Data Partitioning using QD Tree

Mahima Dewangan
Mtech, Software Systems
Dhirubhai Ambani University
Gandhinagar , India
202411037@dau.ac.in

Supervisor : Prof.Minal Bhise

***Abstract*—This project explores the use of the QD-tree (Query-data routing tree) as a workload-aware partitioning technique in database systems. The core objective is to construct partition layouts informed by historical query workloads, thereby optimizing query routing and data access efficiency. By leveraging the QD-tree's ability to adapt to past query patterns, the study aims to assess how effectively such partitions support future query workloads. Experimental evaluation is conducted using range-based queries from the TPC-H benchmark, with implementation in Python.**

***Index Terms*—QD-tree, Partitioning, Workload-aware Optimization, Query Routing, TPC-H Benchmark, Database Systems**

I. Introduction

A QD-tree (Query-data routing tree) is a special kind of binary tree designed specifically for query workload-based partitioning and routing. It is used in database systems to create intelligent, workload-aware partitions of data and to route incoming queries to the most relevant partitions efficiently. Each node in a QD-tree represents a region of the data space, and the leaves of the tree represent the actual partitions that contain the data. The key difference in construction lies in how the tree is built—splits at each node are guided by query workload characteristics rather than just data values, making it highly adaptive to historical access patterns.

What sets the QD-tree apart from a traditional binary tree is its ability to learn from query behavior and adapt its structure accordingly. While a standard binary tree follows a fixed splitting rule based on data keys (e.g., less than or greater than a median), it is completely unaware of how often or in what patterns the data is accessed. This often leads to inefficient partitioning in scenarios where query distribution is skewed. In contrast, a QD-tree dynamically adjusts its splits to reflect the density and locality of past queries, allowing it to route incoming queries more precisely and avoid scanning irrelevant partitions. This workload-awareness is something a binary tree cannot achieve, making the QD-tree significantly more effective for optimizing range-based analytical workloads.

II. Objective

The objective of this project is to investigate the efficacy of QD-tree (Query-data routing tree) as a workload-aware partitioning strategy for database systems. The approach focuses on leveraging historical query workload information to construct data partitions that are better aligned with observed access patterns. By doing so, the project aims to enhance query performance by reducing the volume of irrelevant data accessed during query execution.

III. Methodology

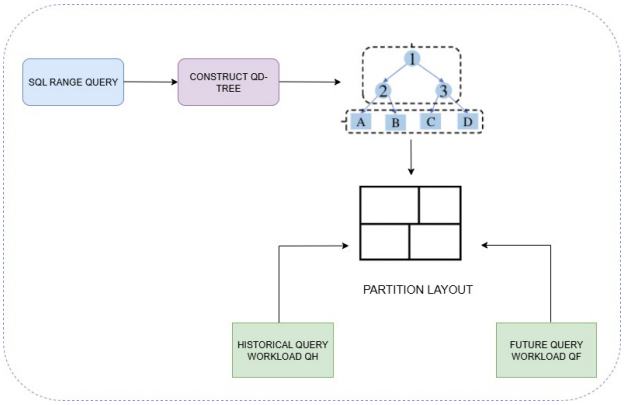

Fig. 1. Partition layout generated by QD-tree

A. Workload Analysis

The methodology adopted in this study focuses on constructing a workload-aware partitioning scheme using the QD-tree (Query-data routing tree) based on historical query patterns. The workload under consideration consists of range-based queries operating on two specific attributes of the TPC-H benchmark dataset.

In the initial phase, the historical query workload is analyzed to identify the distribution of queries.

B. QD-Tree Construction and Partitioning

Based on the workload analysis, a QD-tree is constructed where each internal node represents a decision split, and each leaf node corresponds to a distinct region of the data space. To ensure that partitions are practical for storage and query performance, a minimum size constraint of 128 MB is imposed on each partition during tree construction.

Following the construction of the QD-tree, a corresponding partition layout is generated. Each historical query is then routed to the relevant partitions based on its range predicates. This routing mechanism significantly enhances query efficiency by allowing the system to bypass partitions that fall outside the query's scope, thereby reducing unnecessary data access.

C. Evaluation on Future Workload

To evaluate the robustness of the generated partition layout, a future query workload—composed of similar range-based queries—is applied to the same partition structure. This enables a comparative assessment of whether the workload-informed partitions derived from historical queries support future query execution or not.

D. Experimental Setup

The experimental setup for this study includes a system equipped with 16 GB of RAM, an Intel Core i5-10505 processor (3.19 GHz), and the TPC-H benchmark dataset. All implementations are carried out in Python within a Jupyter Notebook environment, facilitating interactive development and performance monitoring of the QD-tree-based partitioning and query routing system.

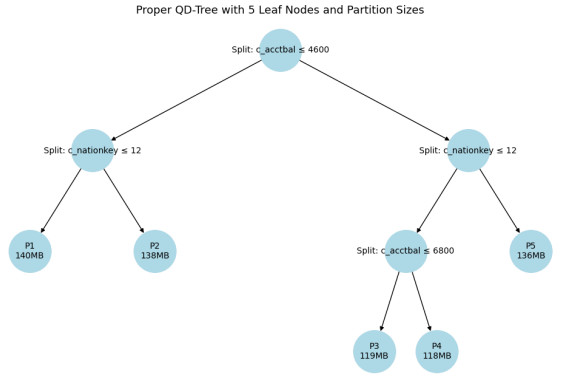

Fig. 2. QD-tree with partition size

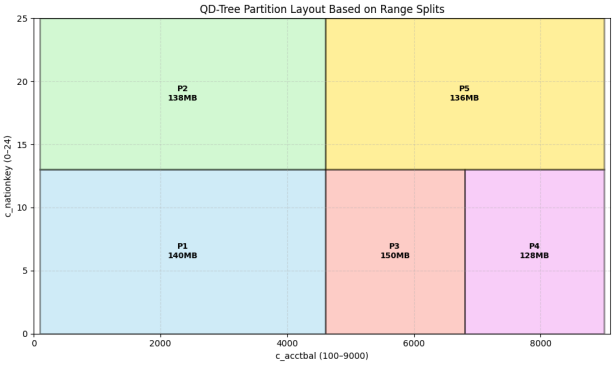

Fig. 3. Partition layout generated by QD-tree

IV. Results and Observations

The experimental results indicate that the partition layout generated using the QD-tree method performs effectively for the historical workload it was trained on. Queries from the historical set are routed precisely to relevant partitions, significantly reducing the amount of data scanned and minimizing I/O overhead.

However, when the same partition layout is applied to a similar future query workload, the performance degrades. The queries are less effectively routed, resulting in increased partition accesses and greater overall data scanned. This suggests that the QD-tree partitioning is highly sensitive to the specific distribution and range of the training workload and may not generalize well to future workloads, even if they appear similar on the surface.

This behavior is evident from the comparative table of partition sizes accessed by historical queries versus those accessed by future queries.

Fig. 4. Partition Layout on QH

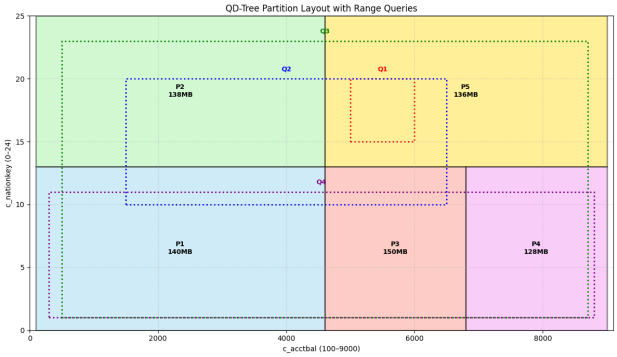

Fig. 5. Partition Layout on QF

Query	Touched_Partitions	Total Partition Size (MB)
q1	['P5']	136
q2	['P5']	136
q3	['P2']	138
q4	['P3']	150

Fig. 6. Partition Layout on QH

Query	Partitions Touched	Total Size (MB)
Q1 (Red)	P5	136
Q2 (Blue)	P1, P2, P3, P5	564
Q3 (Green)	P1, P2, P3, P4, P5	692
Q4 (Purple)	P1, P3, P4	418

Fig. 7. Partition Layout on QF

V. Discussion

Existing partitioning methods, typically produce rectangular partitions. While this simplifies partition management and query routing, it introduces inefficiencies in cases where a single partition covers both frequently and rarely accessed regions. For example, partitions such as P5. This degrades query performance due to increased I/O and wider data scans.

To address this, the partitioning strategy should aim to explicitly cover rarely queried regions with dedicated, smaller partitions. By isolating these cold regions, the overall size of the frequently accessed partitions (e.g., P1, P2, P3, P4, P5) can be reduced.

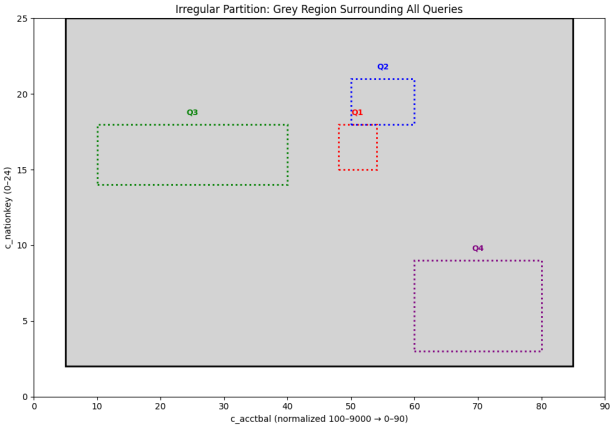

Fig. 8. Irregular shape partitioning on QH

VI. Conclusion

The QD-tree-based partitioning method demonstrates strong performance when aligned with historical query workloads, effectively reducing unnecessary data scans through intelligent query routing. However, its effectiveness declines when applied to future workloads, revealing its sensitivity to shifts in query patterns. Additionally, the use of rectangular partitions often leads to inefficiencies, as frequently accessed partitions may encompass rarely queried regions. These findings highlight the need for more adaptive partitioning strategies that can dynamically adjust to evolving workloads.

VII. Future Work

Future efforts should focus on designing partitioning approaches that not only optimize for historical query workloads but also generalize effectively to future access patterns. This calls for adaptive techniques capable of evolving with workload changes, ensuring consistent query performance and improved partitioning resilience in dynamic data environments.

References

[1] Y. Zhao, X. Zhang, Y. Jiao, B. Cui, and C. Wang, *"Data Partitioning Meets Workload Variance: Towards Adaptive Query Processing,"* in Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2021, pp. 279–292.

[2] C. Yang, J. Ding, Y. Xu, J. Tang, C. Wang, and B. Cui, *"QD-Tree: Learning Data Layouts for Big Data Analytics,"* in Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2020, pp. 1443–1458.