance of queries. In the *retain phase,* the new case is indexed at the end of the CB repository after the existing ones.

3.2.5. Cluster case-base

In this subsection, the cluster CB is presented, that is, the knowledge repository and all the cases that are stored in it. When a query enters into the system for execution, its features are extracted, and the performance is predicted. Cluster CB contains a number of cases. A case consists of nine query input features and seven query performance metrics, as shown in Table 2. The cases in the CB are distributed into training data and testing data. The testing data are compared with the training data through a similarity measure. Improving the efficiency of case retrieval can improve the prediction performance. Traditional CBR is not efficient for prediction as it searches the whole CB for case retrieval, which may take a long time for large repositories. In this study, the cases are stored in matrix form, where each column represents the input (query input features) and output (query performance metrics), and each row represents a single case. The stored cases are searched in the CB repository for performance metric prediction. Matching occurs between the new case (test query) and existing cases in the CB repository. For predicting the performance of a query, a test query is matched in the CB repository. Initially, the size of the repository is manageable; however, the size dynamically increases gradually (due to the adaptive nature of CBR) through the insertion of more new cases in the repository. Due to the heavy size of the CB repository, a DW system takes a long time to search for a case in the repository.

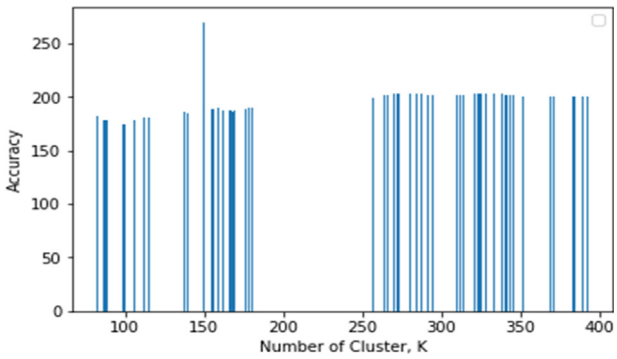

Fig. 3. K-means CBR clustering using Monte Carlo.

Therefore, a strategy is required that can minimize retrieval time. To improve the retrieval time, we have applied an unsupervised clustering technique known as *k-means* clustering on a repository. The advantage of clustering is that it divides the dataset into k clusters, and the search time is reduced to only a single cluster instead of the whole CB. Each cluster comprises data points that are similar to each other and different from the data points in other clusters.

To create the clusters dynamically, we used the *k-means* clustering technique. The selection of k is important in *k-means* clustering, and this study proposes to choose a random k dynamically. We used two approaches, the *Monte Carlo* [35] simulation and the elbow method [37] for finding the optimal value of k and its accuracy. The elbow method is applied for selection of k, and *Monte Carlo* is applied to observe the impact of varying k on the accuracy. Different numbers of clusters with varying values of k are tested through *k-means* and other algorithms, such as the *fuzzy c-means (FCM)* and *self-organizing map (SOM)* clustering algorithms [38,39].

3.2.6. Proposed algorithms for cluster-based CBR approach

Algorithm 1 shows *k-means* CB clustering using the *Monte Carlo* method. It takes as input a CB containing n cases CB, similarity measure SM, solution algorithm SA, clustering algorithm CA, test case base TCB, and number of iterations t. The output of the algorithm is the cluster case base (CCB). It finds the optimal value of k dynamically based on accuracy. First of all, the clustering algorithm (CA) is executed on the CB that divides the cases into a number of clusters depending on the value of k. For each cluster, it finds the similarity between cluster cases c_p and test casesc_j. When the similarity is greater than the threshold (sim > 80), the accuracy is updated, and the optimal k is selected. Results show that increasing the number of clusters beyond a certain level does not increase the accuracy significantly, as shown in Fig. 3.

The output of Algorithm 1 (a CCB that is the optimal value for the number of clusters) is used as input in Algorithm 2, which executes cluster-based CBR.

Algorithm 2 shows the cluster-based CBR approach. It takes IC, DS, and C as input, where IC is the input case having query features (Sq, Sp, Eq, Neq, J, $EqJNeJ$, Ag, Sc); DS is the data store repository of cases, and CCB is the cluster case base. The output of the algorithm is the query performance metric (CPS, CM, CT, $CPUT$, EPC, NON, ET, ED). It calculates the centroid point for each cluster and the ED for each centroid point using Eq. (1). Further, ED is calculated for each case in the selected cluster from the input case. On the basis of ED, the similarity is calculated using Eq. (2). Based on defined similarity ranges, the *retrieve*, *reuse*, and *revise* phases are performed, and the performance metric is predicted.

Algorithm 1. K-means CB clustering using elbow and Monte Carlo methods

Input: *A Case Base containing n cases CB, similarity measure SM, solution algorithm SA, clustering algorithm CA, test case base TCB, number of iterations t.*
Output: *Cluster Case Base CCB*
Procedure:
current Accuracy := 0
for i: = 1 *to* t do // *generate random number for every iteration*
k := i
$accuracy_k$:= 0
CCB := CA(CB, k)
for each $c_j \in TCB$ do // *calculate cluster for every test case base*
$cluster_i$:= classify(c_j, CCB)
for each $c_p \in cluster_i$ *and* $p \neq j$ *do* // *calculate similarity for every cluster*
$sim := findSimilarity(c_j, c_p, SM)$
end for
if $sim > 80$ then
$accuracy_k := updateaccuracy()$
endif
end for
if $accuracy_k > currentAccuracy$ then
$currentAccuracy := accuracy_k$
$k_{opt} := k$
endif
end for
$CCB := CA(CB, k_{opt})$

Algorithm 2. Cluster-based CBR approach

Input: $IC\{Sq, Sp, Eq, Neq, J, EqJ\ NeJ, Ag, Sc\}, DS, CCB$
Output: PM: *Performance Mertic* $\{CPS, CM, CT, CPUT, EPC, NON, ET, ED\}$
Procedure
$sim_{max} := 0; ED_{min} := 500$
For each $c_i \in CCB$ *do* //*calculate centroid point for each cluster*
$Centpt_i \leftarrow centpt(c_i)$ //*calculate E.D for each centroid point*
$ED_i = ED(IC, Centpt_i)$
IF $ED_i < ED_{min}$
$ED_{min} := ED_i$
$c_{selected} = c_i$
END IF
End For
For each $ds_j \in DS$ *do* //*calculate ED between input and each case in selected cluster*
$ED_j = ED(IC, ds_j)$ *j is every element of selected cluster and generate list*
$sim_j = \frac{1}{1 + ED_j} * 100$
IF $sim_j > sim_{max}$
$sim_{max} := sim_j$
$ds_{selected} = ds_j$
END IF
End For
IF $sim_{max} > 80$ *then*
Reuse Case // *PM is reused for prediction*
Else IF $sim_j < 80$ *and* > 60 *then*
Revise Case // *PM is updated for prediction*
Retain Case // *PM is adapted for prediction*
Append IC and $ds_{selected}$ // *Append the new case in data store repository*
End IF

If no match is found, then the *retain* phase is performed, and the new case is appended in the DS. The flowchart in Fig. 4 shows the main steps of Algorithm 1 and Algorithm 2 for the proposed autonomic performance prediction framework.

For computing similarity, we have used Euclidian distance. For all four phases of CBR, the similarity was computed between the test case and the stored cases of the repository. The mathematical formula for ED is given in Eq. (1).

$$ED(IC, centpt) = \sqrt{\sum_{i=1}^{n} (IC_i - centpt_i;)^2} \quad (1)$$

where IC_i is the ith value of the input case, and $centpt_i$ is the ith value of the centroid point of a cluster.

We need to compute the ED of the test query with every cluster. After computing the ED, we selected one cluster, whose ED value is the least one among computed differences, for further processing. After finding the cluster, the next step is to convert the ED into a percentage of similarity using Eq. (2), where *Sim* is the similarity percentage between the input query and the stored case in the CB repository.

$$Sim = \frac{1}{1 + ED} X 100 \quad (2)$$

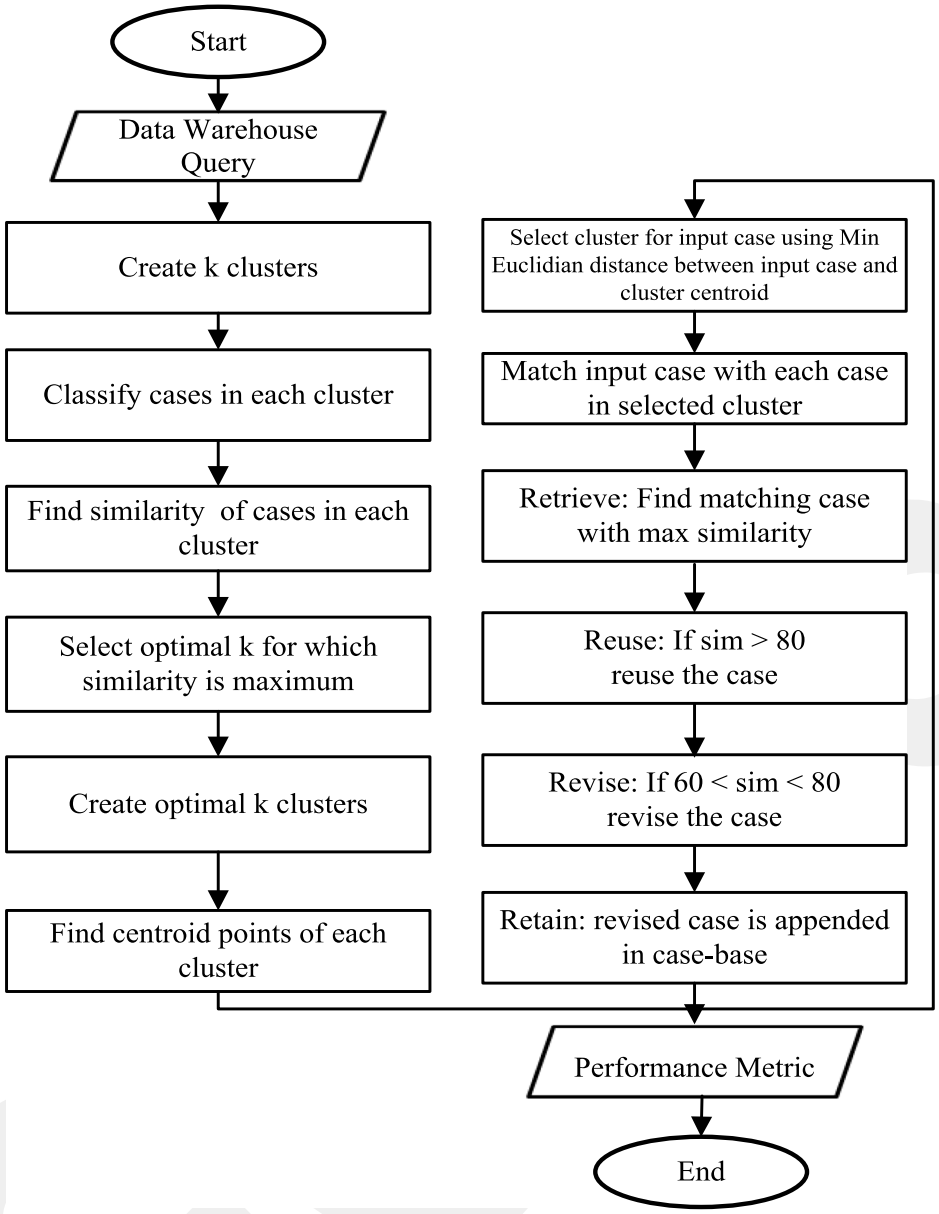

Fig. 4. Flowchart of cluster-based CBR.

4. Experimental setup

This section provides the experimental setup for the proposed framework and the dataset used. To validate the proposed framework, the performance evaluation metric used in the experiments is also presented. Microsoft SQL Server 2017 with the Java Database Connectivity driver for connection with Java was used for extracting information on SQL queries. Standard open-source APIs, such as JDOM and SQL-Parser for parsing large queries, were used in our experiment. The CBR Model was developed using JAVA with the latest Java Development Kit and Eclipse as the integrated development environment. For visualization, we used Python with Matplotlib and Pandas support. Array list and hash map data structures were used for containing variable values dynamically while performing or normalizing data. The Waikato Environment for Knowledge Analysis, a standard library that contains lazy learning algorithms, was used. Microsoft SQL Server was used on a machine having Core i5 CPU, 8GB RAM, and Windows 10.

4.1. Dataset description

For our experiments, we used the TPC-H dataset [40]. TPC-H is a Transaction Processing Council decision support benchmark. It consists of the TPC-H database and a collection of queries relevant to organizations, industries, and businesses. TPC-H assists in decision-making support systems, as it consists of an immense amount of data and complex queries. The *DBGen* tool was used to create the TPC-H benchmark. The standard set of TPC-H queries containing 22 standard queries is available online for research. The estimated database size is two gigabytes. Using the Query Builder tool, we created and executed over 1000 TPC-H-like SQL queries. Available data was augmented against different dimensions using the Synthetic Minority Oversampling Technique (SMOTE) [41]. After the discretization process, where classes are defined in our original dataset, the next step is data augmentation. In this step, we duplicate and place some records against some minority classes of different dimensions. SMOTE is a technique that works on the basis of the nearest neighbor. We used the ED between different points in feature space for

the placement of new records between different original data points. For experimental purposes, the training and testing data was prepared from a dataset that consisted of 70% training and 30% testing data.

4.2. Evaluation metrics

In this study, we used accuracy, precision, recall, and f-measure for comparing different lazy learning algorithms. Classification accuracy was computed to analyze the number of instances that were accurately classified. The mathematical representation of the accuracy measure is shown in Eq. (3).

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{TP + TN + FP + FN}, \tag{3}$$

where TP, TN, FP, and FN represent the number of true positives, true negatives, false positives, and false negatives, respectively. Precision is also called the positive predictive value, which provides the fraction of predicted positives that are actually positive in the dataset. Precision is a computation of quality. The mathematical expression for precision is shown in Eq. (4).

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

Another evaluation measure, named recall, also known as sensitivity, is a measure of completeness. The mathematical expression for the recall is shown in Eq. (5).

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

F-measure is a measure of accuracy based on both precision and recall. It is also known as F-score. The mathematical expression of the F-measure is shown in Eq. (6).

$$F - measure = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})} \tag{6}$$

We also evaluated the accuracy of prediction in terms of the mean absolute error (MAE), as defined in Eq. (7).

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{x_i - x}{x}\right| \tag{7}$$

5. Results and discussion

This section provides the results of the experiments that were carried out on the proposed framework for query performance prediction in DWs. Various queries related to DSS were executed on SQL server, and variables indicating the query's status were recorded. The proposed framework was validated through the evaluation metrics accuracy, error rate, precision, recall, and f-measure. Traditionally, machine-learning techniques have been applied for performance predictions; however, lazy learning approaches have proven to have better performance in predictions due to their learning mechanism based on existing solutions. Several lazy learning techniques, such as CBR, K*, IBK, and LWL, were analyzed, and their performance was compared with the proposed cluster-based CBR approach.

We performed experiments for performance prediction using the proposed CBR approach and computed the accuracy. The accuracy of the selected performance metrics is shown in Fig. 5. The accurate classification rates for selected performance metrics is above 80%, except for ET and EPC. From this result, it can be concluded that the performance metrics selected through PCA are of high significance.

We measured the relative error for the selected features. A lower error rate indicates better prediction performance. Fig. 6 presents the MAEs of our proposed CBR and other lazy learning techniques.

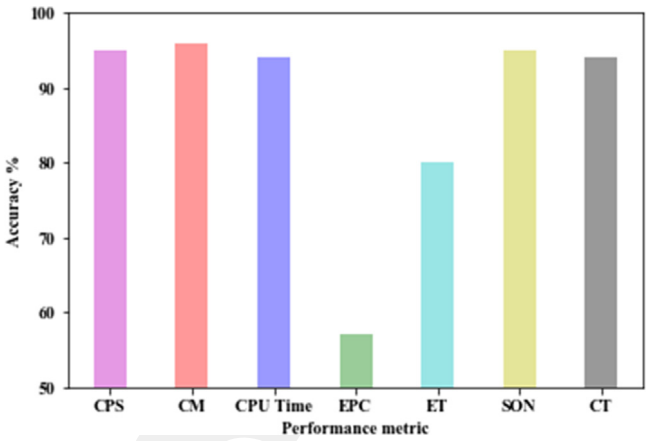

Fig. 5. Accuracy of case-based reasoning.

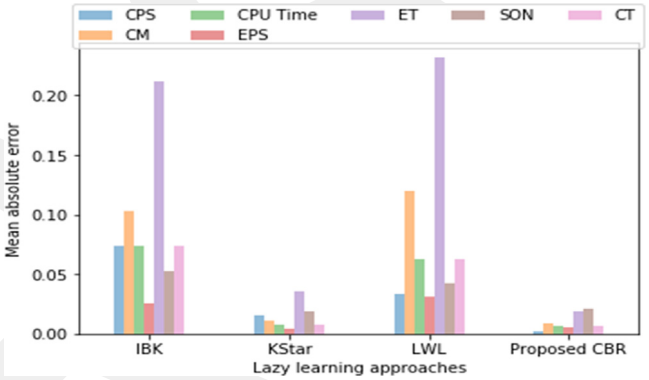

Fig. 6. Relative error of selected features.

In the elbow method, *k-means* clustering was applied to the data for different values of k, and the *sum of squares error* was calculated. With this method, the elbow bend in the graph line represents the best value of k. According to Fig. 7, the elbow bend is between the k values of 2 and 5. So, we used these cluster numbers in our experiments. Moreover, using a large number of clusters increases overhead.

The lazy learning approach has to deal with a large, continually changing dataset for the creation of clusters and finding the distance of input to each cluster centroid, which is time-consuming for predictions, so it is optimal to choose a minimal value of k. A randomized approach to cluster the CB is proposed in the form of Monte Carlo variations. The random process can be repeated a few times to ensure acceptable performance. The relatively best value of k among t iterations (where$t = 9$ iterations) gives a good approximation of the optimal number of clusters, as shown in Algorithm 1. Fig. 7(a) – (d) show the optimal number of clusters using elbow method *k-means* clustering for k>=2 on some of the performance metric features.

5.1. Integrating clustering techniques with CBR model

We have integrated clustering techniques with the proposed CBR model. The experiments carried out on the proposed cluster-based CBR using clustering techniques such as *k-means*, fuzzy c-means clustering, and SOM for query performance prediction in DWs. When searching a case for a test query, a small size cluster needs to be searched rather than the entire repository. Clusters

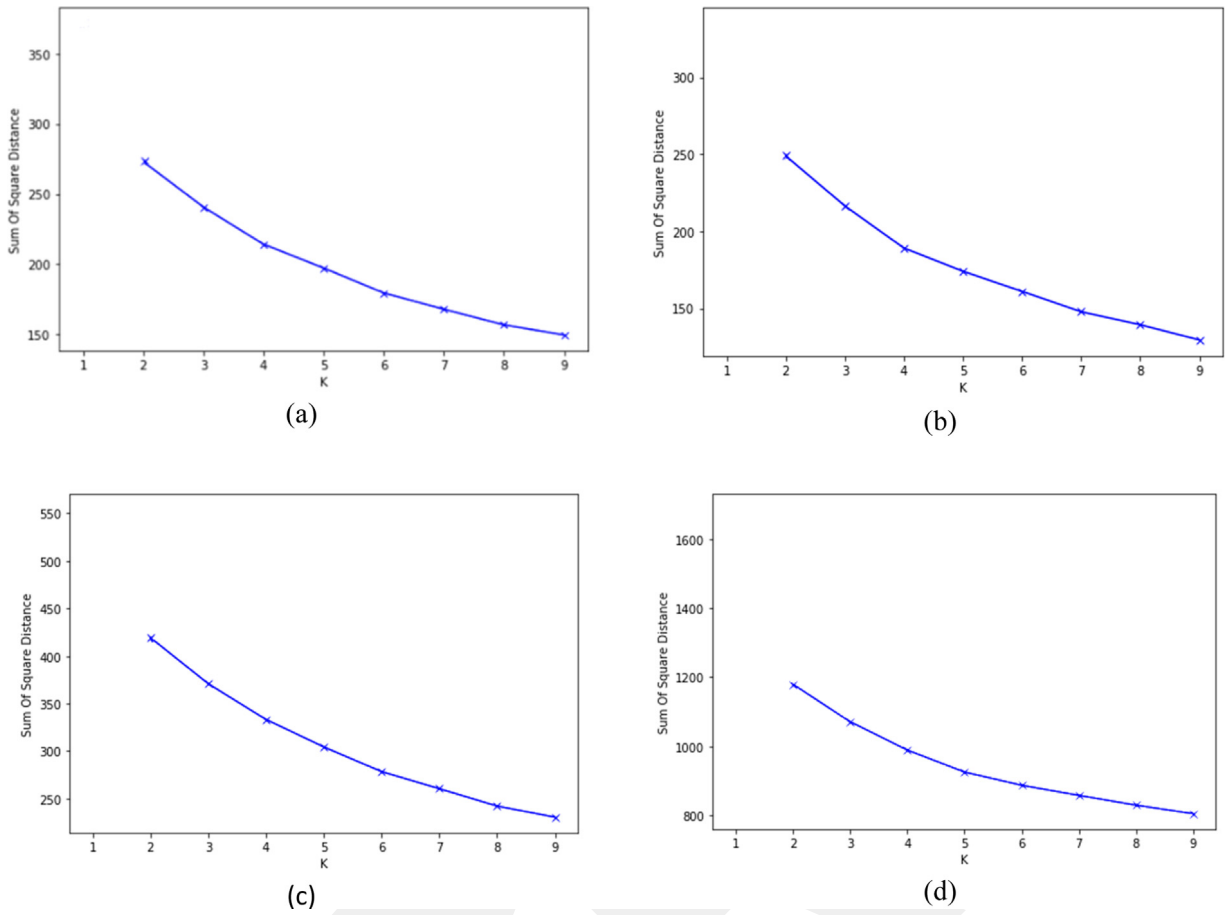

Fig. 7. Optimal number of clusters using elbow method k-means clustering: (a) cached plan size, (b) compile memory, (c) CPU time, and (d) EPC.

are dynamically created, depending upon the requirements of the user. We set the value of k from 2 to 5 to test the clusters. The dataset was divided into a number of clusters. Test data was matched with training data through the similarity of the centroid points of the clusters. Only the matching cluster was loaded into memory, and in this way, main memory requirements can be reduced, resulting in scalability and efficiency. Searching the whole CB against a single input requires much computational time, so we applied a clustering technique that splits the whole CB into a number of clusters, and a relevant cluster is selected depending upon the minimum distance between the cluster centroid and input query. Clustering was applied to split the large CB for efficiency, so we integrated *k-means*, fuzzy c-means, and SOM clustering with our proposed CBR model.

5.1.1. K-means clustering

The K-means algorithm is used to categorize group to data. The number of groups is represented by the variable k. It iteratively assigns a group to each data point depending upon its similarity. The time complexity of the *k-means* algorithm is O(ncdi), where n is the number of data points; c is the number of clusters; d is the dimension, and i is the number of iterations). It gives us the centroid of each group or cluster and a label for each data point belonging to the cluster. Fig. 8 shows the varying numbers of clusters for selected features using *k-means*. In our case, we are predicting performance metrics consisting of 7 features and show *k-means* cluster results with varying values of k. Fig. 8(a) – (d) show that when $k = 2$, it has two dense clusters, whereas, for other values of k such as $k = 3$, it shows sparse clusters.

5.1.2. Fuzzy c-means

Fuzzy c-means (FCM) is a clustering approach where each data point may belong to more than one cluster. It is also an extension of the k-means algorithm. In this type of clustering, we initially assign some degree of membership randomly to each data point. The next step is centroid calculation and calculation of the dissimilarity between object and centroid using ED. The member function is updated using Eq. (8). We used $m = 2$ as the fuzzification observed from the literature. These steps are repeated until the centroid does not change. We integrate FCM with our model and varying values of k. The time complexity of the fuzzy c-means algorithm is $O(nc^2di)$. Fig. 9 shows the varying numbers of clusters for selected features using fuzzy c-means. Results are presented in Fig. 9(a)–(d). The color densities show more clustering of data points for $k = 2$, whereas, for other values of k, such as $k = 3$, it shows sparse clusters.

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}} \tag{8}$$

5.1.3. Self-organizing map

SOM is a type of neural network that is trained using unsupervised learning and is used for dimensionality reduction. First, the node weight vector is randomized in a map, and an input vector is randomly selected. After that, each node in the map is traversed to find the similarity between the input vector and node weight vector using ED. In this way, a node called the best matching unit (BMU) is selected. The weight vectors of the neighbors of the BMU are updated, pulling them closer to the input vector. These steps are repeated until weight convergence. The minimum ED is used

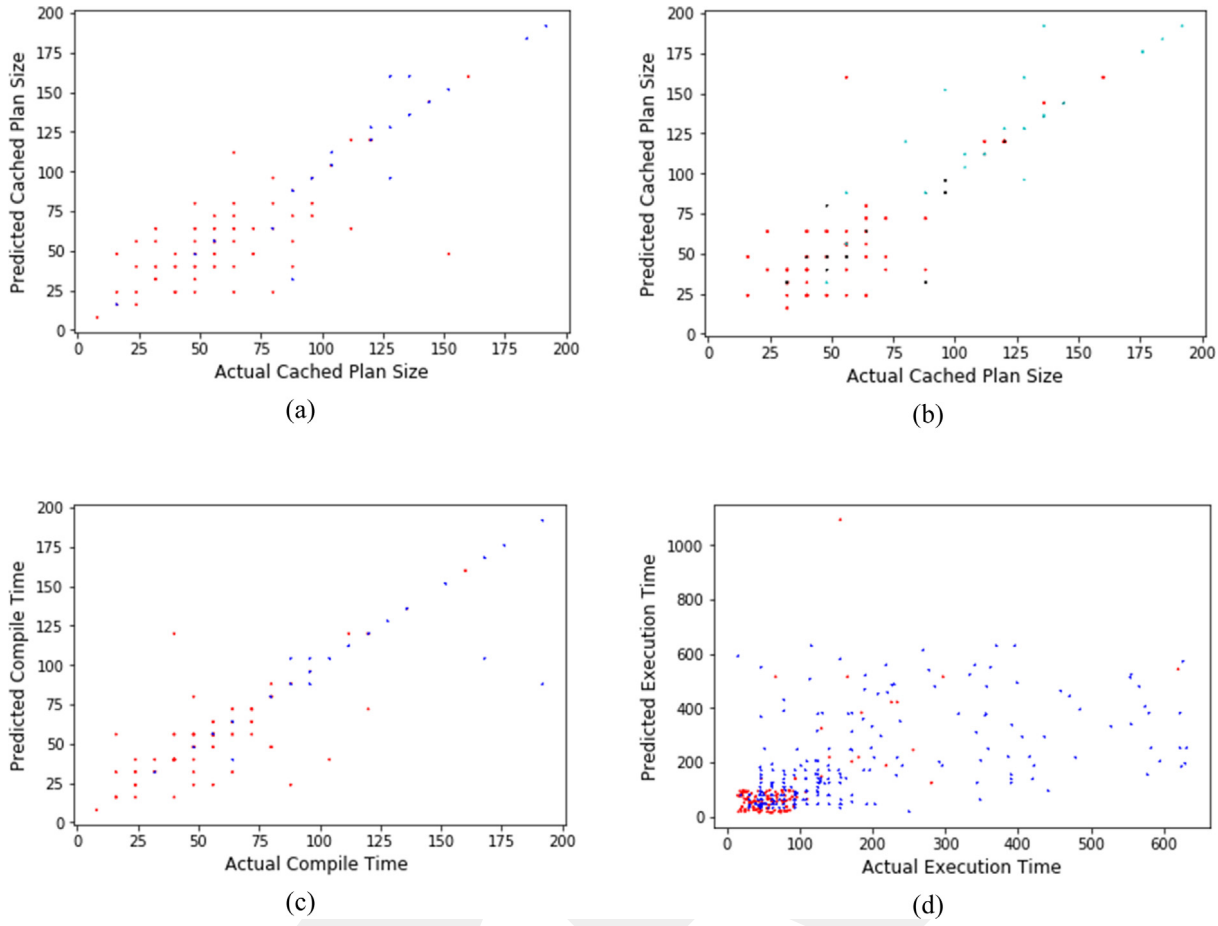

Fig. 8. Varying numbers of clusters for selected features using k-means. (a) K-means clustering of cached plan size $k = 2$, (b) K-means clustering of cached plan size $k = 3$, (c) K-means clustering of CT $k = 2$, and (d) K-means clustering of ET $k = 2$.

for the selection of an appropriate cluster. The time complexity of SOM is $O(S^2)$, where S represents the size of the sample. The algorithm is available on Github (github.com/saedakbr). Fig. 10 shows the varying numbers of clusters for selected features using SOM. Fig. 10(a)–(d) show that for $k = 2$, there is a dense cluster; however, for other values of k, such as $k = 3$, it produces sparse clusters.

5.1.4. Analysis of clustering techniques

Clustering allows creating homogeneous subgroups within the original dataset. In lazy learning, the size of the dataset is continually increasing because when the similarity of a new case does not match with existing CB results, then the new case will become part of the CB. In this way, the size of the CB rapidly increases, and it requires too much search space. Clustering allows grouping of datasets depending upon their similarity measures, such as ED or correlation-based distance. Our purpose for clustering is the grouping of similar types of data to reduce the search space for new input. We used *k-means*, *fuzzy c-means* and *SOM* clustering with our proposed CBR model. In this study, we integrated clustering with CBR for improving efficiency and prediction accuracy. The time complexity of the k-means algorithm is O(ncdi); however, the FCM and SOM algorithms have higher time complexity, $O(nc^2di)$ and $O(S^2)$ respectively, when clustering is performed on a large volume of data. Different values of k in cluster-based CBR with *k-means* were observed, and we found that our results were better when $k = 2$. The reason is that the nature of the data consists of two types, DSS and OLTP; thus, two clusters are appropriate. A higher number of clusters involves more complexity and the increased overhead of finding the similarity of each new input with the centroid distance of each cluster. Similarly, an increasing number of clusters and centroid calculations are still a big overhead for FCM and SOM. FCM produces results closer to *k-means* clustering results; however, it still requires more computational time than *k-means* clustering. Therefore, FCM and SOM are not suitable for integrating with the CBR model; *k-means* is much better to reduce the search space overhead.

We compared the performance of CBR, IBK, K*, and LWL on the same dataset. Figs. 11 and 12 compare the proposed cluster-based CBR approach with other lazy learning techniques using f-measure and accuracy, respectively.

The F-measure rate of CBR was better than the other lazy learning techniques in most of the performance metrics, except compile memory and execution time. However, considering its better performance in the majority of the performance metrics, we can say the proposed CBR performs well compared to other lazy learning techniques, which makes CBR an appropriate technique for improving the performance of DW systems. Table 3 shows the comparison of MAE with other techniques.

It can be seen that the MAE rates of other techniques, such as linear regression, Gaussian process, and multilayer perceptron, are the highest, while the proposed CBR has the lowest error rate. CBR performs well compared to other machine-learning techniques, which makes the proposed CBR approach an appropriate technique for improving the performance of DW systems.

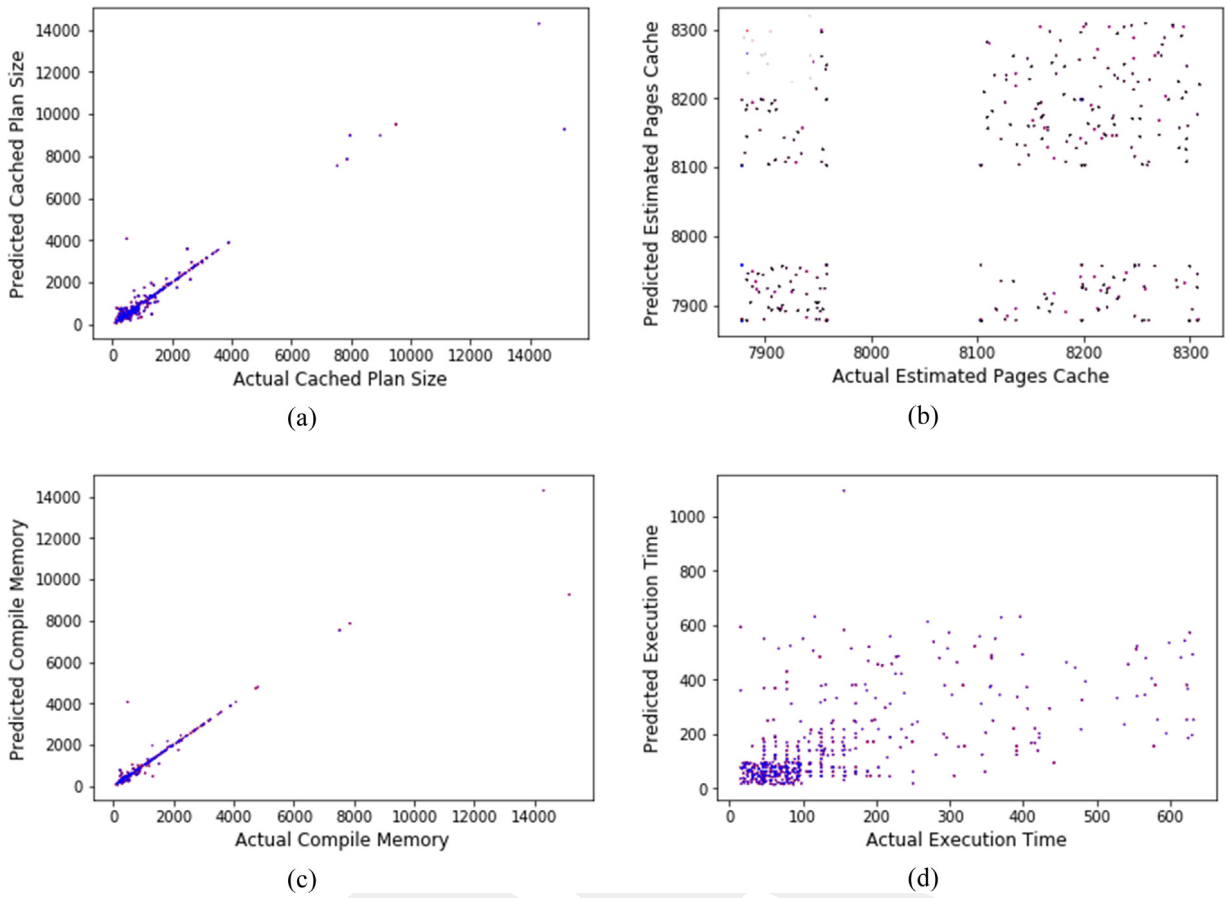

Fig. 9. Varying numbers of clusters for selected features using fuzzy c-means. (a) FCM of cached plan size $k = 2$, (b) FCM of EPC $k = 3$, (c) FCM of compile memory $k = 2$, and (d) FCM of execution time $k = 2$.

Table 3
Comparison of mean absolute error with other techniques.

Technique/performance metric	Cache plan size	Compile memory	CPU time	Estimated pages cache	Execution time	Size of node	Compile time
Proposed CBR	0.002	0.009	0.007	0.005	0.019	0.021	0.007
LWL	0.033	0.12	0.062	0.031	0.232	0.042	0.0621
K*	0.015	0.011	0.008	0.004	0.036	0.019	0.008
IBK	0.074	0.103	0.074	0.025	0.211	0.052	0.074
Linear regression	0.1937	0.2346	0.2192	0.1111	0.2833	0.1854	0.21
Gaussian process	0.262	0.3132	0.3069	0.192	0.295	0.2969	0.30
Multilayer perceptron	0.0135	0.055	0.0156	0.0027	0.0414	0.0175	0.01

5.2. Discussion

This study proposed a framework for DW query performance prediction. For a query, we selected performance metrics from an initial set of features using different feature selection techniques, and then PCA produced the best seven features. For the proposed cluster-based CBR approach, we developed two interconnected algorithms: one for clustering the CB for efficient retrieval of cases for prediction, and one for managing the four phases involved in CBR. We applied the proposed CBR model, which is a lazy learning approach that learns from previous cases and produces solutions.

In DW, query/workload is non-deterministic and dynamic. The CBR approach, due to its adaptive nature, can handle the evolving behavior of queries dynamically and makes it workload-aware or autonomic [29]. First, we perform the clustering for efficient prediction. The proposed Algorithm 1 makes clusters dynamically in an unsupervised manner, using the *k-means* clustering algorithm based on different types of workload stored in the CB repository. The elbow and Monte Carlo methods are used for selecting k and testing the accuracy of clusters, respectively. When a query enters the system, Algorithm 2 implements the four phases of CBR. It retrieves a number of cases as solutions through the *retrieve phase*, and in the *reuse phase*, the best match is selected for the prediction. When a different type of query is encountered in the system for which an appropriate solution is not found in the CB, it is revised in the *revise phase,* and the *retain phase* adapts that case as a new case in the CB for future use.

For clustering, we applied the *k-means*, fuzzy c-means, and SOM techniques. In our algorithm, the number of clusters was selected dynamically in an unsupervised manner through *k-means* using the elbow and Monte Carlo methods. The elbow method showed the range of $k = 2$ to $k = 5$ for the accuracy of clusters. The Monte Carlo method was used to determine the impact of the varying values of k on accuracy. In our study, $k = 2$ was selected. The reason is that it is the nature of data to have two types of queries, DSS and OLTP, that are handled by a DW. In this study, *k-means* clustering performed better than the *fuzzy c-means* and *SOM* techniques due to its better time complexity.

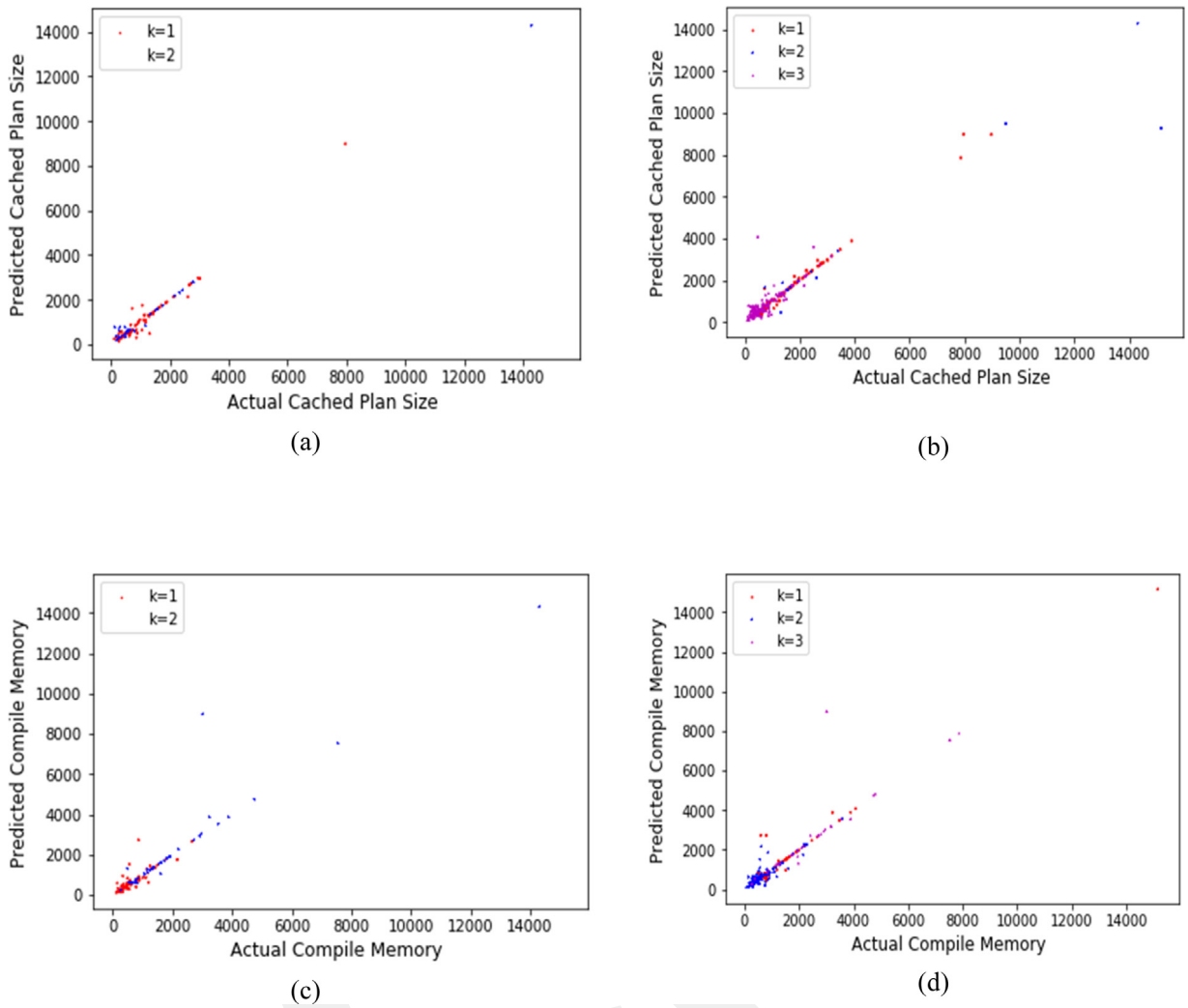

Fig. 10. Varying numbers of clusters for selected features using SOM. (a) SOM clustering of cached plan size $k = 2$, (b) SOM of cached plan size $k = 3$, (c) SOM of compile memory $k = 2$, and (d) SOM of compile memory $k = 3$.

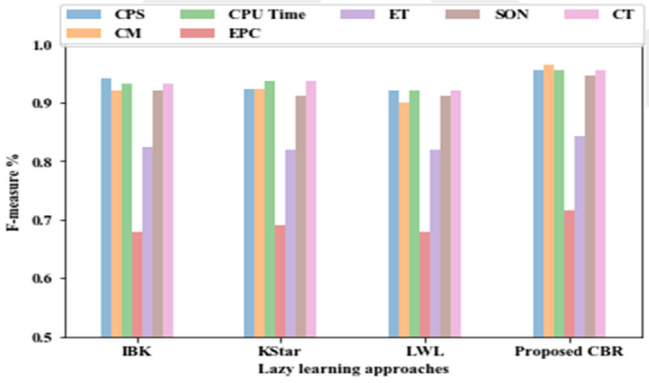

Fig. 11. Comparison of proposed cluster-based CBR approach with other lazy learning techniques using f-measure.

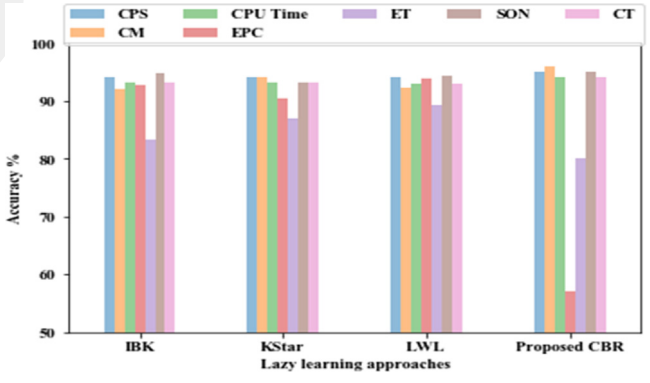

Fig. 12. Comparison of proposed cluster-based CBR approach with other lazy learning techniques using accuracy.

The lazy learning approach is good, and it deals successfully with changes in the problem domain. The size of the CB regularly increases, depending on new non-similar cases found. Clustering allows for the grouping of datasets. Instead of searching an entire large case base for a new query, only relevant clusters having shorter EDs between the new case and the centroids of the clusters are selected. To analyze the proposed cluster-based CBR approach, we compared it with other well-known lazy learning approaches, such as LWL, K*, and IBK. The results show that the accuracy and f-measure of the proposed approach were better than those of the other lazy learning approaches. Further, the error rate for the proposed approach was also very low. The results of different predicted dimensions, such as *CPS* = 95%,

CM = 96%, *CPUT* = 94%, *SON* = 95%, *CT* = 94%, show that the proposed approach outperformed other lazy learning approaches. However, the results of *LWL* in regard to *EPC* and *ET* were better than the others. Similarly, the f-measure of the proposed cluster-based CBR approach was better than that of other lazy learning approaches. The f-measure results of the proposed approach are *CPS* = 95.5%, *CM* = 96.3%, *CPUT* = 95.4%, *EPC* = 71.6%, *ET* = 84.2%, *SON* = 94.4%, and *CT* = 95.4%. Thus, the f-measure and accuracy of the proposed cluster-based CBR approach are better, and the error rate is less than the existing lazy learning approaches.

In this study, we proposed a CBR lazy learning approach that works on the basis of reasoning with historical data that exists in the CB. Lazy learning classifiers are mostly used for large continually changing data with attributes that are commonly queried. Lazy learning approaches perform prediction based on reasoning and provide solutions. Furthermore, traditional machine-learning models learn on training data and work based on that learning, so with any change in data, they require re-training. In contrast, lazy learning techniques perform prediction based on reasoning and handle the changes in data dynamically due to their predictive and adaptive abilities. Hence, they provide the solution which makes lazy learning the best as compared to machine learning. Therefore, in this study, we compared our proposed cluster-based CBR approach with other lazy learning approaches and not with traditional machine learning techniques.

6. Conclusion and future work

In this study, an autonomic performance prediction framework was developed for query performance prediction in a DW environment. Three AC characteristics were incorporated: *self-inspection* was performed on the incoming query; *self-prediction* was performed to forecast the performance of the DW query, and *self-adaptation* was performed to manage the changing behavior of a query. Many queries were executed for experimentation, and essential performance metrics were selected. A lazy learning approach, *CBR*, was used to predict the performance metrics for the query. A new query was matched with the stored cases using the four phases of CBR, and matching cases were retrieved. However, the retrieval time became high owing to the large data size. To minimize the retrieval time for test queries, instead of searching for a match in a large size CB, a *k-means* clustering technique was applied. By varying *k* in *k-means* clustering, an optimum number of clusters was achieved for $k = 2$. For comparison, other lazy learning techniques such as K^*, *IBK*, and *LWL* were applied to the same dataset. The CBR approach performed efficiently and produced the best F-measure, which was 79% for $k = 2$. The relative error for the selected features was measured, and it showed that the adequate uncertainty level was less than 9%. The results provided quantitative evidence that, overall, the *CBR* approach produced better results for performance prediction in DWs and the proposed approach is a suitable option for knowledge workers in business and industry.

In the future, we plan to enhance our proposed autonomic performance prediction framework by investigating other eager learning and lazy learning approaches to gain insight into improving the performance. Further, other AC characteristics can be incorporated to make it more autonomic.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2020.106216.

CRediT authorship contribution statement

Basit Raza: Conceptualization, Methodology, Validation, Formal analysis, Resources, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration. **Adeel Aslam:** Methodology, Software, Validation, Formal analysis, Data curation. **Asma Sher:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Writing - original draft. **Ahmad Kamran Malik:** Validation, Investigation, Writing - review & editing, Visualization. **Muhammad Faheem:** Investigation, Resources, Writing - review & editing, Supervision.

Acknowledgment

This work was supported by COMSATS University Islamabad (CUI), Islamabad, Pakistan CUI/ORIC-PD/2020.

References

[1] B. Raza, A. Sher, S. Afzal, A.K. Malik, A. Anjum, Y.J. Kumar, M. Faheem, Autonomic workload performance tuning in large-scale data repositories, Knowl. Inf. Syst. (2018) 1–37.
[2] M. Zhang, P. Martin, W. Powley, J. Chen, Workload management in database management systems: A taxonomy, IEEE Trans. Knowl. Data Eng. 30 (7) (2018) 1386–1402.
[3] A. Mehta, C. Gupta, U. Dayal, BI batch manager: A system for managing batch workloads on enterprise data-warehouses, in: Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology, 2008, pp. 640–651.
[4] M. Parashar, S. Hariri, Autonomic computing: An overview, in: International Workshop on Unconventional Programming Paradigms, Springer, Berlin, Heidelberg, 2005, pp. 257–269.
[5] M.B. Shahid, U. Sheikh, B. Raza, M.A. Shah, A. Kamran, A. Anjum, Q. Javaid, Application of data warehouse in real life: State-of-the-art survey from user preferences' perspective, Int. J. Adv. Comput. Sci. Appl. 7 (4) (2016) 415–426.
[6] B. Raza, A. Mateen, M.M. Awais, M. Sher, Survey on autonomic workload management: algorithms, techniques and models, J. Comput. 3 (7) (2011) 29–38.
[7] B. Raza, A. Mateen, T. Hussain, M.M. Awais, Autonomic success in database management systems, in: 2009 Eighth IEEE/ACIS International Conference on Computer and Information System, 2009, pp. 439–444.
[8] J.M. Medina, C.D. Barranco, O. Pons, Indexing techniques to improve the performance of necessity-based fuzzy queries using classical indexing of RDBMS, Fuzzy Sets and Systems 351 (2018) 90–107.
[9] B. Niu, P. Martin, W. Powley, R. Horman, P. Bird, Workload adaptation in autonomic DBMSs, in: Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research IBM Corporation, 2006 pp. 13-es.
[10] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wycko, R. Murthy, Hive: A warehousing solution over a map-reduce framework, in: Proceedings of the VLDB Endowment, 2009, pp. 1626–1629.
[11] S. Muller, A. Nica, L. Butzmann, S. Klauck, H. Plattner, Using object-awareness to optimize join processing in the SAP HANA aggregate cache, in: 18th International Conference on Extending Database Technology, 2015 pp. 557–568.
[12] V. Nicolicin-Georgescu, V. Benatier, R. Lehn, H. Briand, An ontology-based autonomic system for improving data warehouse performances, in: International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, Springer, Berlin, Heidelberg, 2009, pp. 261–268.
[13] A. Kemper, T. Neumann, Hyper: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots, in: 2011 IEEE 27th International Conference on Data Engineering, IEEE, 2011, pp. 195–206.
[14] R. Singhal, M. Nambiar, Predicting SQL query execution time for large data volume, in: Proceedings of the 20th International Database Engineering & Applications Symposium, ACM, 2016, pp. 378–385.
[15] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigumus, J.F. Naughton, Predicting query execution time: Are optimizer cost models really unusable?, in: 2013 IEEE 29th International Conference on Data Engineering, IEEE, 2013, pp. 1081–1092.
[16] W. Wu, Y. Chi, H. Hacigumuş, J.F. Naughton, Towards predicting query execution time for concurrent and dynamic database workloads, in: Proceedings of the VLDB Endowment, 2013, pp. 925–936.
[17] J. Duggan, O. Papaemmanouil, U. Cetintemel, E. Upfal, Contender: A resource modeling approach for concurrent query performance prediction, in: 17th International Conference on Extending Database Technology, 2014, pp. 109–120.

[18] C. Chi, Y. Zhou, X. Ye, Performance prediction for performance-sensitive queries based on algorithmic complexity, Tsinghua Sci. Technol. 18 (6) (2013) 618–628.
[19] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, I. Stoica, Ernest: efficient performance prediction for large-scale advanced analytics, in: 13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 16, 2016, pp. 363–378.
[20] S. Lee, J.S. Meredith, J.S. Vetter, Compass: A framework for automated performance modeling and prediction. in: Proceedings of the 29th ACM on International Conference on Supercomputing, 2015, pp. 405–414.
[21] E.E. Yusufoglu, M. Ayyildiz, E. Gul, Neural network-based approaches for predicting query response times, in: 2014 International Conference on Data Science and Advanced Analytics, IEEE, 2014, pp. 491–497.
[22] B. Mozafari, C. Curino, A. Jindal, S. Madden, Performance and resource modeling in highly-concurrent OLTP workloads, in: Proceedings of the 2013 ACM Sigmod International Conference on Management of Data, 2013, pp. 301–312.
[23] A. Khattab, A. Algergawy, A. Sarhan, MAG: A performance evaluation framework for database systems, Knowl.-Based Syst. 85 (2015) 245–255.
[24] M. Milicevic, M. Baranovic, K. Zubrinic, Application of machine learning algorithms for the query performance prediction, Adv. Elec. Comput. Eng. 15 (3) (2015) 33–44.
[25] C. Gupta, A. Mehta, PQR: Predicting query execution times for autonomous workload management, in: 2008 IEEE International Conference on Autonomic Computing, 2008, pp. 13–22.
[26] A. Ganapathi, A.K. Harumi, D. Umeshwar, W. Janet, F. Armando, J. Michael, P. David, Predicting multiple metrics for queries: Better decisions enabled by machine learning, in: 2009 IEEE 25th International Conference on Data Engineering, 2009, pp. 592–603.
[27] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.A. Muller, Deep learning for time series classification: A review, Data Min. Knowl. Discov. 33 (4) (2019) 917–963.
[28] G. Tsoumakas, A survey of machine learning techniques for food sales prediction, Artif. Intell. Rev. 52 (1) (2019) 441–447.
[29] K. Amin, Building an integrated CBR-big data oriented architecture for case-based reasoning systems, in: International Conference on Case-Based Reasoning, 2017, pp. 189–193.
[30] A. Wojciechowski, ETL Workflow reparation by means of case-based reasoning, Inform. Syst. Front. (2018) 21–43.
[31] T.T. Nguyen, P. Krishnakumari, S.C. Calvert, H.L. Vu, H.V. Lint, Feature extraction and clustering analysis of highway congestion, Transport. Res. 100 (2019) 238–258.
[32] B. Raza, Y.J. Kumar, A.K. Malik, A. Anjum, M. Faheem, Performance prediction and adaptation for database management system workload using case-based reasoning approach, Inf. Syst. 76 (2018) 46–58.
[33] N. Shaheen, B. Raza, A.K. Malik, A CBR model for workload characterization in autonomic database management system, in: 2018 14th International Conference on Emerging Technologies, IEEE, 2018, pp. 1–6.
[34] D.P.G. Naik, D.K. Oza, JSP Custom tag library for in-place editing in disconnected architecture–A case study, Int. J. Rec. Innov. Tre. Comput. Comput. 4 (4) (2016) 319–326.
[35] Z. Gu, R. Yang, J. Yang, X. Qiu, R. Liu, Y. Liu, Z. Zhou, Y. Nie, Dynamic Monte Carlo simulations of effects of nanoparticle on polymer crystallization in polymer solutions, Comput. Mater. Sci. 147 (2018) 217–226.
[36] S. Lee, J.S. Meredith, J.S. Vetter, Performance evaluation of classifiers for spam detection with benchmark datasets, in: 2016 International Conference on Data Mining and Advanced Computing, IEEE, 2016, pp. 17–22.
[37] M.A. Syakur, B.K. Khotimah, E.M.S. Rochman, B.D. Satoto, Integration K-means clustering method and elbow method for identification of the best customer profile cluster, in: IOP Conference Series: Materials Science and Engineering, vol. 336(1), IOP Publishing, 2018, p. 012017.
[38] O.P. Patel, N. Bharill, A. Tiwari, M. Prasad, A novel quantum-inspired fuzzy based neural network for data classification, IEEE Trans. Emerg. Top. Comput. (2019) http://dx.doi.org/10.1109/TETC.2019.2901272.
[39] L.J. Moreira, L.A. Silva, Data classification combining self organizing maps and informative nearest neighbor, in: Proceedings of IEEE International Joint Conference on Neural Networks, Vancouver, Canada, 2016, pp. 706–713.
[40] TPC-H Benchmark, 2020, http://www.tpc.org/tpch. (Accessed 28 February 2020).
[41] R. Ruch, SOMO: An Oversampling Approach Based on Self-Organized Map Oversampling and Geometric SMOTE (doctoral dissertation), 2019, http://hdl.handle.net/10362/63811.