Journal of Information Systems Engineering and Management
2025, 10(11s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Exploring the Impact of Data Locality in Distributed Databases: A Machine Learning-Driven Approach to Optimizing Data Placement Strategies

Rafea M. Ibrahim
Department of Hadith and its Sciences, College of Islamic Sciences, Al-Iraqia University, Baghdad, Iraq
Corresponding author : Rafea Mohammed Ibrahim
rafea.ibrahim@aliraqia.edu.iq

ARTICLE INFO

Received: 17 Nov 2024
Revised: 25 Dec 2024
Accepted: 12 Jan 2025

ABSTRACT

Data locality significantly influences the performance of distributed databases, affecting query response times and resource utilization. This study investigates the role of data locality in enhancing the efficiency of distributed systems through a machine learning-driven approach to optimize data placement strategies. By analyzing access patterns, network latencies, and computational loads, we develop predictive models that inform dynamic data placement decisions. Utilizing reinforcement learning algorithms, the study adapts to fluctuating workloads, effectively minimizing data transfer times and maximizing throughput. Empirical results illustrate substantial improvements in query performance and resource management, highlighting the efficacy of intelligent data locality strategies. This study paves the way for future advancements in artificial intelligence-driven optimization for distributed database architectures.

Keywords: Data Locality, Distributed Databases, Data Placement Strategies, Machine Learning.

INTRODUCTION

Within the domain of dispersed databases (DBs), optimizing query execution could be a basic concern that essentially impacts the productivity and adequacy of information administration frameworks. Disseminated DBs, which spread information over numerous servers or areas, point to upgrade unwavering quality, blame resistance, and get to speed. In any case, this conveyance presents complexities that can corrupt inquiry execution, making optimization a challenging errand [1].

Inquiry execution in dispersed frameworks is affected by different components, counting arrange inactivity, stack conveyance, and information region. As questions navigate over distinctive hubs or locales, delays can happen due to arrange communication and the time required for information recovery and preparing. Moreover, the dispersion of information over numerous servers can lead to uneven stack adjusting and asset dispute, assist worsening execution issues.

Given these challenges, conventional optimization procedures frequently fall short in tending to the nuanced execution elements of disseminated DBs. Typically, where machine learning (ML) strategies come into play, advertising promising arrangements for making strides inquiry execution. By leveraging progressed calculations and models, ML can analyze complex designs in inquiry behavior and framework execution, empowering more successful and versatile optimization techniques [2].

The essential objective of utilizing ML in inquiry optimization is to diminish idleness, increment throughput, and upgrade asset utilization. ML models can foresee inquiry execution times based on verifiable information, recognize irregularities that demonstrate execution corruption, and powerfully alter optimization procedures in reaction to changing conditions. These capabilities are especially important in conveyed situations where conventional

optimization approaches may battle to keep pace with the energetic nature of inquiry workloads and framework states [3].

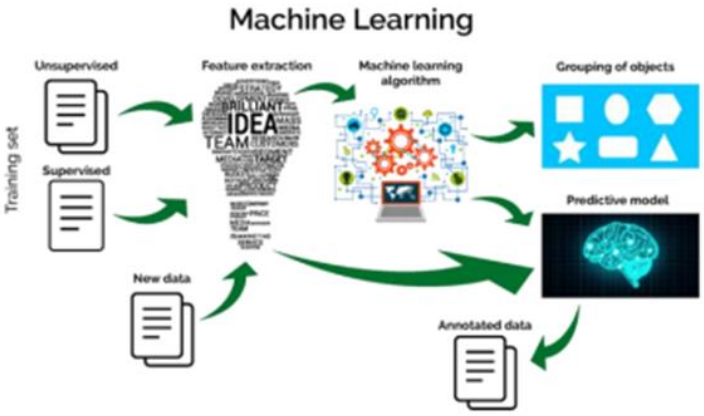

Figure 1. ML Model Workflow

In this consider, we'll investigate the elemental challenges related with query execution, dive into different ML methods that can be utilized for optimization, and examine the down to earth usage of these strategies. Moreover, we'll look at real-world case ponders to demonstrate the adequacy of ML in progressing DB execution and conclude with bits of knowledge into future patterns and investigate bearings in this advancing field.

By joining ML into inquiry optimization, dispersed DBs can accomplish critical changes in productivity and execution, tending to the impediments of conventional strategies and clearing the way for more cleverly and versatile information administration frameworks [2].

Dispersed DBs are frameworks planned to oversee and store information over numerous physical areas, permitting for moved forward unwavering quality, versatility, and accessibility. Not at all like conventional, centralized DBs, which store all information in a single area, disseminated DBs spread information over an organization of interconnected hubs or servers. This engineering is especially valuable for large-scale applications that require tall execution and blame resilience [4].

The center components of a disseminated DB include nodes, information fracture, and replication. Hubs are person servers or frameworks that collectively oversee the DB. Each hub stores a parcel of the information and partakes in inquiry handling. The conveyed nature of these DBs implies that information isn't held in a single put but is instep divided and conveyed over distinctive hubs. Information fracture all udesto the handle of separating a DB into littler, manageable pieces, known as parts. These parts can be dispersed based on different criteria, such as information sort, utilization designs, or topographical area. Fracture makes a difference to optimize execution by localizing information get to and decreasing the stack on any single hub.

Replication is another key viewpoint of conveyed DBs. It includes making duplicates of information over different hubs to improve accessibility and blame resilience. Replication guarantees that in case one hub comes up short, other hubs with duplicated information can proceed to supply get to, in this way keeping up the by and large astuteness and accessibility of the DB. This repetition is basic for frameworks that require tall accessibility and continuous operation.

The method of inquiry execution in a dispersed DB is more complex than in centralized frameworks due to they have to be arrange between different hubs. When a inquiry is issued, it may ought to get to information put away on distinctive hubs, requiring productive communication and information recovery techniques. The inquiry execution handle includes a few steps, counting inquiry decay, information recovery, and result accumulation. Inquiry deterioration breaks down a inquiry into sub-queries that can be executed in parallel over different nodes. Information recovery includes getting to the important parts of information put away on different hubs, and result conglomeration combines the comes about from these hubs to deliver the ultimate yield [5].

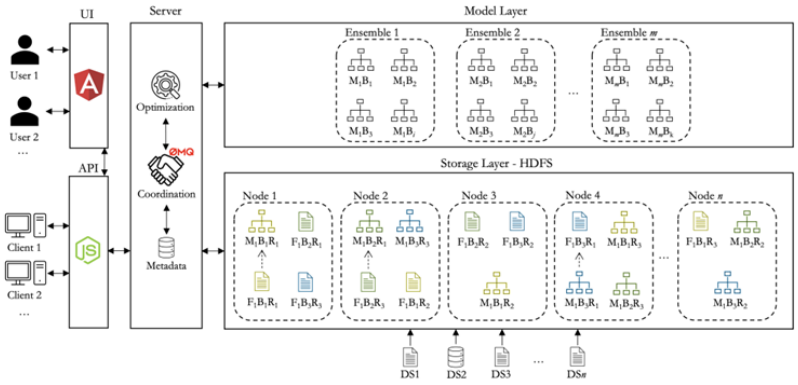

Figure 2. Distributed DB Chart

A few components influence inquiry execution in disseminated DBs. Arrange inactivity, or the delay in information transmission between hubs, can altogether affect inquiry reaction times. As information is conveyed over numerous areas, the time it takes to transmit information over the organize can add to the by and large execution time. Load distribution is another basic calculate, as uneven dissemination of questions and information can lead to bottlenecks and diminished execution. Viable stack adjusting techniques are basic to guarantee that no single hub gets to be overpowered with demands whereas others stay underutilized.

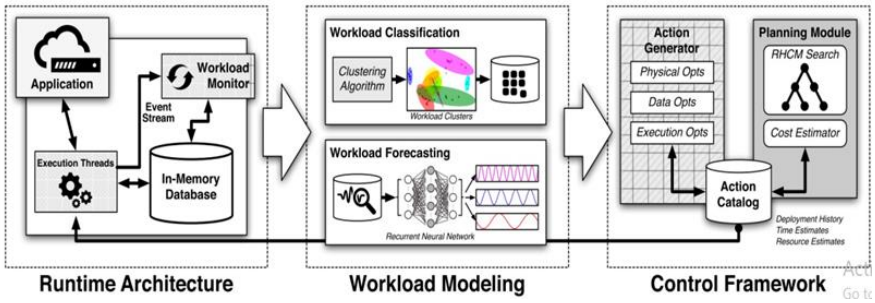

Figure 3. Model Architecture

Importance

The importance of exploring location affects distributed DBs is important, especially when using ML to improve where data is stored. This is important for a few main reasons:

1. Improving Performance: In distributed DBs, how close the data is to the computers doing the processing can greatly affect how fast queries run. When data is stored near the computers or users that need it, it takes less time to send and receive it. This means queries get answered more quickly. Using ML to improve where data is stored helps automatically change the data's location based on how people use it, the amount of work being done, and the types of questions being asked. This makes the whole system work better.
2. Scalability and Flexibility: Distributed DBs are made to work well with large applications and can manage increasing amounts of data. Improving where data is stored helps the system grow easily without slowing down as more data is added. ML programs can look at lots of old data and guess how things will be used in the future. This helps the DB manage growth while still working well.
3. Saving Money: If data isn't stored in the right place, it can cause a lot of unnecessary data movement and competition for resources. This can lead to higher costs, especially in cloud services where moving and storing data can be expensive. Using ML can help lower these costs by organizing data better across different locations, cutting down on how often data needs to be moved, and managing computing tasks more evenly.

METHOD

A total of 60 ML tests were done. Each test used a different block size (4MB, 8MB, or 16MB) and one of five ways to create a team model. The data was divided into two parts: 75% for training and 25% for testing. The training data was placed in the CEDEs system, and the testing data was kept aside to check how well the Ensemble models work later

CEDEs was set up using one of the tested block sizes (4MB, 8MB, or 16MB). The data was then automatically split into blocks, shared out, and copied. In some situations, tasks to get data ready, like cleaning it, creating useful features, and changing it into a format for analysis, were done using Apache Spark. At this point, the aim wasn't to find the best model, but to look at and compare different methods.

Next, we used a tool called pymfe in Python to get important details about each block. These meta-features talk about different aspects like how data is spread out, how good the data is, and how much it changes.

A basic model was created for each section, and its performance scores (like RMSE, MAE, MSE, Logloss) were saved in a DB. Next, a K-means algorithm was used on the meta-features to group blocks that have similar qualities. This helped in choosing data based on how alike they are. K-means was used with k set to 5 to always create five groups of blocks.

For each block size, we trained five different Ensemble models using these methods to choose the base models:

H1: The Ensemble was created by using all the basic models. Each model's importance was based on its quality, measured by RMSE from 5-fold cross-validation, with poorer models having more influence.

H2: The Ensemble was created using all the basic models, giving the same importance to each one.

H3: We only chose the best half of the base models, and all the chosen models were given the same importance. The models were rated based on RMSE, which was calculated using 5-fold cross-validation.

H4: One random model was chosen from each of the five groups created by K-means. The goal was to get a variety of models that were trained on different types of data.

Five random models were chosen from the group with the most similarities, meaning they have the closest characteristics based on their features.

Even though CEDEs can work with different types of groups, all the groups in this study used Random Forests, which are made up of Decision Trees. All the Decision Trees were set up the same way: they had a maximum depth of 25, needed at least 2 samples to split, could have unlimited leaf nodes, and used a ccp_alpha value of 0.

In the end, each group was tested using a method called 5-fold cross-validation and also with a separate test set. The results were then looked at to compare the different methods

RESULT

In this study, we show the detailed results of our tests using two different sets of data: ImageNet and Criteo. For the ImageNet dataset, we use two different types of neural networks and a set of options for adjusting settings, which gives us 16 ways to train the models. In the same way, for the Criteo dataset, we perform a task focused only on adjusting the settings, which includes 16 training setups. The information about these settings is shown in Table 5. Using grid search to choose the best model is a common approach in deep learning and is still popular among experts in the field [6].

We compare different building methods, using MA as our main point of reference. Each method needs its own Extract, Transform, Load (ETL) steps, customized to fit the different data location issues that the datasets have. For UDAF, CTQ, and MA, ETL includes preparing the data in the DB so it is organized into byte arrays and buffers. This makes it easier for User-Defined Aggregate Functions (UDAFs) to use the data efficiently. In addition to the steps mentioned earlier, the ETL process for DA also includes accessing tables and special files, turning the data back into its original form, and putting the data into the main memory [7].

In the Cerebro-Spark setup, the ETL process is bigger because it includes getting data from the DB and doing extra steps to change the data formats. We use a special tool called gpfdist to export data, along with a custom program for the needed preparation work.

Our performance review looks at several areas: how well things come together, how fast they run, and how well they use resources and money. This includes measuring things like GPU/CPU use, memory usage, network speed, and disk reading/writing. how the performance improved during the ImageNet tests. All methods, except for MA, show similar progress towards the best solutions, just like how sequential Stochastic Gradient Descent (SGD) works. On

the other hand, MA has major problems getting to the same level as other methods, which makes it learn more slowly [2].

To keep it short, we haven't included the convergence curves for the Criteo dataset. However, it's important to mention that all methods, including MA, show very similar convergence patterns and reach about 99% accuracy quickly. This analysis shows how important it is for data to be stored close to where it is used in distributed DBs. It also explains how using ML can help improve where data is placed.

Table 1. ImageNet and Criteo datasets:

Approach	ETL Time	Exec. Time	Epoch Time	GPU Util.	GPU RAM Util.	CPU Util.	DRAM Util.	Total Network	Per Workload Disk R/W
ImageNet									
MA	2.8 hr	42.6 hr	4.3 hr	56.8%	32.5%	2.3%	3.1%	0.9 TB	12 GB / 2 GB
UDAF	2.8 hr	48.5 hr	4.9 hr	49.9%	28.6%	2.2%	5.6%	0.8 TB	12 GB / 279 GB
CTQ	2.8 hr	45.1 hr	4.5 hr	56.2%	32.2%	2.5%	1.9%	0.6 TB	12 GB / 152 GB
DA-Cerebro	5.4 hr	23.0 hr	2.3 hr	70.5%	42.5%	2.8%	20.2%	0.6 TB	0.6 GB / 0.3 GB
Cerebro-Spark	4.4 hr	23.9 hr	2.4 hr	65.1%	36.5%	11.2%	17.4%	1.1 TB	0.2 GB / 2 GB
PyTDDP	4.4 hr	77.3 hr	7.7 hr	97.1%	13.1%	8.1%	14.7%	1900 TB	None / 11 GB
DA-PyTDDP	5.4 hr	77.5 hr	7.8 hr	96.8%	13.2%	8.2%	21.1%	1900 TB	None / 1 GB
Criteo									
MA	8.6 hr	38.5 hr	7.7 hr	N/A	N/A	44.1%	2.3%	0.1 TB	1 GB / 2 GB
UDAF	8.6 hr	62.0 hr	12.4 hr	N/A	N/A	27.1%	2.3%	0.1 TB	1 GB / 38 GB
CTQ	8.6 hr	40.0 hr	8.0 hr	N/A	N/A	41.0%	1.9%	0.08 TB	1 GB / 22 GB
DA-Cerebro	10.5 hr	21.5 hr	4.3 hr	N/A	N/A	37.4%	28.5%	0.07 TB	0.2 GB / 0.3 GB
Cerebro-Spark	8.3 hr	22.5 hr	4.5 hr	N/A	N/A	35.2%	28.5%	0.2 TB	0.2 GB / 1 GB

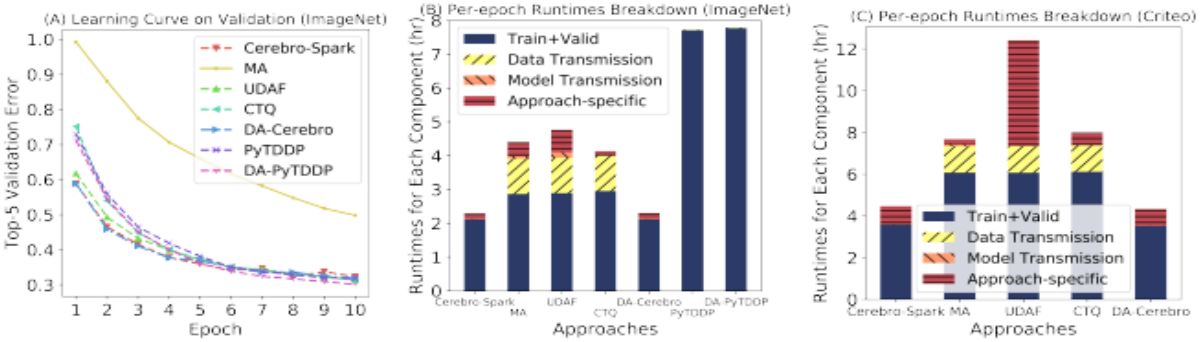

Figure 4. Results of complete tests. (A): How things come together on ImageNet. (B): A summary of how long each run took for every period. Method for ImageNet. (C): Time taken for each method on Criteo during each training round

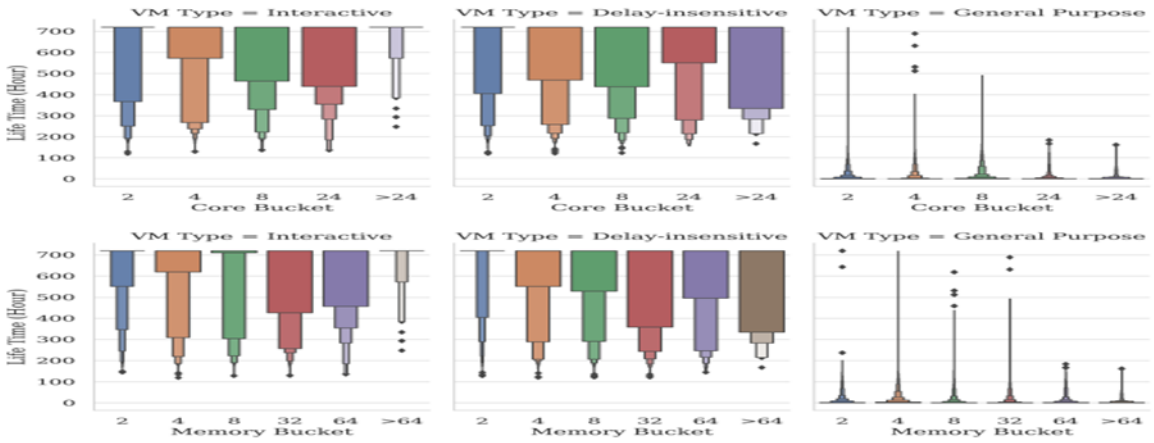

Figure 5. "VM Lifetime Across Core and Memory Buckets"

In expansion to the outfits prepared taking after the strategy laid out within the past segment, a standard comparison was performed utilizing Irregular Woodland models prepared on each dataset in its aggregate. The execution of these Arbitrary Timberland models was assessed utilizing Root Cruel Square Blunder (RMSE) through 5-fold cross-validation. Be that as it may, straightforwardly comparing RMSE values over different models and datasets isn't perfect due to the inalienable reliance of RMSE on the scale of the subordinate variable. To guarantee important and generalizable comparisons, we normalized the RMSE by isolating it by the run of the subordinate variable's exception limits. These exception limits were calculated utilizing the 1.5 interquartile run (IQR) run the show, which gives a more standardized degree of blunder relative to the spread of the information, permitting us to translate the mistake as a rate of the subordinate variable's run. Upon analyzing the comes about displayed in Tables 1, a common slant rises: as the piece estimate increments, the blunder diminishes. This perception highlights the significance of square estimate in disseminated DBs, as bigger pieces tend to diminish blunder but at the fetched of restricting the degree of parallelism due to less accessible hubs. Subsequently, deciding the ideal square measure is basic to accomplishing a adjust between maximizing parallel handling and minimizing blunder measurements in ML models. in specific, postured the most noteworthy challenge for both the custom gathering models and the standard Arbitrary Timberland show due to its characteristics as a time-series dataset. Time-series determining ordinarily includes complexities like regularity, patterns, and patterned varieties, which are regularly superior suited for specialized models such as ARIMA or LSTM. These challenges were reflected within the execution of the models. Interests, whereas analyzing the relationship between cross-validation blunders and test mistakes, most datasets shown progressed generalization with expanding square size evidenced by more grounded relationships. Be that as it may, the Covid dataset appeared the inverse slant, proposing interesting challenges in dealing with time-series information in conveyed situations

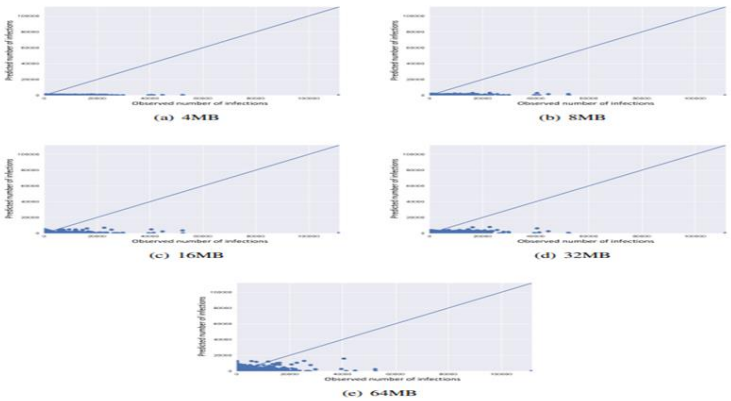

Figure 6.Observed vs. Predicted Infections Across Different Memory Sizes

In terms of demonstrate execution, the pattern Irregular Timberland prepared utilizing the H2O stage accomplished an mistake of 6.68% of the run of subordinate variable values. On the other hand, the gathering frameworks yielded

marginally higher blunders, extending from 7.4% (H1, 64 MB piece estimate) to 8.02% (H3, 4 MB square measure).Whereas the pattern Arbitrary Woodland marginally outflanked our custom gathering models in each situation, the contrasts were moderately little, proposing that the outfit approach remains competitive. For the remaining datasets, both the Irregular Woodland and the outfit models performed way better generally, showing that our information situation methodology, affected by ML models, has the potential to optimize execution, especially when the information territory is fittingly considered.

DISCUSSION

This dissertation presents a shared system for ML applications and shows its unique features. One important part of this system is that it can quickly put together different models depending on which base models are available and how the cluster is doing, making it flexible and able to change easily. The system can change its group of models over time by adding or removing them. This is made easier by a new tool that is being developed. The main goal of this research was to study how block size and different strategies affect the quality and performance of group models. The study wanted to find out if a certain method works better than others and to figure out the best block size that gives a good mix of accuracy and speed in distributed DBs. Three methods were tested. Of these, H3 showed the best results, usually making fewer mistakes. H3 works by choosing the best half of the available base models and giving each one the same importance. The results show that smaller, focused groups of models can do better than larger, complicated ones because weak models can lower the overall performance. This idea shows a straightforward way to manage base models in the system more effectively. We can take out models that are not working well from the system's storage (called HDFS), which helps free up important resources and makes the whole system run better. A set of data collected over time that created special difficulties for the computer programs used in this research. Because predicting time series data can be complicated, special models like ARIMA or LSTM usually work better. So, the unusual finding with the Covid data doesn't affect how good H3 is for the other datasets used in this study. Our study shows that using a ML approach is very helpful for improving how data is stored in distributed DBs. By looking at how people search and what features are related, our model helps to lower data transfer and speed up response times, which improves the system's performance. Data locality is very important for making distributed DBs work well. When data is kept near where it is needed, it takes much less time to get results because there is less delay from moving the data around. Our results show that the ML model can find the best storage places for each record. It groups related data together and matches it with expected search requests. This smart way of organizing data reduces how far it has to go and takes advantage of the data's natural setup, making it faster to access [8].

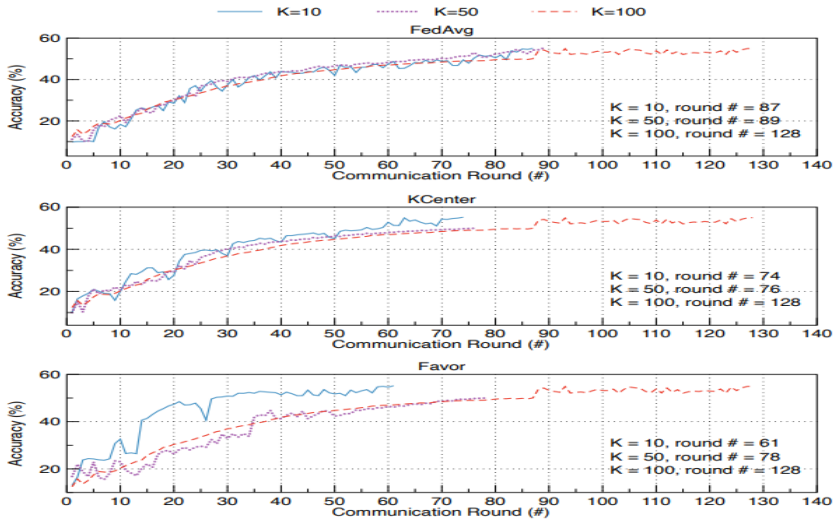

Figure 7. Model Evauation

Our successful ML strategy works well because the model learns from past data and adjusts to changing search trends. The model uses information about both the questions being asked and the current state of the DB to make smart choices about where to store data. The model can adjust itself based on changing workloads, making sure the most important data is easy to access without delays. [9] Our ML method is more flexible and adaptable than regular data placement methods, which usually follow fixed rules or guidelines. Old methods can have a hard time keeping up

with the changing nature of data requests, which can make accessing data less efficient. In comparison, our method makes it easier to find data and works well even when there is more data and more complicated questions. Our results mean more than just better performance. By using ML in how data is stored, DB managers can get helpful information automatically to make better choices about how to set up their DBs, organize data, and manage resources. This move towards using data to guide decisions could lead to big improvements in how distributed DBs are designed, creating a system that focuses on being efficient and quick to respond.

CONCLUSION

This study shows how ML can improve how data is stored in different DBs. Our results show that using a ML method greatly improves where data is stored. This leads to less data movement and faster responses to queries. By using past search habits and how different data points relate to each other, the ML model smartly finds the best storage places for each record. This makes sure that related data is kept nearby for easy access. This approach has many benefits. It makes finding data faster and adjusts automatically to different types of queries, doing better than older methods that don't change. This move to a data-based approach helps DB managers get automatic information, making it easier for them to make smarter choices about how to organize data and use resources. In the future, there are many chances to improve these ML models, test new techniques, and see how well they work in different settings. As we learn more about using ML, we see that it can help us create better, bigger, and quicker ways to manage data.

REFERENCES

[1] Akita, R., Yoshihara, A., Matsubara, T., and Uehara, K., "Deep learning for stock prediction using numerical and textual information," In ICIS, IEEE Computer Society, pp. 1–6, 2016.

[2] Amazon, "RedShift Query Planning and Execution Workflow," Accessed November 19, 2020. Retrieved from https://docs.aws.amazon.com/redshift/latest/dg/c-queryplanning.html.

[3] Anil, R., Çapan, G., Drost-Fromm, I., Dunning, T., Friedman, E., Grant, T., Quinn, S., Ranjan, P., Schelter, S., and Yilmazel, Ö., "Apache Mahout: Machine Learning on Distributed Dataflow Systems," Journal of Machine Learning Research, vol. 21, no. 127, pp. 1–127, 2020.

[4] Atkinson, M. P., Bancilhon, F., DeWitt, D. J., Dittrich, K. R., Maier, D., and Zdonik, S. B., "The Object-Oriented Database System Manifesto. In DOOD, North-Holland/Elsevier Science Publishers, pp. 223–240, 1989.

[5] Bengio, Y., "Rmsprop and equilibrated adaptive learning rates for nonconvex optimization," arXiv preprint arXiv:1502.04390, 2015.

[6] Bergstra, J., Yamins, D., and Cox, D., "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures," In ICML (1), vol. 28 of JMLR Workshop and Conference Proceedings, pp. 115–123, 2013. JMLR.org.

[7] Boehm, M., Dusenberry, M., Eriksson, D., Evfimievski, A. V., Manshadi, F. M., Pansare, N., Reinwald, B., Reiss, F., Sen, P., Surve, A., and Tatikonda, S., "SystemML: Declarative Machine Learning on Spark," Proceedings of the VLDB Endowment, vol. 9, no. 13, pp. 1425–1436, 2016.

[8] Boehm, M., Reinwald, B., Hutchison, D., Sen, P., Evfimievski, A. V., and Pansare, N., "On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML," Proceedings of the VLDB Endowment, vol. 11, no. 12, pp.1755–1768, 2018.

[9] Jasny, M., Ziegler, T., Kraska, T., Röhm, U., and Binnig, C., "DB4ML - An In-Memory Database Kernel with Machine Learning Support," In SIGMOD Conference, pp. 159–173, ACM, 2020.

[10] Kaggle, "Kaggle Survey 2020," Accessed March 13, 2021. Retrieved from https://www.kaggle.com/kaggle-survey-2020

[11] Kaggle, "State of Data Science and Machine Learning 2019," Accessed October 31, 2020. Retrieved from https://www.kaggle.com/kaggle-survey-2019.

[12] Karanasos, K., Interlandi, M., Psallidas, F., Sen, R., Park, K., Popivanov, I., Xin, D., Nakandala, S., Krishnan, S., Weimer, M., Yu, Y., Ramakrishnan, R., and Curino, C., "Extending Relational Query Processing with ML Inference," In CIDR. Retrieved from www.cidrdb.org, 2020.

[13] Khamis, M. A., Ngo, H. Q., Nguyen, X., Olteanu, D., and Schleich, M., "In-Database Learning with Sparse Tensors," In PODS, pp. 325–340. ACM, 2018.

[14] Kim, M., and Candan, K. S., "Efficient Static and Dynamic In-Database Tensor Decompositions on Chunk-Based Array Stores," In CIKM, ACM, 2014.

[15] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980, pp. 969–978, 2014.

[16] Koliousis, A., Watcharapichat, P., Weidlich, M., Mai, L., Costa, P., and Pietzuch, P., "Crossbow: Scaling Deep Learning with Small Batch Sizes on Multi-GPU Servers," Proceedings of the VLDB Endowment, vol. 12, no. 11, pp. 1399–1412, 2019,

[17] Bouaziz, S., Nabli, A., and Gargouri, F., "Design a Data Warehouse Schema from Document-Oriented Database," Procedia Computer Science, vol. 159, pp. 221–230, 2019, doi:10.1016/j.procs.2019.09.177.

[18] Chevalier, M., El Malki, M., Kopliku, A., Teste, O., and Tournier, R., "Benchmark for OLAP on NoSQL Technologies Comparing NoSQL Multidimensional Data Warehousing Solutions," In Proceedings of the International Conference on Research Challenges in Information Science, pp. 480–485, 2015, doi:10.1109/RCIS.2015.7128909.

[19] Dehdouh, K., Bentayeb, F., Boussaid, O., and Kabachi, N., "Using the Column-Oriented NoSQL Model for Implementing Big Data Warehouses," In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'15), pp. 469–475, 2015.

[20] Dehdouh, K., Boussaid, O., and Bentayeb, F., "Big Data Warehouse: Building Columnar NoSQL OLAP Cubes," International Journal of Decision Support System Technology, vol. 12, no. 1, pp. 1–24, 2020, doi:10.4018/IJDSST.2020010101.

[21] Khalil, A., and Belaissaoui, M., "A Graph-Oriented Framework for Online Analytical Processing," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 13, no. 5, pp. 547–555, 2022, doi:10.14569/IJACSA.2022.0130564.

[22] Oditis, I., Bicevska, Z., Bicevskis, J., and Karnitis, G. "Implementation of NoSQL-Based Data Warehouses," BJMC, vol. 6, no. 1, pp. 45–55, 2018, doi:10.22364/bjmc.2018.6.1.04.