IndexAI: AI Based Index Selection for NoSQL Databases

Mohammad Mahdi Khosravi
Computer Engineering Dept.
Middle East Technical University
Ankara, Turkey
Email: mahdi.khosravi@metu.edu.tr

Pinar Karagoz
Computer Engineering Dept.
Middle East Technical University
Ankara, Turkey
Email: karagoz@ceng.metu.edu.tr

Ismail Hakki Toroslu
Computer Engineering Dept.
Middle East Technical University
Ankara, Turkey
Email: toroslu@ceng.metu.edu.tr

***Abstract*—In the big data era, automated index selection and recommendation has been an important research problem to improve the data access efficiency. Previous efforts on artificial intelligence based database index selection have focused on relational databases. In this work, we consider the automated index selection for NoSQL databases and investigate the feasibility of supervised learning and reinforcement learning based solutions. The experiments conducted on the YCSB dataset show that reinforcement learning improves index selection performance as in relational databases, and supervised learning gives promising results and can be considered applicable under sufficient amount of training data.**

1. Introduction

Indexing is one of the most important aspects of database management systems for efficiency of data access. Although there are popular and effective index structures such as B-trees, hash index etc., automated index selection for different workloads still remains as an open problem with room for improvement. In recent years, we witness studies exploring the the use of Artificial Intelligence (AI) and Machine Learning (ML) for automated index selection and recommendation. Such recent studies mostly consider relational databases to propose automated index selection [1], [2].

With the rise of big data, NoSQL databases are widely used in modern applications and software systems due to the increasing demand for processing large amounts of data. NoSQL databases provide high scalability, performance, and availability compared to traditional relational databases. In NoSQL databases, indexing still plays an important role to provide scalability; yet under dynamic workload, it is not straightforward to have an adaptive indexing mechanism. In the literature, there are limited number of recent studies on automated index selection for NoSQL databases. As one of the most studies, in [3], Yan et al. propose the DRLISA framework, which uses deep reinforcement learning for NoSQL database index selection.

In this paper, we focus on the problem of AI-based index selection for NoSQL databases, and study two alternatives towards AI-based index selection. As the first one, we use supervised Machine Learning to recommend an index structure such as a B-tree according to the workload. The second one is the use of a Reinforcement Learning (RL) model that is based the framework SmartIX proposed by on Licks et al [1]. SmartIX is proposed for relational databases. In this work it is adapted for NoSQL databases and Yahoo! Cloud Serving Benchmark (YCSB) [4].

For the Machine Learning model, we use supervised learning based approach such that a training data set is collected to develop a model to predict the time cost of workload under a given index structure. We model the it both as regression and classification problem to compare which method would be most suitable. Our metrics for the prediction performance are as follows: (1) The R^2 value and the Root Mean Squared Error (RMSE) are calculated for the regression algorithms. (2) For classification, accuracy, recall and precision metrics are used along with f1-score.

On the other hand, RL has the benefit of not requiring an annotated data set to train a model; rather interaction between the agent and environment through a trial-and-error process is applied. Using RL, however, requires tackling a few challenges:

- A proper state representation is needed to suit the type of problem we are solving. In other words, a NoSQL database representation is needed to be modeled in the Reinforcement Learning environment.
- It is important to represent the action and reward mechanism for our model. If the reward function for index selection is too complex, the training process may require a large amount of training data to achieve good performance.

Therefore, we adapted the RL model of the SmartIX framework to NoSQL database to solve these issues.

The rest of the paper is organized as follows. Section 2 provides background information on RL, supervised Machine Learning and YCSB. Section 3 describes the proposed approach for optimal index selection. We present the experimental setup and results in Section 4 and discuss the implications of the findings in Section 5. In Section 6, related work is presented. Finally the paper is concluded with an overview in Section 7.

2. Background

2.1. Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of machine learning related to how to make optimal decisions in uncertain environments. It involves an approach where an agent learns through trial and error by interacting with an environment. The agent takes action and moves through different states of the environment while receiving feedback in the form of rewards. It is particularly useful in scenarios where a model of the environment is unknown or where the agent must interact with the environment over time to obtain feedback on its actions [5].

Formally, the agent takes actions based on the current state s of the environment and receives feedback in the form of rewards r. The goal of the agent is to learn a policy π that maximizes the expected cumulative reward over time.

RL can be formulated as a Markov Decision Process (MDP) defined by a tuple (S, A, P, R, γ), where S is the set of possible states of the environment, A is the set of possible actions that the agent can take, P is the transition probability function that specifies the probability of transitioning from state $s \in S$ to state $s' \in S$ when taking action $a \in A$, R is the reward function that maps state-action pairs to rewards, and γ is the discount factor that determines the importance of future rewards [5].

The agent's behavior is defined by a policy π, which is a mapping from states to actions. The goal of the agent is to learn an optimal policy π^* that maximizes the expected cumulative reward over time. The value function $V^\pi(s)$ and the action-value function $Q^\pi(s, a)$ are important concepts in RL. The value function $V^\pi(s)$ represents the expected cumulative reward starting from state s following the policy π. The action-value function $Q^\pi(s, a)$ represents the expected cumulative reward starting from state s, taking action a, and following policy π thereafter. The value function for a state-action pair, following a policy π, is computed using the Bellman Expectation Equation given in Equation 1.

$$Q(s, a) = E_\pi[r_{t+1} + \gamma Q(s_{t+1}, a') \mid s, a] \tag{1}$$

In the equation, the state-action value of a state can be decomposed into the immediate reward (r_{t+1}) we get on performing a certain action in state(s) (a') and moving to another state (s_{t+1}) plus the discounted value (γ) of the state-action value of the state (s_{t+1}) with respect to the some action(a) our agent will take from that state on-wards.

One popular RL algorithm is the Q-learning algorithm, which is suitable for MDP environments. The idea behind this algorithm is to incrementally update the values of $Q(s, a)$ based on the reward from an action. At each step, $Q(s, a)$ can be updated by using Equation 2.

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma_{max'_a} Q(s', a') - Q(s, a)) \tag{2}$$

There are different ways to implement the Q-learning algorithm. The basic implementation of Q-learning involves using tabular Q-value structures to store state-action values. However, this approach becomes impractical when the state space is large and there are many possible actions. It would require visiting all state-action pairs frequently enough to accurately estimate their values, which is not feasible.

Neural networks offer an alternative approach for Q-learning by approximating the Q-values using function approximation techniques. Instead of explicitly storing and updating Q-values in a table, a neural network (Q-network) is trained to learn the Q-value function, which takes the state as input and outputs the Q-values for all possible actions. This allows for a more compact representation of the Q-values and enables learning in high-dimensional domains.

2.2. Supervised Learning

Supervised learning is a branch of machine learning that aims to learn from data that is labelled with the desired output. In this work, two of the main types of supervised machine learning are used: regression and classification. Regression methods predict a continuous output value, such as the price of a house or the height of a person. Classification methods predict a discrete output value among predefined classes, such as the type of animal in a given picture. Therefore

Two well-known classifiers, Naive Bayes Classifier (NBC) and Support Vector Machine (SVM) are used in this work. NBC is based on the assumption that the features of the input are independent given the class label, and uses Bayes' Theorem to calculate the probability of each class given the input. SVM is based on the idea of finding a hyperplane that best separates the data points of different classes. The hyperplane is chosen to maximize the margin between the classes, which is defined as the distance from the hyperplane to the nearest data point of each class. SVM can also handle nonlinear data by using kernel functions that map the data to a higher-dimensional space where a linear hyperplane can be found. [6]

Regression is one of the most important and broadly used machine learning and statistics tools. It allows to make predictions from data by learning the relationship between features of the data and observed, continuous-valued responses. In this work, two common methods of regression, Linear Regression (LR) and Support Vector Regression (SVR) are used. LR is one of the most popular form of regression analysis due to its ease of use in predicting and forecasting. It assumes that there is a linear relationship between the input features and the output variable. The goal of linear regression is to find the best-fit line or curve that minimizes the sum of squared errors between the actual and predicted values. SVR is an extension of SVM as a kernel-based method that can capture non-linear and high-dimensional patterns in the data. It works by transforming the input features into a higher-dimensional space using a kernel function and then finding a hyperplane that fits the data with a maximum margin. The prediction for a new data point is then obtained by applying the same kernel function and taking the dot product with the hyperplane coefficients [6], [7].

2.3. YCSB

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source framework designed for benchmarking the performance of NoSQL databases. It provides a standardized workload and a set of performance metrics to enable users to compare the performance of different NoSQL databases under the same conditions.

YCSB is specifically designed to support various types of NoSQL databases, such as key-value stores, column-family stores, and document-oriented databases. It can generate workloads that model different types of applications, including read-intensive, write-intensive, and mixed workloads.

The benchmark consists of two main components: a workload generator and a set of performance metrics. The workload generator generates a set of operations based on the specified workload type and the underlying data model of the NoSQL database being tested. The generated operations include read, scan, update and insert. The performance metrics measure the throughput and latency of the operations, indicating the performance of the database.

3. Proposed Methods for AI-based Index Selection

3.1. Problem Definition

The problem challenged in this study can be briefly described as follows: Given the database index types and the database workload, the target is to determine the index type that will maximize the throughput of the workload.

Formally the collection of workloads is denoted as W where $|W| = N$. Each $w_i \in W$ is a workload configuration where $0 \leq i \leq |W| - 1$ and $w_i = (Re, Sc, Up, RMW, In, Op, Th)$ such that:

1) (Re, Sc, Up, RMW, In) are the *read*, *scan*, *update*, *read-modify-write* and *insert* proportions of the workload respectively. Note that the sum of the workload proportions should be equal to 1.
2) Op is an integer indicating operations to execute which would be partitioned based on the proportions in the first item.
3) Th is the throughput of the workload configuration w_i. This value is obtained by actually running the workload configuration.

I denotes the index type and in this work, it is assumed that $I \in$ (B-tree, Hash, LSM-tree) [8].

3.2. Supervised Learning based Index Selection

In the proposed supervised learning based approach, the basic idea is to use predicted throughput as a metric to suggest the index type. In other words, given the workload and index type, as the first step, the throughput is predicted. Among the index type alternatives, the one with the highest throughput prediction is considered as the index type to select. Since the throughput is a continuous value, regression algorithms are preferred for prediction. However, a classification approach is included as well, such that throughput ranges are considered as class labels.

Algorithm 1 Supervised Learning based Index Suggestion

Input: Workload Configuration W
Output: Index structure I
  $D \leftarrow$ Training data set
  Train Machine Learning model M with D
  Initialize empty dictionary L
  for each $I \in$ {B-tree, Hash, LSM-tree} **do**
    $Throughput \leftarrow M.predict(W)$
    $L[Throughput] \leftarrow I$
  end for
  $O \leftarrow max(L.keys())$
  return $L[O]$

The pseudo-code of the overall algorithm of the proposed method for index recommendation is given in Algorithm 1. Essentially the algorithm gets an input workload configuration W (as described in Section 3.1), and then for that workload configuration, it will predict the throughput as a continuous value for regression or a range for classification. This prediction is performed for the 3 index types, B-tree, LSM-tree and Hash. After this step, the algorithm will select the index type with the maximum throughput as the index structure I to select.

In the supervised learning based approach, both for regression and classification, the workload configurations annotated with index type and throughput value constitute the training dataset. Each data instance is represented with the following features: Index type (I), Read load ratio (Re), Scan load ratio (Sc), Update load ratio (Up), Read-Modify-Write load ratio (RMW), Insert load ratio (In), and Number of operations in the workload (Op).

3.2.1. Regression based Approach. In the regression based approach, the workload collection is represented as a feature vector as described above. The target output is the $Throughput(T)$, which is a numeric value. Thus the regression model predicts a continuous value as the throughput. The evaluation of the model is measured with R^2 and *Root Mean Squared Error (RMSE)* metrics. R^2 value is a measure of how well a regression model fits the data. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variables. R^2 values range from 0 to 1, where 0 means no fit and 1 means perfect fit, therefore, a higher R^2 value indicates a better fit and a lower R^2 value means a worse fit. RMSE, in turn, is the standard deviation of the prediction errors.

3.2.2. Classification based Approach. In this approach, in order to represent the problem as a classification task, the target output is remodeled as discrete value. To this aim, the throughput values are partitioned into bins and instead

of continuous throughput values, discrete bin numbers are used as the target. Let

$$B = \{B_1, B_2, ..., B_N\}$$

be a set of bins where each B_i for $1 \leq i \leq N$ corresponds to an interval of throughput for a database workload configuration. In other words, each B_i for $1 \leq i \leq N$ can be represented as a pair of endpoints (L_i, R_i) where L_i indicates the lower endpoint and R_i indicates the higher endpoint.

With this representation, classification models are trained to predict the throughput interval for the given workload. The classification model evaluation is performed with precision, recall, accuracy and F1-score metrics.

3.3. Reinforcement Learning based Index Selection

In a Reinforcement Learning (RL) model, an agent that interacts with the environment for several episodes is trained and each episode consists of a discrete number of time steps. During this process, the following hyper-parameters are considered:

- Learning rate (α),
- Discount factor (γ),
- ϵ, and
- ϵ-decay.

The ϵ value is used for having a balance of exploitation and exploration in the agent training phase. This means that the agent should start exploring the environment, and then, stabilize on exploiting the rewards. Therefore, usually, $\epsilon \geq 0.9$ in the beginning and the value is decreased, per episode, as given in Equation 3.

$$\epsilon = \epsilon - (\epsilon\text{-decay} \times \epsilon) \tag{3}$$

In RL, we also used experience replay to break any existing temporal correlations [9]. Thus, a memory buffer D which keeps the experience tuples $e = (s, a, r, s')$, where s is the state, a is the action, r is the reward, and s' is the next state, is initialized before starting the training procedure. With sampling from the experience replay buffer, a mini-batch of transitions is obtained with the same structure $e = (s, a, r, s')$. These transitions can be used for updating the parameters of the Q-network.

In the main training loop, the agent starts an episode with the initial hyper-parameters. At each step (within an episode), the agent will make a transition to a new state based on the action it has taken and get the reward of that action. At the end of the step, the agent will store a tuple $e = (s, a, r, s')$ in D and then perform the random mini-batch sampling step. When the agent has exhausted all the time steps within an episode, it will update the ϵ value, and reset the state and cumulative reward of the episode. Algorithm 2 presents the training phase, as just explained, in pseudo-code.

In this work, we used the SmartIX [1] RL framework by Licks et al., and adapted it to the NoSQL index selection problem. To this aim, significant changes have been applied to the state representation, action and reward mechanisms. SmartIX uses the columns of a relation as its basis for state representation by keeping a bit vector where each entry would correspond to the indexing status on that column. In our design, we index the attributes of the NoSQL database. The reward function in the original approach involved using the $QphH@Size$ metric given in the relational database benchmark of TPC-H [1]. In this work, we use the YCSB framework [4] and the throughput metric provided by YCSB for our reward evaluation. The actions in both approaches are similar. However, in the original paper, linear function approximation was used for the Q-learning algorithm, while we use a deep Q-learning algorithm by utilizing neural networks.

Algorithm 2 Reinforcement Learning using Deep Q-learning

```
Initialize γ, α, ε and ε-decay
Initialize number of episodes E, number of steps M
Initialize neural network Q
Initialize replay memory D
for i = 0 to E do
    for j = 0 to M do
        s', r ← a.take_action()
        D.add(⟨s, a, r, s'⟩)
        sample_and_update_network(D, Q)
        s ← s'
    end for
    ε ← ε − (ε-decay×ε)
end for
```

In the following subsections, we elaborate on the state, action and reward representation of our RL model. Additionally, the neural network architecture used within the RL model is described.

3.3.1. States. In SmartIX [1], the state is defined based on the columns of their relational table. Instead, we define our state according to the attributes (fields) of a NoSQL database. In our RL model, formally the state is a set $S = (F_1, F_2, ..., F_N)$ where $F_i = \langle Field_i, Indexed_t \rangle$. $Field_i$ is a field identifier and $Indexed_t$ is either 1 if the field is indexed at time t or 0 if not. For simplicity, the name of each field is just "field *i*" where $0 \leq i \leq (N-1)$. This is also the default name that YCSB [4] gives to the attributes when loading data onto the DBMS. We keep our set S as a vector to keep track of every field of all keys in our NoSQL database. When a field is indexed, it means that the corresponding field of all keys will be indexed.

3.3.2. Action. The actions of this model will select a field within the the set of available fields and either drop or create an index on that field based on the current status of the field. For example, in a time step, if the action selects *field 4* and if it is currently indexed, the index will be dropped and vice versa.

1. https://www.tpc.org/tpch/

Here the ϵ value plays an important role, since based on that value the agent chooses whether to take a random action or the maximum of all actions. Formally this cab written as in Equation 4.

$$a = \begin{cases} max_{a \in A}\ Q(s,a), & \text{with probability 1-}\epsilon \\ random_{a \in A}, & \text{with probability } \epsilon \end{cases} \quad (4)$$

3.3.3. Reward. As the reward value, the throughput of the workload under the index selection is used. After the agent takes action, the YCSB workload is run and the resulting throughput value is used as the reward value. Essentially, the reward function assigns a scalar reward based on the benchmarking results. The RL model aims to maximize this reward, indicating improved database performance. By associating the reward with the highest performance metric achieved during the exploration of various index configurations, the RL model learns to identify the optimal index configuration that leads to the best query performance.

3.3.4. Neural Network Architecture. As mentioned in earlier sections, we used the Deep Q-learning approach. The neural network model employed in Q-learning consists of three layers: (1) The first layer is a linear layer that maps the input (state) to a hidden layer. (2) The second layer applies the Rectified Linear Unit (ReLU) activation function to introduce non-linearity with 64 neurons. (3) The third layer is another linear layer that maps the hidden layer to the output layer, which has a size equal to the number of possible actions in the environment.

4. Experiments

In this section, we present the details of the conducted experiments on the proposed solutions. First, the experimental setup is described, and then the results are presented.

4.1. Experimental Setup

All experiments were performed on a computer with an AMD Ryzen (8 cores) processor. We used Python 3.6.10 and NoSQL version of PostgreSQL 13.11 for the development and database environment. The RL setup uses PyTorch, while for the Machine Learning experiments, Scikit-learn was used. To include the LSM-Tree index structure in PostgreSQL, we used a RockDB Foreign Data Wrapper [10].

In the experiments, 10 workload samples are selected and benchmarked using the YCSB [4] under different instances of the 3 index structures *B-tree, Hash* and *LSM-tree*. The first 5 workloads are 100% workloads which focus on one operation only. The other 5 workloads are generated randomly. For this, we wrote a script that would generate the following values randomly:

1) Operation proportions: The percentage of read, write, scan, update and read-modify-write operations.
2) Operation count: The total number of operations to run in the workload.

One constraint on the operation proportions is that the summation of the 5 operations must be equal to 1. It should also be noted that the random number of operations was in the range of [500, 5000]. In Table 1 the workload configurations used in the experiments.

4.2. Supervised Learning Setup

For supervised learning, we generated a dataset including 294 instances with the attributes described in Section 3.2. The experiments are conducted under 5-fold cross-validation.

The hyper-parameter tuning of SVM and SVR is performed by PyTorch. According to tuning process, for SVR, the C value is set to 1000, and a Radial basis function (RBF) is used as the kernel function. For SVM, a linear kernel function provides the best prediction performance.

For the classification, we split the throughput values into 9 bins where each bin would correspond to an interval of 500 $operations/s$ as given in Equation 5.

$$B_1 = [0\text{-}500], B_2 = [501\text{-}1000], ..., B_9 = [4000\text{-}4500] \quad (5)$$

The throughput prediction performance of regression and classification approaches are presented in Table 2 and Table 3. For regression, R^2 and RMSE values are reported. For the classifier, the accuracy, precision and recall along with the F1 score are presented as the performance metrics.

4.3. Reinforcement Learning Setup

In the RL model, the agent is trained in 20 episodes and each episode is 100 time steps. The agent training took about 2 days to complete. The learning rate (α) was set as 0.01 and the discount factor (γ) was 0.8. The batch size for the sampling part was chosen as 10, so, at each step, we would do the experience replay from 10 stored samples.

Initially, ϵ is set to 0.95 and it is decayed by 25% at the end of each episode. Figure 1 shows that ϵ reached its minimum value of 0.01 by the 16^{th} episode. Also, the reward accumulation was stabilized starting from episode 13 as shown in Figure 2.

5. Discussion

When the classification based prediction models are compared, it is seen that the SVM model has an accuracy score of 0.8, whereas the Naive Bayes model has a higher accuracy score of 0.86. Both models have the same precision score of 0.78, which means that they have the same proportion of true positives among all predicted positives. Similarly, the Naive Bayes model has a higher recall score of 0.86 while the SVM model has a recall score of 0.8. The F1 score, which is the harmonic mean of precision and recall, suggests further that the Naive Bayes model is more effective. Based on these observations, we can conclude

TABLE 1. WORKLOAD CONFIGURATIONS

Workload Number	Read	Scan	Update	RMW	Insert	Operation Count
1	1	0	0	0	0	1000
2	0	1	0	0	0	1000
3	0	0	1	0	0	1000
4	0	0	0	1	0	1000
5	0	0	0	0	1	1000
6	0.3	0.3	0.09	0.07	0.24	3794
7	0.18	0.23	0.21	0.11	0.27	1783
8	0.27	0.04	0.14	0.3	0.25	3768
9	0.21	0.32	0.25	0.01	0.21	3577
10	0.09	0.25	0.02	0.29	0.35	2397

TABLE 2. CLASSIFICATION PERFORMANCE RESULTS

Metrics	Classification Method	
	Naive Bayes	SVM
Accuracy	0.86	0.80
Precision	0.78	0.78
Recall	0.86	0.8
F1	0.82	0.78

TABLE 3. REGRESSION PERFORMANCE RESULTS

Metrics	Regression Method	
	SVR	Linear
Mean R^2 (of 5-folds)	0.68	0.61
R^2 Value of Test Data	0.90	0.32
RMSE	334.02	877.09

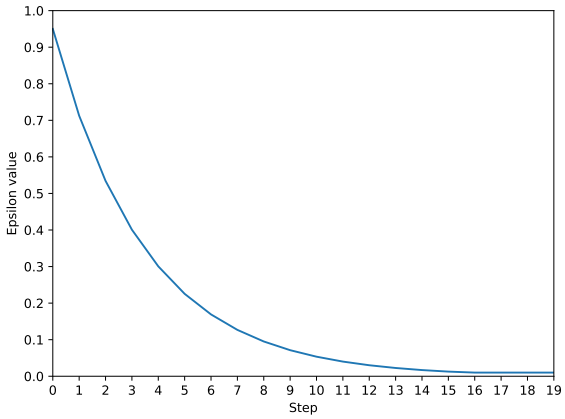

Figure 1. ϵ value per episode

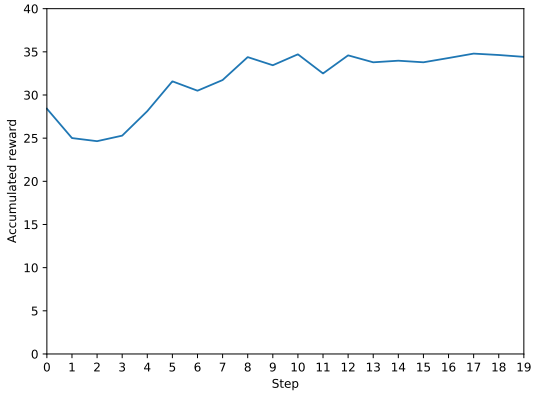

Figure 2. Accumulated reward per episode

TABLE 4. SUPERVISED LEARNING MODELS' INDEX SUGGESTIONS

Workload	NB	SVM	SVR	LR
1	B-tree	Hash	B-tree	B-tree
2	B-tree	B-tree	B-tree	B-tree
3	Hash/B-tree	B-tree	Hash	B-tree
4	Hash/B-tree	B-tree	B-tree	B-tree
5	B-tree	LSM-tree	LSM-tree	B-tree
6	B-tree	LSM-tree	B-tree	B-tree
7	B-tree	B-tree	B-tree	B-tree
8	B-tree	B-tree	B-tree	B-tree
9	B-tree	B-tree	B-tree	B-tree
10	B-tree	B-tree	B-tree	B-tree

that the Naive Bayes model outperforms the SVM for the classifiers.

The comparison of two regression models (SVR and Linear Regression), given in Table 3, shows that SVR performs much better than Linear Regression on the given data set. The SVR model has a high R^2 value of 0.9, which means that it explains 90% of the variance in the data. The Linear Regression model has a low R^2 value of 0.32, which is much worse in comparison to SVR. Comparatively, the SVR model has a low RMSE value of 334.02, and the Linear Regression model has a much higher RMSE value of 877.09, which means that it has a large average error in predicting the data. Therefore, based on these metrics, the SVR model appears more accurate and reliable than the Linear Regression.

In Table 4, the indices selected/recommended by different methods are listed. In the table it is seen that, B-tree is the best of the three index structures for any type of workload that has a mixture of different operations or is read, scan or RMW-focused. Otherwise for the exceptional case of write and/or update heavy workloads, LSM-tree is the best structure, as it is designed to handle write-intensive workloads [8]. Moreover, in the throughput comparisons, it can be seen that Hash indexing provides a better result for read-intensive workloads which only the SVM classifier manages to identify.

The throughput comparison of the learning models is presented in Table 5. Here it is seen that the RL approach can outperform 7 of the workload configurations against

TABLE 5. THROUGHPUT (OPERATIONS/SECOND) COMPARISON

Workload	RL	Naive Bayes			SVM			SVR	LR	B-tree	Hash	LSM-tree
		Min	Average	Max	Min	Average	Max					
1	2590.67	1500	1750	2000	4000	4250	4500	805.67	1355.09	3808.74	4076.64	21.97
2	862.07	1500	1750	2000	1500	1750	2000	792.59	771.45	1508.00	85.98	15.61
3	1655.63	500	750	1000	500	750	1000	790.81	957.31	617.67	686.44	22.45
4	1381.22	500	750	1000	500	750	1000	789.79	868.30	612.51	642.43	10.89
5	404.37	500	750	1000	2500	2750	3000	791.1	866.92	655.33	699.88	2718.86
6	2148.36	1000	1250	1500	1000	1250	1500	1281.03	1055.01	1443.68	110.14	16.01
7	1523.93	1000	1250	1500	500	750	1000	919.88	969.36	1172.26	118.53	15.25
8	1320.25	500	750	1000	1000	1250	1500	1067.2	1069.45	1420.81	379.69	14.55
9	2140.63	1000	1250	1500	1000	1250	1500	1209.7	1018.73	1612.71	94.89	17.78
10	1032.30	1000	1250	1500	500	750	1000	950.71	920.22	1086.09	106.16	14.51
Accumulated Throughput	15059.43	9000	11500	14000	13000	15500	18000	9398.48	9851.84	13937.8	7000.78	2867.88

TABLE 6. THROUGHPUT COMPARISON ON PROPOSED RL METHOD VS. DRLISA [3]

Workload Number	Proposed RL model	DRLISA
1	2590.67	2403.84
2	862.07	39.25
3	1655.63	4524.90
4	1381.22	38.70
5	404.37	2415.50
Accumulated Throughput	6893.96	9422.19

the three baseline index structures (given in the rightmost three columns). Additionally, the accumulated throughput value of all 10 workloads indicates that, on the overall, the RL agent still manages to increase performance by at least 8% against the baseline results and by 30% against the Supervised Learning models (if we consider the average value of the bins in the classifiers). The only exception is SVM where the accumulated average value is higher by 2.9%.

When we compare the time cost between training a Supervised Learning model and RL, as also mentioned in Section 4.3, RL model required 2 days for the agent to complete its training phase, whereas the Supervised Learning models took only a few minutes to get trained. This shows a large difference between the time cost of the two approaches. On the other hand, Supervised Learning would require an annotated training dataset, while Reinforcement Learning can be trained through interaction with the environment.

In Table 6, a comparison between the proposed RL model and a previous solution in the literature, DRLISA [3] is given. The throughput results for the first 5 workloads given in [3] are compared with the results obtained with the proposed RL model. According to the results, DRLISA manages to outperform the proposed RL approach (considering the accumulated throughput) by 26%. However, this result is not conclusive since only a limited set of throughput performance is reported in [3]. Additionally, it is seen that the index suggestion does not provide stable results. For the workload 3, DRLISA provides a very high output, whereas for the workloads 2 and 4, the performances are very low compared to the proposed solution. On the other hand, the proposed RL solution provides a comparatively steady throughput performance.

6. Related Work

In the literature, automated index selection and recommendation has been studied mostly for relational databases. Studies on NoSQL databases have only started recently and the number of such studies is still few.

For relational databases, the previous index selection studies focus on cost estimation and generally employ RL as learning technique [1], [11], [12], [13], [14], [15].

In a recent study by Gao et al. [16] both a cost estimator and a Deep Reinforcement Learning (DRL) based index selector are proposed. The proposed cost estimator uses a graph convolution network and is able to make accurate estimations without using much storage. Their DRL-based index selector is merged with their implemented cost estimator to improve index selection performance.

In [17], Shi et al. also focus on relational databases and introduce a machine learning-based index benefit estimator (LIB) to overcome the limitations of index benefit estimation based on "what-if" calls. The authors claim that their approach is the first machine learning-based approach to quantify index benefits. Their evaluations show that LIB not only outperforms the "what-if" method but also reduces prediction errors significantly (by 91%).

Kossman et al. [18] present a performance study of RL-based index selection. Additionally, the authors introduce a novel RL-based index selection framework, which matches the performance of the state-of-the-art while outperforming their runtime. Their solutions involve having a complex model for workloads which allows generalizing it to unseen workloads. Finally, their approach supports multi-attribute indexes which is not done by other RL-based solutions.

Additionally, Kossman's dissertation [19] aims to improve DBMS performance by researching two directions. The first is introducing new algorithms for efficient index selection and the second direction is query optimization by utilizing data dependencies. For efficient index selection two methods, SWIRL and Extend which complemented each other, are proposed. The approaches taken in this dissertation

are experimented with industry-standard benchmarks where promising results are shown.

As one of the most recent studies on index selection for NoSQL Databases, in [3], Yan et al. propose the DRLISA framework, which uses DRL for NoSQL database index selection. As NoSQL database, a document database is used, and the experiments demonstrate the improvement by RL based index selection. However, crucial details such as workload representation and RL implementation are not presented. This prevents the repeatability of the results.

In [20], Chawathe outlines a cost model for NoSQL databases, particularly those on the cloud, to capture the specific monetary costs associated with database operations in this context. The paper highlights the importance of this problem and the lack of reported work on such a cost model for NoSQL databases despite its necessity. The development of such a cost model allows for optimal (or near-optimal) index selection in NoSQL database systems.

7. Conclusion

The motivation for this study was to explore AI for index selection in NoSQL databases. Considering the fact that a large majority of work focused on relational databases and using RL only, it is worthwhile to see how we could use Supervised Learning to obtain index suggestions for different workloads. Additionally, we provided a RL based approach to get better index structures in NoSQL databases which has a different architecture than the DRLISA [3] framework. The proposed RL method is based on SmartIX [1] where the framework was modified to make it applicable for NoSQL databases.

In our evaluation, it is observed that, using RL to find an optimal index structure based on attributes of a NoSQL database, leads to an overall increase in the throughput. Suggestions by regression models, in contrast, do not seem to provide improvement in comparison to only using B-tree for any workload configuration. Classification, on the other hand, can be a more effective approach to predict optimum index structure while sacrificing having a continuous valued output. The performance of classification based approach has potential for improvement if larger training datasets can be provided.

Acknowledgments

This research received the support of EXA4MIND project, funded by a European Union´s Horizon Europe Research and Innovation Programme, under Grant Agreement N° 101092944. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] G. P. Licks, J. L. Couto, P. De Fátima Miehe, R. De Paris, D. D. Ruiz, and F. Meneguzzi, "SmartIX: A database indexing agent based on reinforcement learning," *Applied Intelligence*, vol. 50, no. 8, pp. 2575–2588, 8 2020. [Online]. Available: https://doi.org/10.1007/s10489-020-01674-8

[2] A. Sharma, F. M. Schuhknecht, and J. Dittrich, "The case for automatic database administration using deep reinforcement learning," *CoRR*, vol. abs/1801.05643, 2018. [Online]. Available: http://arxiv.org/abs/1801.05643

[3] Y. Yan, S. Yao, H. Wang, and M. Gao, "Index selection for NoSQL database with deep reinforcement learning," *Information Sciences*, vol. 561, pp. 20–30, 6 2021.

[4] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," *Symposium on Cloud Computing*, 6 2010.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2015.

[6] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 9 1995. [Online]. Available: https://doi.org/10.1007/bf00994018

[7] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, pp. 199–222, 2004.

[8] J. Shin, J. Wang, and W. G. Aref, "An update-intensive lsm-based r-tree index," *arXiv preprint arXiv:2305.01087*, 2023.

[9] R. Liu and J. Zou, "The effects of memory replay in reinforcement learning," 2017.

[10] Vidardb, "pgrocks-fdw: A Foreign Data Wrapper for PostgreSQL using RocksDB." [Online]. Available: https://github.com/vidardb/pgrocks-fdw

[11] S. Das, M. Grbic, I. Ilic, I. Jovandic, A. Jovanovic, V. R. Narasayya, M. Radulovic, M. Stikic, G. Xu, and S. Chaudhuri, "Automatically indexing millions of databases in microsoft azure sql database," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 666–679.

[12] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya, "Ai meets ai: Leveraging query executions to improve index recommendations," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1241–1258.

[13] J. Kossmann, S. Halfpap, M. Jankrift, and R. Schlosser, "Magic mirror in my hand, which is the best in the land? an experimental evaluation of index selection algorithms," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2382–2395, 2020.

[14] H. Lan, Z. Bao, and Y. Peng, "An index advisor using deep reinforcement learning," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2105–2108.

[15] Z. Sadri, L. Gruenwald, and E. Lead, "Drlindex: deep reinforcement learning index advisor for a cluster database," in *Proceedings of the 24th Symposium on International Database Engineering & Applications*, 2020, pp. 1–8.

[16] J. Gao, N. Zhao, N. Wang, S. Hao, and H. Wu, "Automatic index selection with learned cost estimator," *Information Sciences*, vol. 612, pp. 706–723, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025522009379

[17] J. Shi, G. Cong, and X.-L. Li, "Learned index benefits: Machine learning based index performance estimation," *Proc. VLDB Endow.*, vol. 15, no. 13, p. 3950–3962, sep 2022. [Online]. Available: https://doi.org/10.14778/3565838.3565848

[18] J. Kossmann, A. Kastius, and R. Schlosser, "Swirl: Selection of workload-aware indexes using reinforcement learning." in *EDBT*, 2022, pp. 2–155.

[19] J. Kossmann, "Unsupervised database optimization," Doctoral thesis, Universität Potsdam, 2023.

[20] S. S. Chawathe, "Index-selection for minimizing costs of a nosql cloud database," in *Economics of Grids, Clouds, Systems, and Services*, K. Djemame, J. Altmann, J. Á. Bañares, O. Agmon Ben-Yehuda, V. Stankovski, and B. Tuffin, Eds. Cham: Springer International Publishing, 2020, pp. 189–197.