Indexer++: Workload-Aware Online Index Tuning with Transformers and RL

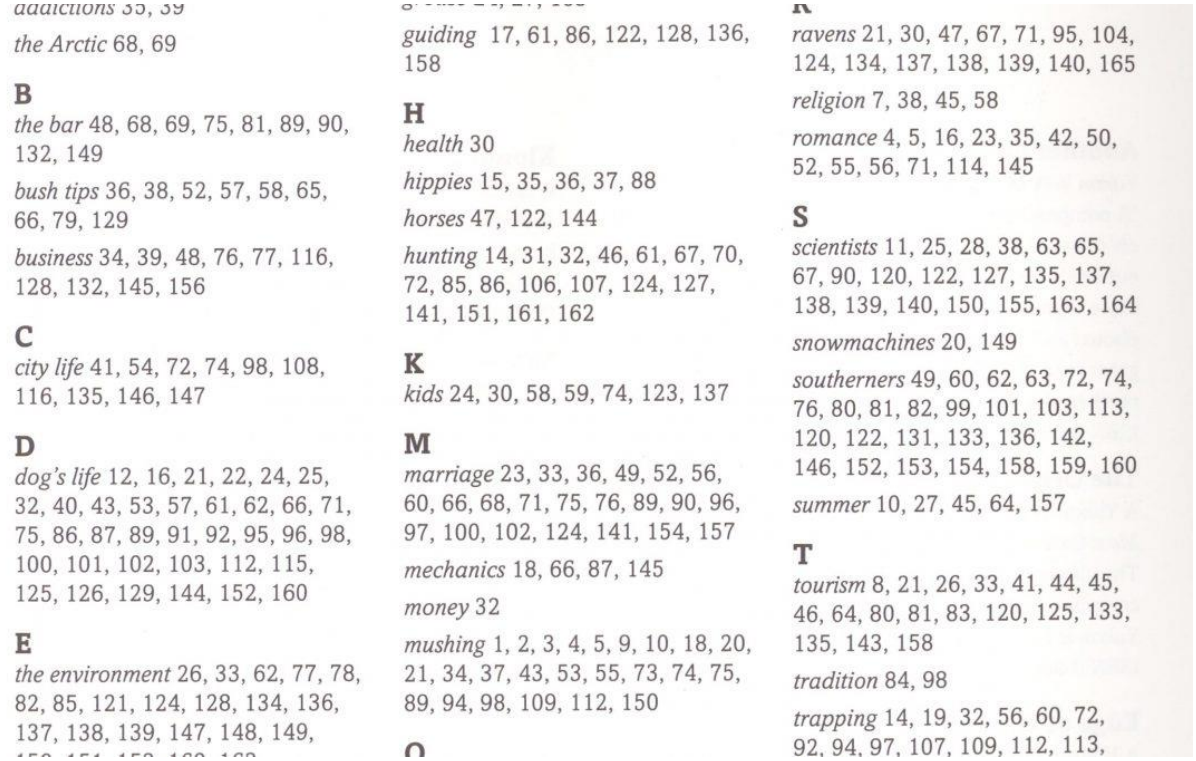
Tutorial on autonomous, workload-aware, online index tuner

SRIN Tutorial #02

Contents

Overusing Index Problem 3
Workload Embedding and Trend Detection with Pre-Trained Transformer 5
Online Index Selection 6
Priority Experience Sweeping with DQN 7
Experiments Result 8

Overusing Index Problem



addictions 35, 39

the Arctic 68, 69

B

the bar 48, 68, 69, 75, 81, 89, 90, 132, 149

bush tips 36, 38, 52, 57, 58, 65, 66, 79, 129

business 34, 39, 48, 76, 77, 116, 128, 132, 145, 156

C

city life 41, 54, 72, 74, 98, 108, 116, 135, 146, 147

D

dog's life 12, 16, 21, 22, 24, 25, 32, 40, 43, 53, 57, 61, 62, 66, 71, 75, 86, 87, 89, 91, 92, 95, 96, 98, 100, 101, 102, 103, 112, 115, 125, 126, 129, 144, 152, 160

E

the environment 26, 33, 62, 77, 78, 82, 85, 121, 124, 128, 134, 136, 137, 138, 139, 147, 148, 149,

guiding 17, 61, 86, 122, 128, 136, 158

H

health 30

hippies 15, 35, 36, 37, 88

horses 47, 122, 144

hunting 14, 31, 32, 46, 61, 67, 70, 72, 85, 86, 106, 107, 124, 127, 141, 151, 161, 162

K

kids 24, 30, 58, 59, 74, 123, 137

M

marriage 23, 33, 36, 49, 52, 56, 60, 66, 68, 71, 75, 76, 89, 90, 96, 97, 100, 102, 124, 141, 154, 157

mechanics 18, 66, 87, 145

money 32

mushing 1, 2, 3, 4, 5, 9, 10, 18, 20, 21, 34, 37, 43, 53, 55, 73, 74, 75, 89, 94, 98, 109, 112, 150

O

R

ravens 21, 30, 47, 67, 71, 95, 104, 124, 134, 137, 138, 139, 140, 165

religion 7, 38, 45, 58

romance 4, 5, 16, 23, 35, 42, 50, 52, 55, 56, 71, 114, 145

S

scientists 11, 25, 28, 38, 63, 65, 67, 90, 120, 122, 127, 135, 137, 138, 139, 140, 150, 155, 163, 164

snowmachines 20, 149

southerners 49, 60, 62, 63, 72, 74, 76, 80, 81, 82, 99, 101, 103, 113, 120, 122, 131, 133, 136, 142, 146, 152, 153, 154, 158, 159, 160

summer 10, 27, 45, 64, 157

T

tourism 8, 21, 26, 33, 41, 44, 45, 46, 64, 80, 81, 83, 120, 125, 133, 135, 143, 158

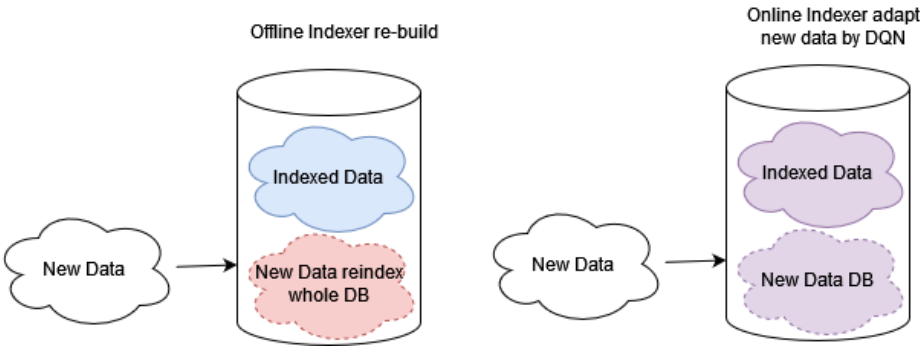
tradition 84, 98

trapping 14, 19, 32, 56, 60, 72, 92, 94, 97, 107, 109, 112, 113,

Book indexing. Image taken from Book Editing Associates.

Indexing is an important task in a database. Let's say we have a book with 500 pages, with 50 chapters. Find the word “tourism”, if we do not have indexing then we simply need to read the whole book to find them. But what if we have a book index? Then, we simply look it up and speed up the search process to only one page. In a database similar thing occurs. To speed up large databases, we create an index to use when fetching reading data later.

Now, the problem with databases is they are not books. They are not static and usually, databases grow in size exponentially. Then as similar case to the book, how do we deal with new data being created? We need to rebuild an index, and the over usage of the index causes severe impact on database performance.



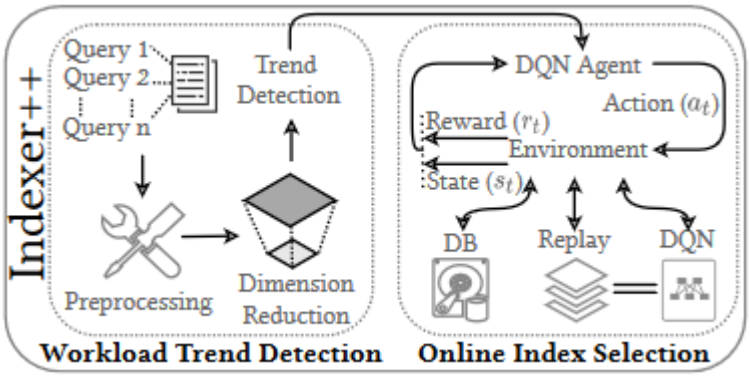
Using online indexer can improve performance over offline indexer counterpart.

Index selection costs need to be minimalized to achieve good performance in the database. As of current, much research has shown automated index selection can outperform manual index selection by utilizing Machine Learning. This automated index selection research mostly focuses on offline method or non-continuous.

However, overusing index cause problem especially when writing operations occurs on database. When many writing operations occurs the index become stale and the offline automated indexer need to recompute indexes which could take hours.

Another way, online method for automated indexing can prevent writing operation problems by keeping the set of indexes fresh. Online method has the ability to progressively adapt the set-in response to workload or database changes. While online indexer has its own problem, indexer++ were able to mitigate them. Some of the challenges, online indexer needs to be 1. Noise resilience, 2. Excellent overhead performance, 3. Trend detection, and 4. Responsiveness to tuning on new data.

In this tutorial, we will cover how online indexer++ works.

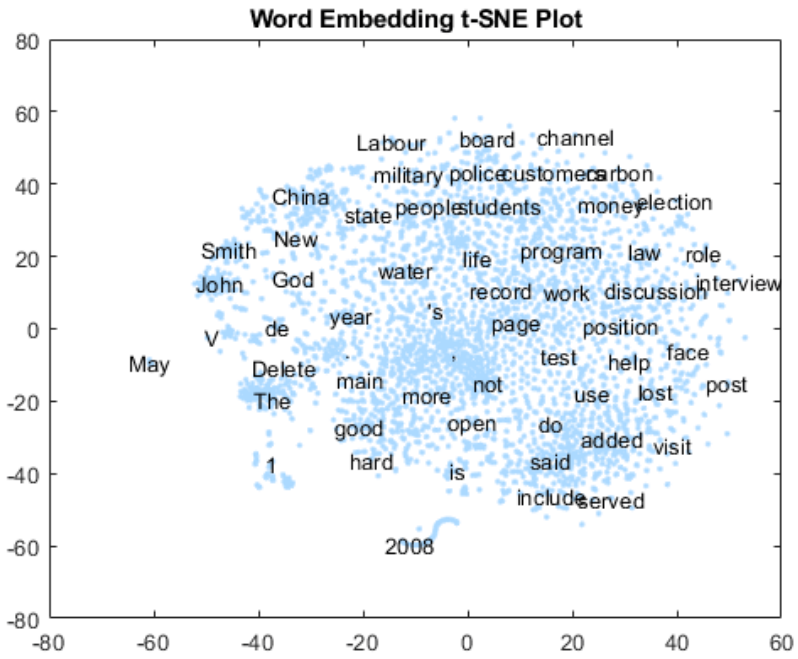


Indexer++ framework with two phases (i) Workload Trend Detection (left) (ii) Online Index Selection (right)

In general, the indexer has 2 step process, first queries get represented as workload embeddings in which workload trends are identified with K-medoids clustering. After finishing with the first workload trend detection, the second phase selects set of indexes using online deep reinforcement learning.

Workload Embedding and Trend Detection with Pre-Trained Transformer

Embedding in Natural Language Processing (NLP) is a technique to compress many different objects into a vector representation of a word. From word2vec to other technique such as CNN, RNN, LSTM, and finally Transformer model with examples of BERT, Transformer-XL, XML, word embedding has changed the field.



Visualization Word Embedding. Image taken from MathWorks.

The idea of embedding is to transform similar words near one another. In indexer however, universal embedding implemented to represent workload effectively and identified similar workloads. After transforming to embedding with Transformer RoBERTa (best model in later result chapter), the embeddings use t-sne (t-distributed stochastic neighbor embedding) to reduce the dimension and finally cluster them together. To cluster index++ use K-medoids clustering.

Online Index Selection

Deep Q-learning or DQN is value base function Reinforcement Learning algorithm. Where with neural networks it inputs a high-dimensional input data as state and we try to update parameters within the network with following equation

$$Q(s,a;\theta) = Q(s,a;\theta) - \alpha \frac{\partial}{\partial Q(s,a;\theta)} Loss(\theta)$$

Where the loss is

$$Loss(\theta) = \mathbb{E}_{\pi}\left[\frac{1}{2}\overbrace{(Bellman}^{Target} - \overbrace{Q(s,a;\theta))^2}^{Estimate}\right]$$

And in the Bellman equation we can define our reward function for DBMS representation as such

$$r_t = max\left(\frac{Cost(W, I_{t_0}, D)}{Cost(W, I_{t_{n-1}}, D) - Cost(W, I_{t_n}, D)} - 1, 0\right) + r_{size}$$

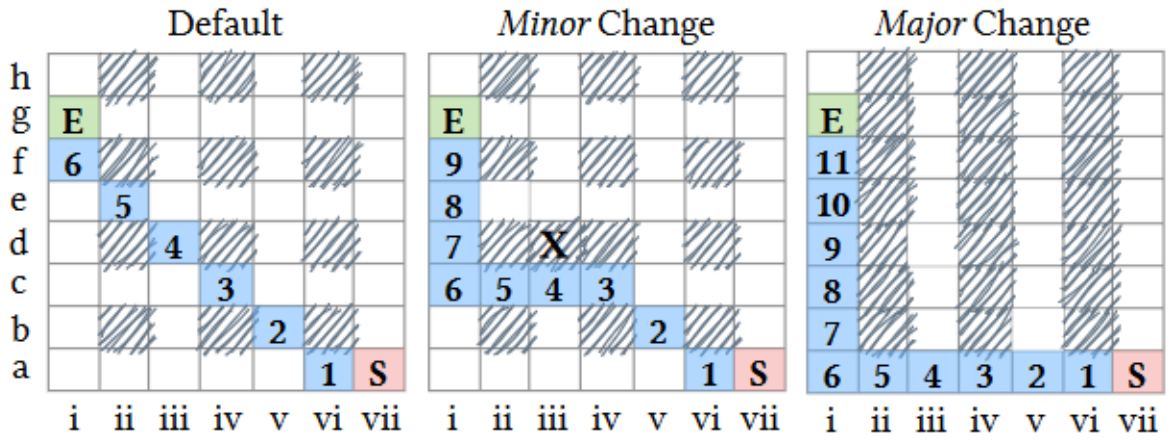
$$\text{where, } r_{size} = \begin{pmatrix} 1 & \text{total index size} < max\_allowed\_size \\ -1 & \text{otherwise} \end{pmatrix}$$

Where max_allowed_size is the maximum constrain in selected indexes e.g., 100MB.

Lastly, Experience Replay commonly used on DQN unfortunately not suitable due uniform sampling leading to slow convergence. To solve the problem, with *Priority Experience Replay (PER)* rather than sampling uniformly, sample are given weight based on their temporal difference error to prioritize experiences.

Priority Experience Sweeping with DQN

DQN is an online learning process that interacts with environment to learn optimum behaviour similar to index tuning process to reduce query cost. But if a workload change, then the environment essentially also changes as well. Usually if this such case happened, DQN will retrain from the beginning, but we can avoid retraining by using *Priority Experience Sweeping*. The idea behind PES is that the environment while changes, the state and action of the environment do not change. With this, rather than retraining we can utilize previous learned behaviour.



This changes in query workload usually only in a minor or major and by PES, we can adjust agent without retrain from scratch.

Experiments Result

For experiment in particular there are two common database tasks used as benchmark:

1. IMDB real-world movie database. With 21 tables and 33 queries. Where the queries create as test scenario on performance of the framework.
2. TPC-H, an industrial benchmark for databases. While it does not have a fixed set of queries, queries can be generated by the provided 22 query templates.

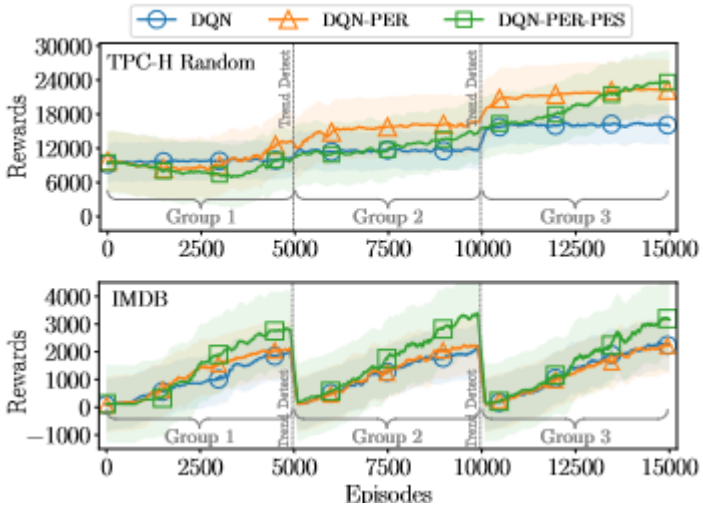
Q1. What is the best combination of universal embedding, clustering, and dimension reduction method for the SQL workload representation?

Vector Representation				Dimentionality Reduction				Clustering			
With SQL Keyword		No SQL Keyword		With SQL Keyword		No SQL Keyword		With SQL Keyword		No SQL Keyword	
BERT	0.77	BERT	0.66	UMAP	0.79	UMAP	0.74	K-Mn	0.78	K-Mn	0.72
ALBERT	0.70	ALBERT	0.57	PCA	0.77	PCA	0.70	K++	0.79	K++	0.72
RoBERTa	0.85	RoBERTa	0.82	TSNE	0.80	TSNE	0.73	K-Md	0.79	K-Md	0.73
XLM	0.57	XLM	0.48								
T5	0.80	T5	0.71								

Table 1: Average accuracies on TPC-H Random and IMDB on Dimentionality Reduction, Vector Representation and Clustering

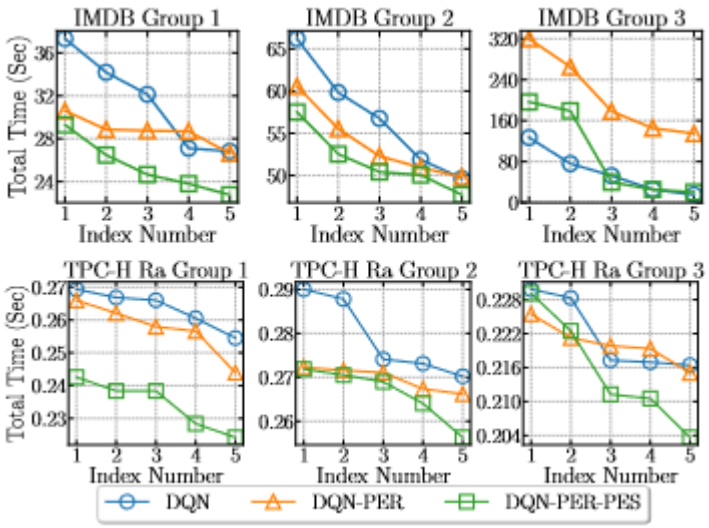
As mentioned on earlier chapter above, the use of RoBERTa+t-SNE+k-Medoids currently is the best performance out of the bunch possible candidate.

Q2. DQN with priority experience sweep learning will it learn and identify workload trends?



Experiment shows above, the green line gives noticeable improvement above baseline. With different behaviour of both TPC-H and IMDB, DQN with PES was able to detect minor and major workload changes.

Q3. Are the selected indexes efficient and reduce the workload execution cost?



From experiment the execution cost from normal DQN, DQN-PER, DQN-PER-PES, indicate that ideal behaviour is shown from every algorithm.

Further improvement (still need more research and experimentation)

- Better efficient online reinforcement learning algorithm
- Directly use self-supervised automatic clustering (FAIR Swav online target)