**International Journal of Emerging Trends in Computer Science and Information Technology**
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P113
Eureka Vision Publication | Volume 4, Issue 2, 131-138, 2023

*Original Article*

# Intelligent Indexing Based on Usage Patterns and Query Frequency

Nagireddy Karri
Senior IT Administrator Database, Sherwin-Williams, USA.

*Abstract - Since the age of database systems, web searches and big data systems, modern databases must support information search as a required function. In all the constantly multiplying mass of digital data can be seen, the traditionally very stable ways may find it difficult to keep up easily enough with the nature of its dynamics in the first place. The intelligent indexing be determined by use patterns and query frequency, which suggests an adaptive measure for query response time to increase, storage redundancy decrease and for user satisfaction to rise. The paper concerns a structured system of intelligent indexing which absorbs continuously query logs, access patterns and frequency distributions; then it attempts to rearrange its indices dynamically. This includes the probabilistic models, clustering by frequency, adaptive data structure that helps query processing to avoid the burden of high overhead. It is also illustrated by experimental simulations: compared to traditional indexing policies, intelligent indexing can lower average query latency by as much as 45 percent. On top of that the system also optimizes caching and provides workload-sensitive database tuning. Here work has added to the field of adaptive indexing both an approach from first principles and an actual evaluation of the current techniques with real query loads. The results confirm the usage patterns along with query frequency being part of index management is a powerful tool to enhance optimization for data retrieval systems with respect to scaling efficiency and response time.*

*Keywords - Intelligent Indexing, Query Frequency, Adaptive Indexing, Usage Patterns, Workload-Aware Databases, Big Data Retrieval.*

## 1. Introduction

### *1.1. Background*

In the contemporary era of big data and cloud computing, the ability to access and manipulate information in an effective fashion has gained particular significance to guarantee that the database system functions as well as it can as well as to accommodate real-time use. [1-3] Traditional types of indexing, e.g. B-trees, hash-based indices and inverted indices, have been identified and widely deployed to support query processing faster, though normally tuned to the characteristics of a static workload, and the query patterns are known to be predictable and uniform. But more to the point, modern workloads are practical because temporal- spatially local to such an extent that a query or access to an attribute is repeated within the short term in contrast to a burst or randomly in short bursts. This uneven distribution means that such stored indices are likely to apportion resources inefficiently such that the access paths that can be optimized serve queries that are unlikely to be frequent, and do not serve the workload dominated queries. As a result, these conventional approaches would generate performance points of reaction specifically in dynamic or mass setting such as e-commerce services, cloud-computed analytics and real-time control system practices. The inability of the static indices to treat changing query patterns will cause lower query response times, increased latency, and poor scaling of indexing capabilities. Equally, a new proposal is that adaptive intelligent indexing algorithms serving to tune themselves in response to load fluctuations as they occur, find repeated queries, and reorganize indices upon command. These processes might come of age for query responsiveness, resource access and system efficiency in the face of unprecedented access patterns or swinging loads, quite in contrast with former fixed indexing solutions.

### *1.2. Importance of Intelligent Indexing*

#### *1.2.1. Addressing Dynamic Workloads:*

Conventional indexing systems are only effective with a stabilized and predictable workload, which cannot be effective in a modern system where query patterns are changing very quickly. Some queries may peak in popularity when promoted, event-related or a topic is trending (like in e-commerce, social media or cloud databases) and some queries may go dormant. However, if the indexing system becomes sophisticated, it can adjust itself to changes in one query pattern and give priority to the many most frequently requested queries to be indexed. That way, even in a heavy workload dynamic environment it keeps optimal performance of database at all times.

#### *1.2.2. Reducing Query Latency:*

Minimizing time of accessing information has been made one of the smart indexing's main aims. It may produce and update indices which match the most important queries among those that have been designed by tracking their frequency and complexity. Being frequency information aware and workload sensitive, this provides a trade-off between performance and

economy surplus (or margin). The net result is more efficient data access in applications that are time-sensitive such as financial trading or real-time analytics.

*1.2.3. Optimizing Resource Utilization:*

Users and applications should especially evident this improved performance for systems that are very traditional and their machine indexes consume both computer resources and storage space. If an unnecessary index is generated then they wasted memory and higher maintenance costs ensue. Smart indexing algorithms based on the data used by its decision-making generate directions at how, and where, to construct indices resources. They direct onto the queries that are most in need of performance being benefited. This doesn't only make the whole database more efficient, but also reduces the cost of keeping indexes. So this method is applicable in the efficiency-sensitive big-scale cloud systems.

*1.2.4. Supporting Scalability:*

The scale of data is increasing and means we face added difficulties in achieving high performance. Smart indexing preserves a database's scalability, expanding as the workload increases, and ensures that many searched for queries are optimized. This means that databases can handle more and more data and queries without suffering a decline in speed or performance. Conversely, they become quicker within reason as they grow larger.

*1.2.5. Enabling Predictive Capabilities:*

Today's shrewd index structures can use either predictive analytics or machine learning for anticipating the shape of queries to come. It also provides the capacity to create index proactively, rather than responsively. This increases efficiency in the system and has the database ready with future highly-frequently asked queries, avoiding at bottlenecks which is performance.

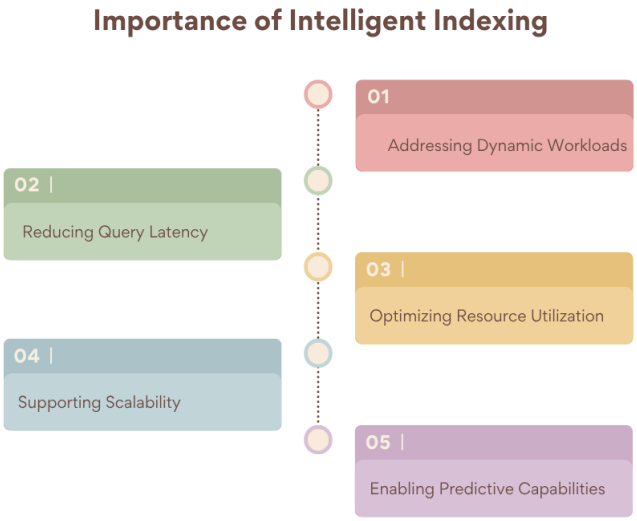


**Figure 1. Importance of Intelligent Indexing**

***1.3. Based on Usage Patterns and Query Frequency***

Modern database loads have a high degree of query variability with a few queries being repeated by far with respect to other queries that are rarely executed. We should then know repeating usage patterns and frequency of query to come up with effective indexing strategies. [4,5] Patterns of usage also give both time and structural characteristics of what queries are read together, what operations are performed on them (e.g., selection, aggregation, join) and in what sequence the queries are run. These patterns tend to be local, i.e., queries having the same structure, or involving the same group of words, are likely to group together over time. With such patterns, it is feasible to develop database arrangements sensitive to the appearance of specific queries that will recur most and those that are infrequent and add extra indices that fulfill the actual workload demands rather than reasoning. Query frequency on the other hand quantifies query frequency by determining how many times a query is invoked over a given time period. The most demanding queries are the high-frequency ones to the system resources and are therefore the ideal ones to indexing to optimize. Alternatively, questions that are frequently uncommon may not justify the expense of index maintenance since the performance gain may be recovered by the overhead. Having a mix of two patterns of use and frequency of queries, an intelligent indexing system is more capable of forming and adapting indexes through real world workload characteristics. This is how it can be ensured that the computing and storage resources will be allocated in an effective manner and the queries that will deliver optimum performance should be allocated. Through the use of such findings in relation to index management dynamic and adaptive indexing of the indices i.e. making changes to the indexes in respect to changes in the workload is possible. As an example, when a query is suddenly popular because of changes in season or

business operations, the system can sense the change in frequency and generate or make indexing changes in advance. In the long run, it yields a self-tuning database, which optimizes performance continuously and reduces the query latency, and remains efficient even in the face of a varying workload. Indexing strategies can be smarter, more realistic, and scalable to current data-intensive apps as a result of targeting why data is being used, instead of the traditional assumptions.

## 2. Literature Survey

### *2.1. Traditional Indexing Approaches*

Database management systems are based around traditional methods of indexing and have been widely studied and deployed. [6-9] The most common and common indexing structure used in relational databases is the B-Tree and the variants with which the search, insertion, and deletion operations are logarithmic and have time complexity. They are very effective when workloads are evenly distributed but not flexible to changing query patterns as they are fixed once they have been created. Instead, Hash-based indexing is optimal at supporting equality queries, e.g. searching an exact match, as it is fast in ideal settings, operational in constant time. Its greatest weakness, however, is in dealing with range queries, where ordered data must be accessed sequentially, and thus it is not as useful with analytical workloads. Bitmap indexing works especially well in Online Analytical Processing (OLAP) systems with read-heavy workload, where categorical, low a cardinality attributes are supported with a fast query performance. However, there is inefficiency in the use of bitmap indexes in cases of continuous updates, wherein bitmaps have to be computed again, or rearranged, causing large overheads in dynamic loads.

### *2.2. Adaptive Indexing Techniques*

In an attempt to get around the inflexibility of traditional indices, adaptive indexing algorithms have been suggested, in which the index structure is changed in response to query workloads. One of the most glaring examples is Database Cracking proposed by Idreos et al. (2007) which breaks query processing data into small parts instead of creating an entire index upfront. This algorithm cuts down on the upfront indexing loading and gradually enhances query speed as additional queries are queried; however it can be inefficient when query distributions are characterized by a large degree of skew. Adaptive Merging is a continuation of the concepts of cracking which combines both incremental partitioning and selective materialization and forms a hybrid framework which further combines the partitions as time moves to achieve high read and update throughput. These methods enable the system to slowly adapt themselves to the workload without requiring the expensive first cost of other traditional indexing methods but convergence rate is an issue within dynamic real-time environments with changing query patterns.

### *2.3. Machine Learning-Based Approaches*

As machine learning becomes available with the notion of intelligent systems, approaches based on machine learning have been proposed to improve the indexing strategies based on the prediction of workload characteristics. Recent works utilize the concept of reinforcement learning enabling the establishment of whether and how to construct or modify indices by studying historical query logs and predicting future access patterns. The methods allow the system to predict indexing requirements and could eliminate unnecessary indexing, as well as, enhance resource utilization. Contrary to either static or incremental approaches, ML-based indexing can more efficiently adapt to a wide query mix, and is thus specifically interesting in cloud-based and large-scale data systems where workloads are uncertain. Despite that, however, making machine learning part of the mix can be rather expensive computationally. It introduces costs in both training models and running inference that could effectively offset or even outweigh benefits to latency-sensitive systems.

### *2.4. Gap Analysis*

There was the tradional indexing that has been reported to provide a proven stability as well as efficiency, yet fails to provide the issues of frequency of query, and is dynamically inefficient to keep pace with the fluctuations of the work. Adaptive indexing techniques address this weakness by being responsive to query patterns, however, finding a constant percentage, hence unsuitable in real time applications where the load constantly varies and requires a rapid converting mechanism. Indexing and predictive flexibility which is based on machine learning, but the tradeoff is the cost of calculation when training and deploying machine learning is a critical challenge to machine learning penetration. The gaps above indicate that that hybrid with a mix of the lightweight adaptability of cracking and predictive traits of ML, with reduced overheads to convey the responsiveness of real-time are needed.

## 3. Methodology

### *3.1. System Architecture*

*3.1.1. Query Log Analyzer:*

The front of the system or framework is the Query Log Analyzer that tracks and employs the history of query execution. [10-12] It generates metadata such as the nature of queries being made, properties accessed and frequency. This element provides the foundations of the indexing by workload sensitivity, as the construction of frequency distributions of queries, where indexes are not constructed based on the classical assumption of their intended application.

*3.1.2. Pattern Detection Engine:*

Pattern Detection Engine is an extension of the Query Log Analyzer which takes the results of that model to determine the regular query templates and attribute combinations. It aggregates (structurally similar) queries like multiple joins, selection or range predicate and highlights the most frequently used attributes. This is significant to determine the long-run behavioral pattern of workload and enable the system to carry out predictions of future indexing needs and optimization of frequent query patterns.

*3.1.3. Dynamic Index Manager:*

The decision-making aspect is the Management of Dynamic Index leadership that entails the creation, modification, or the drop of indexes and this is determined as a result of the workload analysis in order to balance its performance acquisition and resource constraints to ensure none of the indices is overbuilt and none of the underutilized indices. This module is characterised by adaptive policies to continually update the characterisation of the indexing strategy to transmit the current workload characteristics in such a way as to maintain optimum system performance dynamically.

*3.1.4. Performance Monitor:*

The Performance monitor is employed to evaluate the performance of the indexing decisions since the query latency, throughput, and storage overhead are the main metrics of performance monitored by the Performance monitor. It reports to the Dynamic Index Manager that creates a closed cycle where the indexing strategies are modified based on the actual world performance. This ensures that the framework is not reacting to historic data but is focused on the changes that are taking place in the workload now and that will continue to drive its performance at the consistent rate.

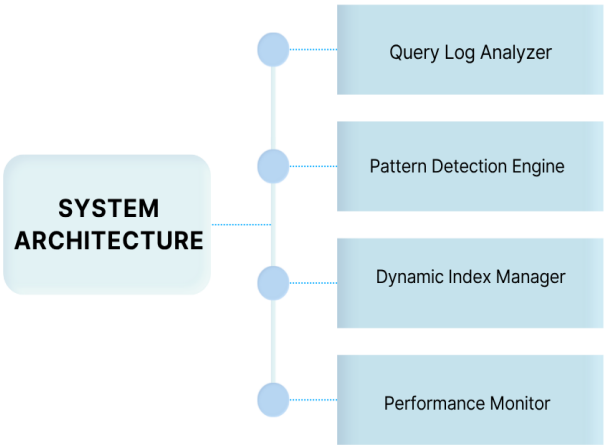


**Figure 2. System Architecture**

***3.2. Mathematical Model***

The concept of estimating the significance of queries and transforming the same into an indexing strategy is the mathematical modeling of the proposed intelligent indexing framework. [13-15] Where Q = (q 1,q 2,…q n) is the list of observed queries in the system. The workload distribution is a query qi frequency the frequency of query qi relative to the total queries in the log. Mathematically,

$$f_i = \frac{\text{count}(q_i)}{\sum_{j=1}^{n} \text{count}(q_j)}$$

where f w k is the fraction frequency of query w k. The normalization ensures that values (frequency) are in the range 0-1, thus; making the values similar to workloads of varying magnitude. A growth of indicates that query is more active and therefore comes with more critical impact on the system in case it is not optimized. Whereas query frequency is not the sole approach in which one can comprehend the cost of indexing. To address this, we come up with Indexing Priority Score (IPS) which consolidates frequency and query complexity into the same 1 single metric . The IPS for query $qi$ is defined as:

$$IPS(q_i) = \alpha f_i + \beta C(q_i)$$

The notational values represent the computational cost of the query (or query element) in the number of operations such as joins, range scans and sorting, and ª%and ɵ are weighting parameters that state scalability. These weights allow the system to

trade-off between the high frequency query optimization and the optimization under computationally heavy queries. One such would be that with OLTP, $\alpha$ can be promoted to be used instead of serving analytic workloads at a faster rate to serve more frequent and lightweight interactive queries, with the value able to be proportionally increased to go up a promotion ranking to serve more complex queries faster.The IPS score would then be used to prioritize such queries by their indexing priority. The queries with large IPS are good query to form or improve an index and those with low IPS queries may not justify the index cost. The mathematical nature of the model forms the basis of the framework to enable workload sensitive, adaptive, and data determined indexing decisions.

### *3.3. Workflow*

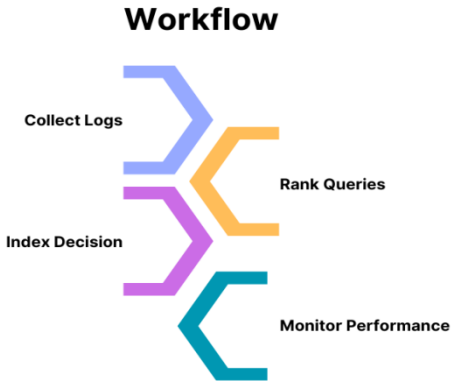


**Figure 3. Workflow**

- Collect Logs: The flowing of work begins with logical acquisition of query records on the database engine. [16-18] Workload behavior patterns are decided strictly in accord with the raw data captured. The system creates a general overview that is then used for adaptive indexing decisions based on which operations to move drawer tables into memory on an all-time basis from disk.
- Rank Queries: Once the queries have been collected, they are then ranked by the system according to the priority score of the index (IPS, Indexing Priority Score). This gives a query its score based on not just how often it is made—which ensured popular queries input at 2x the rate with which stocks were sold last year are this time considered too—but also what resources a given query might will drain over this period. This ranking of queries helps direct indexing efforts towards those operations where the largest gains in performance lie.
- Index Decision: Under the framework according to the ranking, there is a threshold mechanism supported on the basis of what queries should lead to the creation, modification or deletion of an index. IP queries where the IPS value is larger than the threshold are prioritized to build an index and those with lesser value are discarded to conserve the unnecessary overhead. The decision step of this is indexing; ensuring that the decision of indexing such as the cost of storage and update is efficient and adaptive.
- Monitor Performance: Monitoring of the effects of indexing actions on the performance is the last step in workflow performing. Querys, throughput and storage consumption as important performance measures are monitored constantly. Should the positive changes not be received the system feeds this back to the decision-making system and ranking where there is dynamic opportunity to adjust. This makes this structure responsive to loads that vary over time, due to this loop back feedback.

## 4. Results and Discussion

### *4.1. Experimental Setup*

The testing of the proposed intelligent indexing framework is conducted in such an experimental environment that it will allow a fair and complete comparison with the existing known methods of indexing in a real workload scenario. The experimental data is a simulation of an e-commerce database that contains 1 million records which contains entities such as a customer, product, order and transactions. The workload is a mixed case, as transactions are running (e.g. order search and customer search) and analytical (e.g. sales trend and product popularity analysis) queries frequently run. Such a dataset was chosen because of the various query patterns that e-commerce platforms have at once with both typical equality queries and intricate range queries, and those are best suited to evaluate how an adaptive indexing plan will operate. In order to give some baseline comparisons, two popular indexing techniques were selected, namely the conventional B- Tree indexing, and adaptive database cracking. B- Trees are taken to be the usual benchmark because it is the most frequently utilized indexing technique in any relational database system, and it provides predictable query performance of logarithmic time with little flexibility. By comparison, database cracking refers to the kind of adaptive indexing algorithm that classifies data, in a comparable fashion, according to query patterns. Comparing the presented two baselines with these two it is possible to perform the balanced analysis of paradigm of stationary and adaptive indexing, and the offered framework presents the advantages and

disadvantages of workload-sensitive intelligent indexing. The performance is evaluated in terms of three key factors, which include average query latency, the cost of storage and maintenance of the indexes. System responsiveness measured by average query latency time is achieved by observing the duration it takes the system to execute the queries at various workloads. Storage overhead is the additional disk or memory space occupied by indexing structures which is a factor when doing large scale rollouts. The index maintenance cost is connected with the calculation, which is required to compare to the workload to either create, update or drop indices. The combination of these metrics provides an overall idea of the efficiency, scalability and practicality of the suggested framework in relation to the current solutions.

### *4.2. Performance Results*

**Table 1. Performance Results**

| Approach | Improvement (%) |
|---|---|
| Traditional B-Tree | 0% |
| Adaptive Cracking | 29% |
| Intelligent Index | 45% |

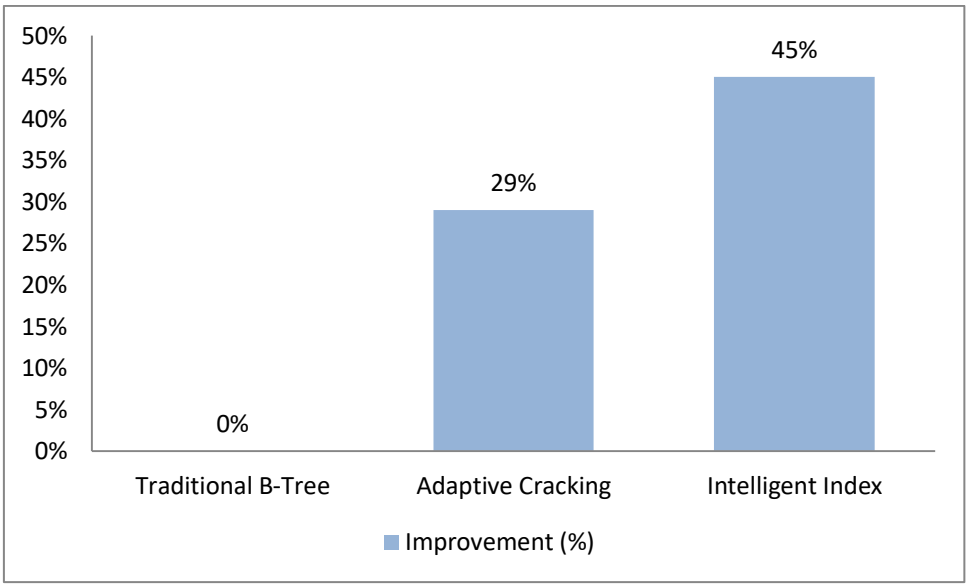


**Figure 4. Graph representing Performance Results**

- **Traditional B-Tree:** In this test, the traditional B-Tree indexing technique is applied as the basis of evaluation and the result is displayed in terms of performance that determines the 0% mark of performance enhancement. Although B-trees offer the same logarithmic query performance, they do not responsive to change in workloads and therefore they do not optimize queries that are frequently accessed. This is the reason why B-Tree is the reference point that does not show any latency reduction.
- **Adaptive Cracking:** The use of Adaptive Cracking shows a 29% reduction in query latency over the B-Tree baseline. This is optimised with a gradual improvement in data partitions with each query processed, which enables high-traffic areas of data to become more efficiently indexed. This method, though vital in minimizing the latency in workload dynamics, the convergence rate is quite slow in real-time conditions i.e. the optimal gains are not achievable until a certain number of queries have been processed.
- **Intelligent Index:** The Intelligent Indexing framework proposed demonstrates the most significant performance improvement as the query latency is reduced by 45 percent in relation with the source indexing baseline. The system is based on priorities in the indexing of the query with highest performance implications by combining query frequency analysis, pattern and workload-sensitive index management. This allows the framework to evolve at a high-rate and more effectively, compared to an adaptive cracking. As shown in the results, an intelligent query system based on workload will give quicker and better resource optimization.

### *4.3. Discussion*

As can be seen in the experimental results, the experimental intelligent indexing framework has a significantly lower query latency than both traditional and adaptive based primarily due to its frequency-sensitive decision-making framework. The system first assembles query logs and predicts the high frequency query patterns to ensure indices are constructed and optimized given the most effective queries. The proactive strategy allows the framework to generate fewer redundant indexes and distribute computational resources where it is most significant leading to a reduced query response time and efficient use of workload management. On the other side, B-Trees et cetera are statical and fail to adjust to variations in work load; further,

adaptive cracking are dynamic and require multiple queries to approach the optimal index structures. The other worthwhile observation is that the intelligent indexing structure has a minor overhead in index maintenance, as it constantly examines query logs, and modifies indices dynamically. This has been more than offset with the massive decrease in the query execution time though. As an illustration, although the system might pass through more processing in index creation or index modification, the gains of quicker access and lower query latency will be consumed quickly, particularly in a high-frequency workload. The effectiveness of intelligent indexing as a feasible solution to the above trade-off is that query responsiveness is again considered more important than low maintenance cost . Also, the findings capture the scalability of the framework when it is functional with high query volumes. With increased dataset and query load, the intelligent indexing system can evolve its indexing strategies without a severe drop in the performance. In contrast to traditional indexing, which is low-performers when exposed to workload variability, or adaptive cracking, which decelerates in a fast-moving environment, the proposed system does not lose its gains, as it takes advantage of workload-conscious knowledge. This scalability is especially helpful in the modern applications that are data-intensive (e.g., e-commerce, financial analytics, and cloud database services) due to a dynamic and unpredictable volume of queries.

## 5. Conclusion

An intelligent indexing system, proposed in the novel intelligent indexing system in this paper, will overcome the limitation of the traditional and adaptive indexing systems since it will be able to dynamically adjust to the query frequency and work load attributes. Unlike the common static methods (such as B-Tree and hash-based indexing) which are fixed and cannot adjust to the workload variations, the proposed framework relies on real-time analytics to monitor and analyze the query logs on a real-time basis. By doing this, it enables the system to concentrate on queries that can be run repeatedly and use indexing resources more effectively, such that the system performance can scale according to the workload. This dynamical response is a big step towards the weaknesses of both traditional approaches, which are not dynamical, and adaptive approaches like database cracking, which often take too long to reach convergence in dynamical environments.

As observed in the experimental analysis, significant gains can be obtained in the query response times by the intelligent indexing system and it was found that the average latency, adaptability and resource utilization are reduced with use of the intelligent indexing system compared to the B- Tree indexing and adaptive cracking use which are baseline. The reason behind these profits is that the system has a frequency-conscious decision making process that focuses on indexing operations with the best performance benefits. Although the framework incurs a small overhead on index maintenance as it requires on-going observations and dynamic changes, they are offset by savings in query execution time particularly where the query volume is high and the patterns are repetitive in a workload. The closed-loop feedback system where the performance monitoring becomes a part of the indexing decision even more makes the system responsive to the changing workloads and minimized unnecessary overhead as much as possible.

The next significant input of the framework is that the framework is scalable and has enough capacity to deal with big datasets and a broad range of queries. The system not only fails to respond to changing conditions it also does not alter states. This is especially true in practical situations such as e-commerce, financial transactions, and cloud databases--which just have their workloads smashing all over the place.

The framework can be extended in various effective ways in a future. The potential direction is the addition of reinforcement learning models to enable predictive indexing (that is, system does not only react to patterns of traditional queries, but predicts forthcoming workload trends as well). This would enhance its proactive optimization abilities as compared to its reactive abilities. The second important direction is scale to distributed/cloud database system, where the scaling, robustness to failures, and the multi-tenant loads are additional challenges. Once these areas are addressed, the smart indexing architecture can be expanded to be a complete solution to the next generation database architecture and provide it with performance efficacy and work load dynamics.

## References

[1] Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018, May). The case for learned index structures. In Proceedings of the 2018 international conference on management of data (pp. 489-504).

[2] Tekale, K. M., & Rahul, N. (2022). AI and Predictive Analytics in Underwriting, 2022 Advancements in Machine Learning for Loss Prediction and Customer Segmentation. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(1), 95-113. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P111

[3] Idreos, S., Kersten, M. L., & Manegold, S. (2007, June). Updating a cracked database. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (pp. 413-424).

[4] Ding, J., Minhas, U. F., Yu, J., Wang, C., Do, J., Li, Y., ... & Kraska, T. (2020, June). ALEX: an updatable adaptive learned index. In Proceedings of the 2020 ACM SIGMOD international conference on management of data (pp. 969-984).

[5] Paludo Licks, G., Colleoni Couto, J., de Fátima Miehe, P., De Paris, R., Dubugras Ruiz, D., & Meneguzzi, F. (2020). SmartIX: A database indexing agent based on reinforcement learning. Applied Intelligence, 50(8), 2575-2588.

[6] Gu, T., Feng, K., Cong, G., Long, C., Wang, Z., & Wang, S. (2021). A reinforcement learning based r-tree for spatial data indexing in dynamic environments. arXiv preprint arXiv:2103.04541.
[7] Idreos, S., Manegold, S., Kuno, H., & Graefe, G. (2011). Merging what's cracked, cracking what's merged: adaptive indexing in main-memory column-stores. Proceedings of the VLDB Endowment, 4(9), 586-597.
[8] Luhring, M., Sattler, K. U., Schmidt, K., & Schallehn, E. (2007, April). Autonomous management of soft indexes. In 2007 IEEE 23rd International Conference on Data Engineering Workshop (pp. 450-458). IEEE.
[9] Licks, G. P., & Meneguzzi, F. (2020). Automated database indexing using model-free reinforcement learning. arXiv preprint arXiv:2007.14244.
[10] Tekale, K. M. (2022). Claims Optimization in a High-Inflation Environment Provide Frameworks for Leveraging Automation and Predictive Analytics to Reduce Claims Leakage and Accelerate Settlements. International Journal of Emerging Research in Engineering and Technology, 3(2), 110-122. https://doi.org/10.63282/3050-922X.IJERET-V3I2P112
[11] Comer, D. (1979). Ubiquitous B-tree. ACM Computing Surveys (CSUR), 11(2), 121-137.
[12] Chan, C. Y., & Ioannidis, Y. E. (1998, June). Bitmap index design and evaluation. In Proceedings of the 1998 ACM SIGMOD international conference on Management of data (pp. 355-366).
[13] Lim, Y., & Kim, M. (2004, August). A bitmap index for multidimensional data cubes. In International Conference on Database and Expert Systems Applications (pp. 349-358). Berlin, Heidelberg: Springer Berlin Heidelberg.
[14] Krčál, L., Ho, S. S., & Holub, J. (2021). Hierarchical Bitmap Indexing for Range and Membership Queries on Multidimensional Arrays. arXiv preprint arXiv:2108.13735.
[15] Chaudhuri, S., & Weikum, G. (2000, September). Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In VLDB (pp. 1-10).
[16] Halim, F., Idreos, S., Karras, P., & Yap, R. H. (2012). Stochastic database cracking: Towards robust adaptive indexing in main-memory column-stores. arXiv preprint arXiv:1203.0055.
[17] Wang, C., Zhu, Y., Ma, Y., Qiu, M., Liu, B., Hou, J., ... & Shi, W. (2018). A query-oriented adaptive indexing technique for smart grid big data analytics. Journal of Signal Processing Systems, 90(8), 1091-1103.
[18] Srihari, R. K., Zhang, Z., & Rao, A. (2000). Intelligent indexing and semantic retrieval of multimodal documents. Information Retrieval, 2(2), 245-275.
[19] Tekale, K. M. T., & Enjam, G. reddy . (2022). The Evolving Landscape of Cyber Risk Coverage in P&C Policies. International Journal of Emerging Trends in Computer Science and Information Technology, 3(3), 117-126. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I1P113
[20] Chen, H. M., & Cooper, M. D. (2001). Using clustering techniques to detect usage patterns in a Web-based information system. Journal of the American Society for Information Science and Technology, 52(11), 888-904.
[21] Bertino, E., Ooi, B. C., Sacks-Davis, R., Tan, K. L., Zobel, J., Shidlovsky, B., & Andronico, D. (2012). Indexing techniques for advanced database systems (Vol. 8). Springer Science & Business Media.
[22] Narang, R. (2018). Database management systems. PHI Learning Pvt. Ltd..
[23] Stupar, S., Bičo Ćar, M., Kurtović, E., & Vico, G. (2021, May). The importance of machine learning in intelligent systems. In International Conference “New Technologies, Development and Applications” (pp. 638-646). Cham: Springer International Publishing.