Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

A Survey on Database Index Tuning and Defragmentation

Mounicasri Valavala
University of the Cumberlands
Williamsburg, Kentucky, USA

Dr. Wasim Alhamdani
University of the Cumberlands
Williamsburg, Kentucky, USA

Abstract:- Database performance tuning is an evolutionary field and is considered a difficult task, especially with large databases. Indexing contributes largely to improve the Database performance by reducing the query response time. The two main areas of indexing are selecting and maintaining the index. Selecting the right index by balancing the cost of data retrieval/manipulation and maintaining the index without fragmentation are challenging tasks with growing data requirements. There is extensive research to automate index tuning and limited research in index defragmentation. This paper provides a survey of research in index tuning and defragmentation techniques proposed in the last decade and the need for using robust technology like Machine Learning (ML) to overcome the limitations of the existing research.

Keywords— Database Performance; Defragmentation, Fragmentation; Index Tuning; Machine Learning;

I. INTRODUCTION

The drastic growth in the Database size in today's environment made performance maintenance necessary to enhance and sustain the Database performance. Database performance tuning lowers the query response time using CPU, disk I/O, and memory efficiently [1]. The different ways to improve the performance are tuning the API that hits the Database Management System (DBMS), altering hardware configurations, index tuning, and partitioning. The researches prove that index tuning plays a vital role in improving query performance [2].

Indexing helps to get swift access to the data requested by the query, acting like a big booster to the performance [3]. In the absence of an index, a full table scan scenario involving every row of a table occurs, resulting in Database slowdown, especially for Large Datasets. It is necessary to maintain a balanced index coverage to avoid any penalties for write-heavy operations. Due to the high importance of indexing, many researchers proposed indexing techniques that perform index selection automatically. The recent research avenues in indexing are to select index using Machine Learning (ML) Models and replace indexing with learned index concept built using hierarchical ML models.

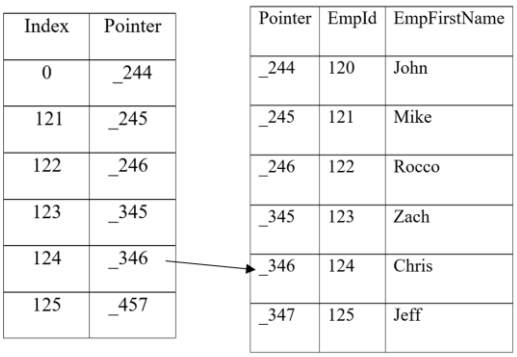

Figure 1: Data Access Using Index

Index maintenance comes into the picture once the index selection and materialization completes. One of the crucial tasks in index maintenance is index fragmentation, where the logical and physical order of data goes out of synch. Database professionals use defragmentation techniques, such as rebuild and reorganize, to handle this scenario. There is limited research towards automating the defragmentation techniques.

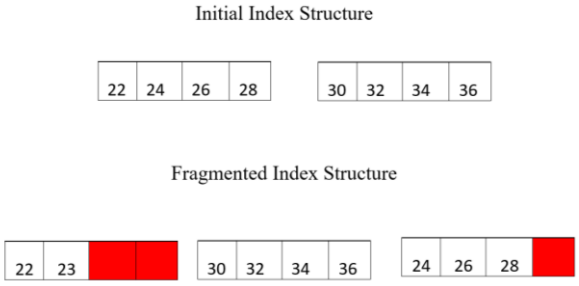

Figure 2: Index Fragmentation

This paper presents the existing research in index tuning, methodologies, and limitations of the proposed models. The next focus area is defragmentation, existing defragmentation techniques, and limitations. We also present our views on the need for a robust model to handle the changing query workloads.

II. INDEX TUNING

Index tuning experienced extensive research in the last decade, where most of the researchers focused on performing index selection using the Cost-Based model. In recent years there is a breakthrough to use ML for indexing. In addition to the traditional index access patterns, recent researches proposed learned indexes where the index is treated as a

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

model to find the required data [4, 5]. We focus our survey on the traditional index patterns, as there is limited to no usage of the learned index in real-time.

A. *Literature Survey*

Index tuning is an essential field in Database tuning. As a result, there is continuous research to develop automated indexing techniques. We present the proposed index tuning models in the last decade, and the focus is mainly on the methodology and limitations of the models.

The index tuning model, named CoPhy, proposed in [6], uses the linear optimization technique to perform index tuning for large workloads. It uses off-the-shelf Binary Integer Program (BIP) and linear programming techniques to support fast re-tuning, early termination, and soft constraints. The authors followed a greedy approach to improve scalability, especially when dealing with large workloads. However, it is unacceptable for a commercial tool to expose massive workloads to index advisor for a longer duration [7].

Schnaitter & Polyzotis (2012) proposed a semi-automatic model based on Database Administrator's (DBA) feedback to create index recommendations [8]. The authors created the framework with three components: Work Function Algorithm (WFA) to create index recommendation, feedback mechanism to incorporate DBA's feedback, and an online algorithm to select the candidate indexes. The model is not entirely automated and waits for the DBA's interference to finalize the index recommendation [9]. As a result, DBAs experience is a key to decide whether to materialize the index or not, which is expensive to the organization [10].

A semi-automatic index model named Kaizen recommends indexes and takes DBA's feedback on the recommended indexes [11]. Kaizen is capable of integrating with any DBMS that supports "What-If" plans, making it portable across different platforms. It follows the divide and conquer mode and relies on the cost of work done by the DBMS. Kaizen relies on past queries and feedback given by the DBAs and does not consider future workloads [12]. It uses "What-If" plans with all the possible candidate indexes, keeping an overhead on the optimizer.

Boronski & Bocewiz (2013) proposed an index tuning that uses a group of queries to perform index selection [13]. The authors used a genetic algorithm to select the efficient indexes out of the index search space created using grouped queries. The model is limited to work with grouped queries and is not applicable if the queries are independent of each other.

The same authors also proposed another model to create index recommendations using a query group, with the restriction that the table should not have any indexes [14]. The model calculates the query group's execution time for an index subset and compares it with the remaining subsets. It creates the index recommendations for the hypothetical indexes in the index set, resulting in the least execution time [14]. The model's drawback is that it needs multiple iterations to finalize the recommendations.

The model proposed by Sharma, Schuhknecht, & Dittrich (2018) creates index recommendations by mapping the index selection problem to a Deep RL model [15]. The combination of the workload and the current index configuration serves as an input to the model. The authors used episodic RL with index creation as the action and calculated the reward using the workload's execution cost with and without the indexes. The model is not an efficient solution for the production environment that encounter unseen data [16].

The index generation model for Microsoft Azure SQL Database contains the Control plane, Index recommender, and Validator [17]. The model applies "What-If" plans on the candidate indexes for 'k' low-performing queries. The Control plane coordinates with other system components, the index recommender creates index recommendations, and the validator checks the performance impact due to index change. The model cannot provide coverage to all exceptional cases across workloads [17].

Ding, Das, Marcus, Wu, Chaudhuri, & Narasayya (2019) proposed a model to recommend indexes using a classification model to calculate the query plans' performance gain [18]. The authors performed mathematical transformations on the feature vectors constructed from the query plans, parallelism, and execution mode. The different classification models used are Random Forest (RF), Deep Neural Networks (DNN), and Hybrid Neural Network to check a query plan's performance. The model needs a sample workload for training and requires human intervention to materialize index recommendation and tuning sessions [19].

The SmartIX model creates index recommendations using RL, with Markov Decision Process (MDP) as the RL agent [20]. The model contains an RL agent for decision-making, an environment to calculate the transitions, and a DBMS interface to apply the index recommendations. This model's drawback is that RL takes more time to converge and expensive as it needs to visit all possible states.

B. *Discussion*

The existing research in index tuning falls into two categories: semi-automatic, where the model requires DBA's intervention, and automatic, where the model recommends and materializes the indexes. The research widely used "What-If" plans and cost-based models to create index recommendations until a couple of years ago. The core principle behind these researches until 2018 is to retrieve the query plans for candidate indexes and calculate the plans' execution cost. The models create index recommendations for the candidate index, whose query plan is of minimum cost.

Though the core principle is the same, the approach to use this principle varied across different models. For instance, the model proposed in [13] for index tuning uses the query group with mutually dependent queries. The models in [8, 11] use DBA's feedback on the index recommendations to improve model accuracy. All the researches work is based on the current workload and do not consider the future workload variations. As a result, the selected index may or may not adapt to future workloads.

The research in index tuning opened new avenues to use promising ML technologies to perform index selection. The models proposed in [15, 18, 20] uses RL and achieved promising results. However, the DBAs need to provide the sample workload, and RL takes a long time to converge, making these techniques consume a longer duration for massive workloads. Below is the share of ML and Non-ML technologies usage in index tuning.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

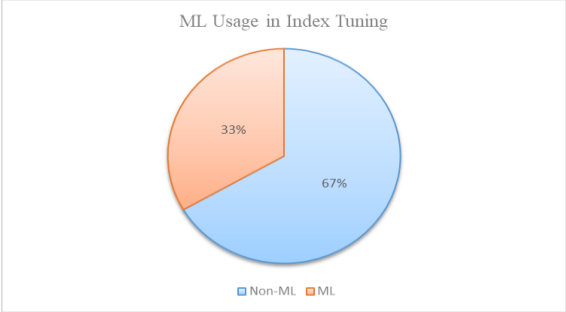

Figure 3: Share of Index Tuning Techniques using ML and Non-ML Methods

C. *Research Findings*

We analyzed the methodologies in each of the proposed models and recognized the need to look at index tuning from a different perspective. The existing models look at index tuning as a problem for the current query workload. Instead, one should consider it as a problem for current and future workloads. Below are our findings and solutions for the limitations in the existing research.

- Lack of index adaptability to future query workloads, as the future index usage is not part of the existing models. Using future workload forecast as part of the model, creates a highly efficient index-tuning advisor.
- The models' speed is an issue to use some of the existing models in commercials tools. Hence, it is necessary to create models with a reasonable speed, mainly if the index selection happens online.
- The models presented in [13, 14] use a dependent query group to perform index selection. However, it keeps an extra overhead on the DBA to provide the query group for different tables in the Database, making it inefficient in terms of resource consumption. As a result, a generic model is necessary to work with all the queries irrespective of dependency properties.
- The recent research also used RL to perform index tuning. However, RL takes long durations to converge, making it unfit to be an online model. Replacing RL with other ML techniques can yield better results for an online model.
- A broad spectrum of existing work use "What-If" plans, making the index tuning expensive. Hence, eliminating their usage is necessary to improve the model efficiency.

We argue that the solution to limitations in existing research is to use supervised ML classification algorithms that give a response on whether to index a column or not. Dataset construction is a crucial stage to use supervised ML models. The dataset should contain the features to include future index usage, query usage rate, actual execution plan, table size, and DML operations count on the predicate column under investigation.

This model needs DBA's expertise to create the training set, and once trained, the model can provide the index selection without any external intervention. Any new index creation can be sent as feedback to the model to improvise the performance. Using Random Forest (RF) algorithm, which can work with different dataset sizes, suits the index selection model. The reasons for choosing RF are its' ability to overcome overfitting, handle unbalanced data efficiently, and perform fault diagnosis [21, 22]. We will present the design and results of this model in the later articles in this series.

III. INDEX DEFRAGMENTATION

The index is a logical structure that contains pointers to the physical location of the record. If there is data update or insert operations on the indexed column, the logical order will go out of synch with the physical order. This scenario of mismatch between logical and physical pages is called fragmentation. The DBMS experience delays as it needs more I/O to access the requested data.

The two types of fragmentation are internal and external fragmentation. The query hit that split the index page causes internal fragmentation [23, 24]. The query hit that disturbs the existing logical index page order causes external fragmentation [23, 24]. Defragmentation compacts and reorders the index pages to mitigate internal and external fragmentation, respectively. The DBAs use fragmentation statistics to determine the need for defragmentation.

Rebuilding and Reorganizing are the two defragmentation techniques used to counter fragmentation. The table size and fragmentation level determine the suitable type of defragmentation technique. Rebuilding index deletes the index and recreates it anew. It is a heavy process and is suitable for high fragmentation levels, especially on larger tables [24]. Reorganizing the index reorders the index pages and is a light-weight operation, suitable for low fragmentation levels [23].

A. *Literature Survey*

Though defragmentation is one of the critical areas that help to use the indexes to the fullest capacity, it received significantly less attention from the research community. It is imperative to address the challenges in this area, to take away the manual effort Database teams keep to do fragmentation analysis and defragmentation. We discuss the research contribution and their corresponding limitations in this field.

The proposed workload driven defragmentation model overcomes the data-driven approach's limitations [24]. The limitations listed by the authors are granular defragmentation levels and defragmenting without considering index usage. The proposed model uses a range-level driven index and workload usage rate to determine the need for fragmentation [24].

The range-level index gives the highly used data range using the "What-If" API, and the workload usage rate gives the number of queries accessing the data [24]. This approach's limitations are the need for domain knowledge and the usage of "What-IF" APIs that often result in extra cost. The authors also always used the heavy-weight rebuilding technique irrespective of the fragmentation levels, keeping an overhead on the DBMS.

The Autonomous Re-indexing model proposed in [25] uses heuristics to identify the indexes that need defragmentation. The authors considered fragmentation level, index scan count, and index size to make a defragmentation

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

decision. This model alleviates the manual effort of DBAs in defragmentation decision-making. This research's drawback is that the defragmentation always happens online, resulting in system slowdown when done for large tables or highly-fragmented data.

B. Discussion

There is minimal research in automatic index defragmentation, and existing work also uses "What-If" APIs and always uses rebuild technique irrespective of fragmentation level and table size. Choosing the defragmentation technique is crucial to performance, mainly when the operation works online. Rebuilding is a heavy operation and slows downs the system when used during work hours. Hence, the Database teams schedule the index rebuilding on a non-work day. The model in [24, 25] considers index usage, which is a crucial factor limiting unnecessary defragmentation for an index with minimal usage. However, using the "What-IF" API for this purpose adds overhead on the DBMS.

C. Research Findings

Index fragmentation results in serious performance issues, even if the table has the most efficient indexes. Hence, index defragmentation needs more attention. Below are the current model's limitations and solutions to make the defragmentation autonomous.

- The existing methods [24, 25] always use rebuild operation whenever defragmentation is required. Applying the right type of defragmentation is very much necessary, mainly when it happens in an online mode.
- The model in [24] uses "What-If" plans, keeping an overhead on the DBMS. Hence creating a model without using the "What-If" plans is necessary.

The problem of defragmentation is a multi-class classification problem with three output labels. The output labels are "No Defragmentation," "Rebuild," and "Reorganize." Defragmentation is similar to diabetes prediction, student grade prediction, and other classification problems solved using Machine learning. Hence, adopting ML techniques to identify the defragmentation technique provides an automated and efficient solution.

The challenge with implementing ML is the training dataset construction, as the fragmentation level standards followed may vary from one domain to another. However, DBAs expertise can solve this problem of data collection. ML is already in use in Database knob configuration management and index tuning and has proven to be an efficient solution. We argue that it is necessary to explore the use of ML to create an efficient autonomous defragmentation technique.

We will use the RF model for defragmentation, as the dataset will be small initially, which will grow over time. RF has robust capabilities to handle overfitting and anomalies and is an efficient model for innovation-friendly environments [21, 22]. We will present the dataset collection, implementation details, and results in later articles.

IV. CONCLUSION

The survey presented the research in index tuning and defragmentation in the last decade. There is exhaustive research in index tuning and limited research in defragmentation. We discussed the index tuning models' limitations, and few models are not suitable for commercial tools due to the required resources. The typical limitations are the usage of expensive "What-If" APIs, slow converging RL models, and lack of adaptability to future workloads.

The research in defragmentation used "What-If" APIs and heuristics and always applied rebuild technique irrespective of fragmentation statistics, leading to performance degradation. As a result, creating a model to decide on the defragmentation technique should use fragmentation level, index usage rate, and table size. We argue that using ML classification techniques for both index tuning and defragmentation can yield better results, and constructing the dataset to include the future usage rates makes the model adaptable to future workloads.

REFERENCES

[1] Kamatkar, S. J., Kamble, A., Viloria, A., Hernández-Fernandez, L., & Cali, E. G. (2018). Database Performance Tuning and Query Optimization. Data Mining and Big Data, 3–11. https://doi.org/10.1007/978-3-319-93803-5_1

[2] Qi, C. (2016). On index-based query in SQL Server Database. 2016 35th Chinese Control Conference (CCC), 9519–9523. https://doi.org/10.1109/chicc.2016.7554868

[3] Mukherjee, S. (2019). Indexes in Microsoft SQL Server. SSRN Electronic Journal, 199. https://doi.org/10.2139/ssrn.3415957

[4] Ding, J., Minhas, U. F., Yu, J., Wang, C., Do, J., Li, Y., Zhang, H., Chandramouli, B., Gehrke, J., Kossmann, D., Lomet, D., & Kraska, T. (2020). ALEX: An Updatable Adaptive Learned Index. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 969–984. https://doi.org/10.1145/3318464.3389711

[5] Kraska, T., Alizadeh, M., Beutel, A., Chi, E.H., Kristo, A., Leclerc, G., Madden, S., Mao, H., & Nathan, V. (2019). SageDB: A Learned Database System. CIDR. http://cidrdb.org/cidr2019/papers/p117-kraska-cidr19.pdf

[6] Dash, D., Polyzotis, N., & Ailamaki, A. (2011). CoPhy. Proceedings of the VLDB Endowment, 4(6), 362–372. https://doi.org/10.14778/1978665.1978668

[7] Kimura, H., Coffrin, C., Rasin, A., & Zdonik, S. B. (2012). Optimizing index deployment order for evolving OLAP. Proceedings of the 15th International Conference on Extending Database Technology - EDBT '12, 276–287. https://doi.org/10.1145/2247596.2247630

[8] Schnaitter, K., & Polyzotis, N. (2012). Semi-automatic index tuning. Proceedings of the VLDB Endowment, 5(5), 478–489. https://doi.org/10.14778/2140436.2140444

[9] Subotić, P., Jordan, H., Chang, L., Fekete, A., & Scholz, B. (2018). Automatic index selection for large-scale datalog computation. Proceedings of the VLDB Endowment, 12(2), 141–153. https://doi.org/10.14778/3282495.3282500

[10] Li, L., & Gruenwald, L. (2013). Self-managing online partitioner for databases (SMOPD). Proceedings of the 17th International Database Engineering & Applications Symposium on - IDEAS '13, 168–173. https://doi.org/10.1145/2513591.2513649

[11] Jimenez, I., Sanchez, H., Tran, Q. T., & Polyzotis, N. (2012). Kaizen. Proceedings of the 2012 International Conference on Management of Data - SIGMOD '12, 666. https://doi.org/10.1145/2213836.2213932

[12] Naik, S. (2017). TIER: Table index evaluator and recommender — A proposed model to improve transaction performance in distributed heterogeneous Database. 2017 International Conference on Soft Computing and Its Engineering Applications (IcSoftComp), 1–8. https://doi.org/10.1109/icsoftcomp.2017.8280085

[13] Boroński, R., & Bocewicz, G. (2013). Indexes driven mechanism for grouped SQL queries.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BSW1-0109-0009/c/Boronski.pdf

[14] Boronski, R., & Bocewicz, G. (2014). Relational Database Index Selection Algorithm. Computer Networks, 338–347. https://doi.org/10.1007/978-3-319-07941-7_34

[15] Sharma, A., Schuhknecht, F., & Dittrich, J. (2018). The Case for Automatic Database Administration using Deep Reinforcement Learning. ArXiv. https://arxiv.org/abs/1801. 05643

[16] Ma, L., Ding, B., Das, S., & Swaminathan, A. (2020). Active Learning for ML Enhanced Database Systems. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 175–191. https://doi.org/10.1145/3318464.3389768

[17] Das, S., Chaudhuri, S., Grbic, M., Ilic, I., Jovandic, I., Jovanovic, A., Narasayya, V. R., Radulovic, M., Stikic, M., & Xu, G. (2019). Automatically Indexing Millions of Databases in Microsoft Azure SQL Database. Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19, 666–679. https://doi.org/10.1145/3299869.3314035

[18] Ding, B., Das, S., Marcus, R., Wu, W., Chaudhuri, S., & Narasayya, V. R. (2019). AI Meets AI. Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19, 1241–1258. https://doi.org/10.1145/3299869.3324957

[19] Sadri, Z., Gruenwald, L., & Lead, E. (2020). DRLindex. Proceedings of the 24th Symposium on International Database Engineering & Applications, 1–8. https://doi.org/10.1145/3410566.3410603

[20] Licks, G.P., & Meneguzzi, F. (2020). Automated Database Indexing using Model-free Reinforcement Learning. *ArXiv*. https://arxiv.org/abs/2007.14244

[21] Couronné, R., Probst, P., & Boulesteix, A.-L. (2018). Random forest versus logistic regression: a large-scale benchmark experiment. BMC Bioinformatics, 19(1), 567. https://doi.org/10.1186/s12859-018-2264-5

[22] Liaw, A. & Wiener, M. Classification, and regression by randomForest (2002). R News 2, 18–22

[23] Cioloca, C., & Georgescu, M. (2011). Increasing Database Performance using Indexes. Database Systems Journal, 13–22. https://pdfs.semanticscholar.org/1b96/91b7957ec3418c2b088177940de07483ee21.pdf

[24] Narasayya, V., & Syamala, M. (2010). Workload driven index defragmentation. 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), 497–508. https://doi.org/10.1109/icde.2010.5447889

[25] Morelli, E., Almeida, A., Lifschitz, S., Monteiro, J. M., & Machado, J. (2012). Autonomous re-indexing. Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12, 893–897. https://doi.org/10.1145/2245276.2245450